Universidad de las Ciencias Informáticas

Facultad 7

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

Implementación de los procesos de observación y administración del módulo Emergencias del Sistema de Información Hospitalaria alas HIS

Autores: Osmany Castro Espinosa

Zadys Carrasco Medina

Tutores: Lic. Dainerys Castañeiro Rodríguez

Ing. Juan Manuel García Orduñez

Ciudad de la Habana, Junio de 2010
"Año 52 de la Revolución"

Datos de contacto

Lic. Dainerys Castañeiro Rodríguez

Graduada de Licenciatura en Ciencias de la Computación en el año 2004 en la universidad Central de las Villas. Actualmente se desempeña como profesor universitario en la Universidad de las Ciencias Informáticas (UCI) donde ha impartido asignaturas como Sistemas de Bases de Datos, Ingeniería de Software y Gestión de Software. Posee categoría docente de Instructor. Desde que se incorporó a la universidad ha estado vinculada a actividades productivas en el área hospitalaria, actualmente se encuentra desempeñando el rol de analista en los módulos Citas, Consulta Externa y Hospitalización del Sistema de Información Hospitalaria perteneciente al Departamento Gestión Hospitalaria del Centro especializado en soluciones integrales para Informática Médica (CESIM).

Correo electrónico: dainerysc@uci.cu

Ing. Juan Manuel García Orduñez

Instructor recién graduado en el año 2009 de Ingeniero en Ciencias Informáticas en la Universidad de las Ciencias Informáticas. Profesor vinculado a la Facultad 7 y miembro del Departamento de Sistema de Gestión Hospitalaria.

Correo electrónico: jmgarcia@uci.cu

RESUMEN

En la actualidad existen muchas instituciones hospitalarias no cuentan con un sistema que maneje todo el flujo de información que se lleva a cabo en el área de Emergencias. El objetivo de este trabajo es implementar los procesos observación y administración del módulo Emergencias del Sistema de Información Hospitalaria alas HIS, para facilitar la gestión de información en esta área de las instituciones hospitalarias.

Para su desarrollo se utilizaron las herramientas establecidas por el Departamento de Sistema de Información Hospitalaria. Para su análisis y diseño se utilizó la metodología RUP. Como lenguaje de programación orientado a objetos del lado del servidor, Java, Eclipse como Entorno de Desarrollo Integrado, PostgreSQL 8.3 como Sistema Gestor de Bases de Datos. Además se empleó Hibernate como herramienta ORM para la persistencia de los datos y el framework Seam para la lógica del negocio, entre otras tecnologías.

Con la incorporación de estos procesos al módulo Emergencias se pretende agilizar el proceso de atención al paciente a partir de los beneficios que ofrecen las tecnologías de la información. Esto permitirá registrar las evoluciones realizadas a los pacientes así como generar reportes de estadísticas, que podrán ser empleados para estudios posteriores. Brindará nuevos beneficios tanto al médico como al propio paciente.

ÍNDICE

INTRODUCCIÓN

Las Tecnologías de la Información y las Comunicaciones (TIC) han revolucionado la vida actual del hombre y la forma en que este utiliza el conocimiento. Estas constituyen la herramienta fundamental de inclusión de la sociedad en el amplio mundo de la informática, logrando a través de su empleo, alcanzar niveles superiores de eficiencia en la manipulación de la información y los recursos que se dispongan. [1]

La aplicación de las TIC se ha evidenciado durante años en la mayoría de los sectores sociales, debido a las necesidades, cada vez mayores, de simplificar y agilizar las tareas que estos realizan. Además, buscar formas más efectivas de obtener conocimiento aplicable a los retos que presenta el desarrollo de la humanidad.

Un sector de suma importancia a nivel mundial es el de la salud, el cual se caracteriza por mantener en constante movimiento grandes volúmenes de información, aspecto que dificulta su gestión en gran medida, y se complica con el crecimiento de la población y la prestación de servicios de mayor grado de complejidad. A raíz de estas dificultades se ha fomentado de manera progresiva la incorporación de sistemas de información en la salud, apoyados en las TIC. Estos facilitan los procesos internos, aumentan la calidad de atención de los pacientes y mejoran la labor o desempeño por parte de los trabajadores en los diferentes niveles de atención.

Desde los inicios los sistemas de información en la salud se desarrollaron en ambientes hospitalarios. Las primeras prácticas de informatización se enfocaron en los procesos administrativos, con énfasis en la gestión contable y financiera de los actos médicos. Posteriormente estas soluciones informáticas se orientaron a la información clínica centrada en el paciente como eje principal, brindando nuevas opciones al personal médico y al propio paciente.

Los Sistemas de Información Hospitalaria (SIH o HIS por sus siglas en inglés) son sistemas basados en computadoras y diseñados con el objetivo de mejorar la calidad y la eficiencia del trabajo en los centros de atención médica. Los mismos se encargan de almacenar, procesar, recopilar, recuperar y comunicar los datos obtenidos en la atención al paciente y en la labor administrativa de las instituciones hospitalarias. Por otra parte, permiten la optimización de los recursos humanos y materiales, minimizan los inconvenientes burocráticos que puede enfrentar el paciente y generan reportes e informes estadísticos que posibilitan la retroalimentación. [2]

Los HIS responden específicamente al funcionamiento de las diferentes áreas que conforman un centro hospitalario, las cuales mantienen una estrecha relación entre sí. Emergencias o como también se le suele llamar Urgencias o Cuerpo de guardia, es un ejemplo de estas áreas, y tiene como primicia

Introducción

la atención al paciente de forma inmediata, independientemente del nivel de gravedad que este presente.

En Emergencias existe el área de observación donde se evolucionan a los pacientes periódicamente, en intervalos de una hora aproximadamente y la información recolectada se registra de forma manual. Al aumentar el tiempo de estancia del paciente en el área, la cantidad de documentos clínicos que se generan crece considerablemente, lo que trae consigo dificultad a la hora de consultar los mismos con el objetivo de llegar a un diagnóstico acertado.

Por otra parte, al evolucionar los pacientes, los médicos pueden prescribir solicitudes de exámenes complementarios asociadas a los servicios de apoyo al diagnóstico. Estas, al igual que el resto de los datos registrados se realizan en formato duro, lo cual puede provocar la aparición de errores humanos en su llenado y la pérdida parcial o total de los mismos, pues dicho formato es susceptible al deterioro debido a la manipulación por parte del personal que lo consulta, así como, a la influencia de las condiciones físicas y ambientales de los locales donde se almacenan.

Las solicitudes de exámenes son enviadas a los respectivos servicios a través de mensajeros, al igual que los resultados arrojados por estas, lo cual provoca un aumento en la espera por parte del personal médico para evaluar los resultados, aspecto poco favorable teniendo en cuenta las necesidades del área.

Al mismo tiempo, el médico no tiene conocimiento de los medicamentos disponibles en la farmacia, situación que atenta contra la emisión de una correcta orden médica, pues existe la posibilidad de que se indiquen medicamentos con los que no cuente la institución hospitalaria.

La ausencia de una administración centralizada de los recursos, es otro de los problemas que enfrenta el área de Emergencias, debido a que se desconoce, en tiempo real, la disponibilidad de cupos para la ubicación de los pacientes recibidos que lo requieran.

Además, en esta área debido a los volúmenes de información que se generan diariamente se hace difícil su procesamiento, lo que trae consigo dificultades a la hora de obtener estadísticas actualizadas y fiables, elementos importantes para la realización de estudios y análisis de las enfermedades más comunes, investigaciones y la generación de reportes, así como para la toma de decisiones.

Las situaciones descritas anteriormente evidencian la necesidad de un software que posibilite la administración de los recursos y gestión de la información generada durante la atención al paciente en las áreas de observación.

Introducción

Por lo anteriormente expuesto, se define el siguiente ***problema a resolver***: ¿Cómo viabilizar la gestión de la información generada en la atención al paciente en observación y en la administración del área de Emergencias de las instituciones hospitalarias?

Este problema se enmarca en el ***objeto de estudio*** concerniente a los procesos de gestión de la información en el área de Emergencias de las instituciones hospitalarias y el ***campo de acción*** que comprende los procesos observación y administración del área de Emergencias de las instituciones hospitalarias.

Basado en esa idea se define el ***objetivo general:*** Implementar los procesos observación y administración del módulo Emergencias del Sistema de Información Hospitalaria alas HIS, que facilite la gestión de información en esta área de las instituciones hospitalarias.

Para dar cumplimiento al objetivo general antes propuesto, se definen las siguientes ***tareas a desarrollar:***

1. Evaluar las tendencias actuales de los Sistemas de Información Hospitalaria que incluyen la gestión de la información en el área de Emergencias.
2. Analizar los procesos de negocio relacionados con la atención al paciente en observación y la administración de los recursos en el área de Emergencias de las instituciones hospitalarias.
3. Asimilar la arquitectura y las pautas definidas por el Centro de Informática Médica para el desarrollo de sus aplicaciones.
4. Obtener los artefactos correspondientes a los flujos de trabajo "Análisis y Diseño", "Implementación" y "Pruebas".
5. Implementar las funcionalidades correspondientes a los procesos observación y administración del módulo Emergencias del Sistema de Información Hospitalaria alas HIS, garantizando la integración con el resto de los módulos.

El presente documento se encuentra estructurado en cuatro capítulos, el primero de ellos, "**FUNDAMENTACIÓN TEÓRICA**", ubica al lector en el ambiente de desarrollo del módulo de Emergencias, justificándose las tendencias, tecnologías, metodologías y herramientas que fueron utilizadas para el desarrollo de los procesos seleccionados. Seguidamente el segundo capítulo, "**DESCRIPCIÓN DE LA ARQUITECTURA**", contiene la fundamentación de la arquitectura, la estrategia de integración, los elementos que garantizan la seguridad en el sistema y la especificación de los estándares y estilos de codificación a utilizar.

Introducción

En el tercer capítulo “**DESCRIPCIÓN Y ANÁLISIS DE LA SOLUCIÓN PROPUESTA**” se implementan las clases y subsistemas en términos de componentes. Se presenta la propuesta de solución para lograr una gestión más eficiente de los procesos hospitalarios asociados al área en cuestión. El cuarto y último capítulo “**MODELO DE PRUEBA**” se caracteriza el método de prueba a utilizar y se describen los casos de prueba.

Capítulo I. FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se abordan conceptos básicos relacionados con el dominio del problema a resolver. Además, se realiza un estudio de la situación a nivel mundial de los sistemas de información hospitalaria existentes que incluyen un módulo de Emergencias y se definen las tecnologías, herramientas y metodologías actuales, utilizadas para el desarrollo de la solución propuesta.

1.1 Conceptos básicos relacionados con el dominio del problema

Para facilitar la comprensión de dicha investigación se deben entender primeramente los términos que se emplean en las instituciones hospitalarias, motivo por el cual se especifican algunos conceptos que no son comunes en el lenguaje cotidiano.

Emergencias o Urgencias como también se le conoce, es el área donde se reciben los pacientes que necesitan atención médica de forma inmediata. En la misma se toman conductas a seguir dependiendo del estado en que estos se encuentren.

Una de las conductas a seguir es *dejar en observación,* con esta se garantiza la estancia del paciente en el área durante un período máximo de 48 horas, con el objetivo de estudiar su cuadro clínico, y su reacción ante distintos tratamientos.

Cuando se toma la conducta de dejar el paciente en seguimiento, este se ubica en un *cupo* determinado, lo cual puede ser una cama, camilla, silla, banqueta, silla de ruedas, o cualquier otro sitio u objeto habilitado para su permanencia en el período que es atendido.

Durante la estadía del paciente en el área de observación el médico tratante se encarga de realizar *evoluciones médicas,* donde registra: el estado clínico, los signos vitales, la impresión diagnóstica, la indicación de análisis complementarios y de órdenes médicas que pueda necesitar el paciente.

Las *órdenes médicas* se componen de medicamentos, tratamientos u órdenes de dietas y deben ser cumplidas en un corto período de tiempo.

1.2 Sistemas automatizados existentes vinculados con el campo de acción

En la actualidad existen diversas soluciones informáticas destinadas a optimizar recursos y mejorar la calidad en la prestación de servicios asistenciales. A partir de un estudio realizado, se identificaron sistemas que permiten la gestión de la información en el área de Emergencias los cuales se describen a continuación.

Capítulo I. Fundamentación Teórica

CNT pacientes HIS

Aplicación desktop diseñada de forma modular y se centra en la Historia Clínica como principal registro de todos los actos médicos realizados al paciente. Está conformada por doce módulos, de ellos 4 son administrativos y 8 están relacionados con la atención al paciente.

Permite una integración total de la información recopilada y generar reportes y análisis asociados a esta a través de herramientas avanzadas. Además, brinda la posibilidad de crear el plan de acción o de tratamiento compuesto por órdenes de medicamentos e indicaciones de suministración.

Desde el módulo de Historias Clínicas Dinámicas Hospitalarias se pueden crear órdenes médicas donde se prescriben medicamentos, se solicitan transfusiones al banco de sangre y se indican instrucciones al equipo de enfermería. Este módulo tiene una estrecha relación con el de Central de Urgencias y/o Emergencias, aquí se aplica un modelo de colas de urgencias, que direcciona al paciente hacia la atención dependiendo su hora de llegada, y nivel de gravedad, de tal manera que el personal asistencial recibe la orden de atención en línea. [3]

CNT pacientes HIS tiene soporte para una arquitectura cliente servidor. Además, funciona haciendo uso de los sistemas gestores de base de datos: Oracle, SQL Server, Sybase.

SIGHO

Es una aplicación web que facilita la gestión de las actividades que se realizan en las instituciones hospitalarias. Permite realizar registros individuales alrededor de la historia clínica electrónica en cada uno de los módulos que lo componen.

Tiene como beneficios mejorar la atención a los usuarios en los servicios médicos, establecer un banco de información estadístico, alinear la tecnología de información de forma estratégica dentro de los hospitales y fortalecer el modelo de gerencia hospitalaria. [4]

Se apoya de estándares internacionales para el diagnóstico de enfermedades y realización de procedimientos médicos, tales como el CIE-10 y CIE9CM. Como gestor de base de datos utiliza SQL Server 2000.

Está compuesto por 14 módulos, 2 administrativos y 12 relacionados con la atención al paciente, dentro de este último se encuentra Urgencias el cual brinda funcionalidades relacionadas con la creación de indicaciones médicas, solicitudes de servicios auxiliares, recetas médicas y notificación de diagnósticos. También permite consultar expedientes y resúmenes clínicos.

Capítulo I. Fundamentación Teórica

MediSys®EM

Aplicación informática diseñada con el objetivo de agilizar las actividades médicas que se realizan en el área de emergencias de las instituciones hospitalarias. Permite registrar notas médicas, impresiones diagnósticas, referencias, tratamientos y solicitudes de exámenes complementarios, además de enviar órdenes médicas lo que facilitan el seguimiento durante su estancia en el área. Los diagnósticos emitidos se codifican teniendo en cuenta el codificador internacional de enfermedades (CIE-10). También posibilita la obtención de reportes estadísticos relacionados con la atención de los pacientes en el área.

Con MediSys®EM tanto los médicos, como el personal de enfermería y el administrativo pueden acceder y actualizar la información del paciente. Es una solución web que utiliza como gestor de base de datos SQL Server 6.5. [5]

x-HIS

Es una herramienta clínica adaptada las necesidades de profesionales asistenciales y tiene como centro al paciente. Implementa funcionalidades que responden a las gestiones clínicas y administrativas que se realizan en un centro hospitalario pacientes. Cuenta con diversos módulos, uno de ellos es el de Urgencias que brinda la posibilidad de llevar el registro y control de tratamientos durante la estancia del paciente en el área y generar partes clínicos y judiciales.

x-HIS se integra con el módulo Gestión de prescripciones y resultados clínicos el cual permite realizar peticiones de exámenes complementarios a los diferentes servicios de apoyo al diagnóstico y realizar un seguimiento completo de las peticiones y sus resultados.

Es multiplataforma (Unix, Windows, Linux), multilenguaje, tiene soporte para una arquitectura cliente/servidor y presenta interdependencia de bases de datos: SQL Server, Oracle, Sybase. [6]

Sistema Informático para el Control de Urgencia Médica (SICUM)

SICUM es un sistema cubano que tiene como principales objetivos obtener información sobre el paciente en cada momento, aumentar la calidad en los servicios prestados, elevar el desarrollo informático en el sector de la salud y crear las bases para futuros sistemas inteligentes, que apoyen en la toma de decisiones.

Permite emitir el diagnóstico e indicar una conducta médica en correspondencia con este. Durante la estancia del paciente en el área de observación el personal médico brinda la posibilidad de registrar información relacionada con medicamentos suministrados y solicitudes de análisis de laboratorio.

Para la elaboración de este producto informático se utilizó como lenguaje visual Borland Delphi y como gestor de base de datos SQL Server. [7]

1.3 Tendencias y tecnologías actuales a considerar

La definición de tecnologías y herramientas a utilizar en el desarrollo de aplicaciones informáticas es una de las principales acciones a acometer por los productores de software. Como parte de la presente investigación son analizadas las características, ventajas y desventajas que poseen cada una de las tecnologías a utilizar durante el desarrollo del sistema, las cuales se mencionan a continuación.

Sistemas distribuidos. Modelo Cliente – Servidor

Un sistema distribuido es una colección de computadores autónomos conectados por una red, y con el software distribuido adecuadamente para que el sistema sea visto por los usuarios como una única entidad capaz de proporcionar todas sus funcionalidades. [8]

Dentro de las principales características de estos sistemas se destacan el alto nivel de fiabilidad, la seguridad contra interferencias externas y la privacidad de la información que gestionan. Además, posibilitan accesos concurrentes a bases de datos por parte de muchos usuarios, garantizan rapidez en el intercambio entre los sistemas y los usuarios y proveen puntos desde los cuales se tiene acceso a los servicios que se encuentran distribuidos en varias ubicaciones.

Modelo Cliente - Servidor

Este modelo o como también suele llamarse tecnología cliente-servidor, ha cobrado fuerza en la actualidad como herramienta poderosa para la manipulación de la información estadística, contable, funcional, entre otras, por las potencialidades que brinda, como pueden ser confiabilidad, rápido acceso, integridad en los datos, sistemas multiusuario, sistema de protección, etc.

Desde el punto de vista funcional, se puede definir el Modelo Cliente - Servidor como una arquitectura que tiene como función básica la realización de peticiones por parte de un programa (cliente), a otro programa (servidor), el cual le da respuesta. En un sistema distribuido cada máquina puede cumplir el rol de servidor para algunas tareas y el rol de cliente para otras.

Algunas de las características más notables de la arquitectura Cliente-Servidor son:

- Combinación de un cliente que interactúa con el usuario, y un servidor que interactúa con los recursos compartidos. El cliente proporciona la interfaz entre el usuario y el resto del sistema. El servidor actúa como un motor de software que maneja recursos compartidos tales como bases de datos, impresoras, módems, etc.

- Las tareas del cliente y del servidor tienen diferentes requerimientos en cuanto a recursos de cómputo como velocidad del procesador, memoria, velocidad y capacidades del disco y dispositivos de entrada o salida.
- Se establece una relación entre procesos distintos, los cuales pueden ser ejecutados en la misma máquina o en máquinas diferentes distribuidas a lo largo de la red.
- La relación establecida puede ser de muchos a uno, en la que un servidor puede dar servicio a muchos clientes, regulando su acceso a recursos compartidos.
- El ambiente es heterogéneo, puede diferir la plataforma de hardware y el sistema operativo del cliente y del servidor.
- El concepto de escalabilidad tanto horizontal como vertical es aplicable a cualquier sistema Cliente-Servidor. La escalabilidad horizontal permite agregar más estaciones de trabajo activas sin afectar significativamente el rendimiento. La escalabilidad vertical permite mejorar las características del servidor o agregar múltiples servidores. [9]

Entre las principales ventajas del esquema Cliente-Servidor están:

- La posibilidad de utilizar máquinas considerablemente más baratas que las requeridas por una solución centralizada, basada en sistemas grandes. Además, se pueden utilizar componentes, tanto de hardware como de software, de varios fabricantes, lo cual contribuye considerablemente a la reducción de costos.
- Al favorecer el uso de interfaces gráficas interactivas, los sistemas construidos bajo este esquema tienen mayor interacción con el usuario. No es siempre necesario transmitir información gráfica por la red pues esta puede residir en el cliente, lo cual permite aprovechar mejor el ancho de banda de la red.
- Las reglas de la organización y las reglas de seguridad se pueden definir una sola vez en el servidor para todos los usuarios.
- La estructura inherentemente modular facilita además la integración de nuevas tecnologías y el crecimiento de la infraestructura computacional, favoreciendo así la escalabilidad de las soluciones.

Patrones de arquitectura y diseño

Los patrones pretenden ser la solución al problema de la comunicación de experiencias que se plantean en la vida práctica. Cada patrón describe un problema concurrente en el entorno, para definir

después el camino a la solución a ese problema, de tal forma que pueda ser reutilizado en proyectos distintos.

Un patrón de diseño, es un conjunto de información que describe una solución recurrente a un problema de diseño no trivial. La efectividad de esta solución debe haber sido probada muchas veces por profesionales expertos dentro de un contexto determinado en situaciones anteriores y dicha solución debe ser reutilizable, para problemas de diseño similares en distintas circunstancias.

Los patrones arquitectónicos se establecen sobre aspectos fundamentales de la estructura de un software. Estos especifican un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes.

Arquitectura en capas

Este estilo arquitectónico se aplica con el objetivo de dividir fácilmente las responsabilidades con respecto al sistema, separando la lógica de negocios de la lógica de diseño. Una forma muy común de aplicarlo es definiendo tres capas: la de presentación, la de negocio y la de datos.

- Capa de presentación: Reúne todos los aspectos del software que tienen que ver con las interfaces y la interacción con los diferentes tipos de usuarios; típicamente incluyen el manejo y la apariencia de las ventanas, el formato de los reportes, menús, gráficos y elementos multimedia en general.
- Capa de negocio: Posee todos los aspectos del software que automatizan o apoyan los procesos de negocio que llevan a cabo los usuarios, incluyendo las tareas que forman parte de los procesos, las reglas y restricciones que aplican.
- Capa de datos: Incluye todos los aspectos del software que tienen que ver con el manejo de los datos persistentes.

Modelo Vista Controlador (MVC)

Este patrón posibilita la separación del modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres entidades diferentes, lo cual proporciona múltiples vistas sobre un mismo modelo de datos. Las tres entidades que define este patrón son los siguientes:

- Modelo: Administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).
- Vista: Maneja la visualización de la información.

- Controlador. Interpreta las acciones del usuario sobre el sistema, informando al modelo y/o a la vista para que cambien según resulte apropiado.

Entre las ventajas del MVC se pueden destacar las siguientes:

- Soporte de vistas múltiples. Dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. Por ejemplo, múltiples páginas de una aplicación de Web pueden utilizar el mismo modelo de objetos, mostrado de maneras diferentes.
- Adaptación al cambio. Los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones de representación, o requerir soporte para nuevos dispositivos como teléfonos celulares o PDAs. Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo. Este patrón sentó las bases para especializaciones ulteriores, tales como Page Controller y Front Controller. [10]

El Modelo Vista Controlador puede relacionarse con la arquitectura en tres capas, logrando un engranaje lógico, asociando cada uno de sus elementos, donde la capa de presentación podría corresponderse con la Vista, la capa de negocio con el Controlador y la capa de datos con el Modelo.

1.4 Tecnologías utilizadas en el proceso de desarrollo

Java Server Faces (JSF)

JavaServer Faces (JSF) es un framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE. JSF usa Facelets como la tecnología que permite hacer el despliegue de las páginas, pero también se puede acomodar a otras tecnologías como XUL. [11]

El uso de JSF permite el manejo de estados y eventos, la asociación entre los datos de la interfaz y los datos de la aplicación web, la especificación de la navegación del usuario, entre otras funcionalidades. Además, posibilita la creación de nuevos componentes por el desarrollador, lo que le imprime un alto grado de flexibilidad.

RichFaces

RichFaces es una librería de componentes web enriquecidos, de código abierto y basada en el estándar JSF. Con RichFaces se puede integrar fácilmente el código JavaScript asíncrono y XML

(AJAX), utilizando para ello el framework Ajax4jsf. Provee facilidades de validación y conversión de los datos proporcionados por el usuario, administración avanzada de recursos como imágenes, código JavaScript y Hojas de Estilo en Cascada (CSS por sus siglas en inglés). Además, permite crear interfaces de usuario modernas de manera eficiente y rápida, basadas en componentes altamente configurables en cuanto a temas y esquemas de colores predefinidos por el propio framework o desarrollados a conveniencia. [12]

Ajax4jsf

Ajax4jsf es una librería de código abierto que se integra totalmente en la implementación de JSF, y extiende la funcionalidad de sus etiquetas dotándolas con tecnología Ajax sin código Javascript. Mediante este framework se puede variar el ciclo de vida de una petición JSF, recargar determinados componentes de la página sin necesidad de refrescarla en su totalidad, realizar peticiones al servidor de forma automática, controlar cualquier evento de usuario, etc. [13]

Facelets

Facelets es un framework para plantillas centrado en la tecnología JavaServer Faces (JSF), permitiendo que JavaServer Pages (JSP) y JSF puedan funcionar conjuntamente en una misma aplicación web, los cuales no se complementan naturalmente. [14]

Con este framework es posible diseñar de forma libre una página web y luego asociarle los componentes JSF específicos; esto aporta mayor libertad al diseñador y mejora los informes de errores que tiene JSF entre otras cosas.

Lenguaje de Marcado de Hipertexto Extensible (XHTML)

XHTML es una versión más estricta y clara de HTML, que elimina sus limitaciones de uso con herramientas basadas en XML. XHTML combina la sintaxis de HTML 4.0, diseñado para mostrar datos, con la de XML, diseñado para describirlos. Además, permite una correcta interpretación de la información independientemente del dispositivo desde el que se accede a ella y puede incluir otros lenguajes como Mathematical Markup Language (MathML), Synchronized Multimedia Integration Language (SMIL) o Scalable Vector Graphics (SVG), al contrario que HTML.

Extended Expression Language

Extended EL es una extensión al estándar Lenguaje de Expresión Unificado (EL) que aumenta su expresividad y sus potencialidades. Este estándar adopta las características ofrecidas por el lenguaje de expresión de JSF. Mediante su utilización se puede reducir considerablemente la cantidad de

código en las páginas creadas, aumentando la productividad, haciendo más fácil el mantenimiento y el entendimiento de las mismas.

JBoss Seam

JBoss Seam 2.0 (Seam) es un framework para el desarrollo de aplicaciones web en Java, que define un modelo de componentes uniforme para toda la lógica de negocio de las aplicaciones que sean desarrolladas mediante su utilización. Integra fácilmente tecnologías estándares como Java Server Faces (JSF), modelo de componentes para la capa de presentación; Enterprise JavaBeans (EJB3), modelo de componentes para la lógica de negocio y persistencia del lado del servidor; Java Persistence API (JPA), y de Business Process Management (BPM).

Integra además librerías de controles de código abierto basadas en JSF como RichFaces y permite a través de ficheros de reglas (Drools) definir las posibles bifurcaciones del negocio permitiendo el fácil manejo de las condiciones sin tener que modificar el código fuente. [15]

Drools

Drools es una implementación del JSR 94 (Java Rule Engine API), especificación que define una interfaz común para un motor de reglas estándar dentro de la plataforma Java. Para definir las reglas emplea XML y permite adaptarse a la semántica de un determinado dominio definiendo un esquema que la represente. [16]

Uno de sus principales beneficios es que permite separar la lógica de negocio del código de la aplicación, de esta manera, un cambio en el flujo del negocio no implica la realización de modificaciones en el código de la aplicación.

Hibernate

Es una herramienta de código abierto dedicada al mapeo objeto-relacional para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones.

Cuenta con el Hibernate Query Language (HQL), que es un lenguaje de consulta similar al SQL. Además brinda, de una manera muy rápida y mejorada, la posibilidad de generar bases de datos en cualquiera de los entornos soportados: PosgtreSQL, Oracle, DB2, MySql, entre otros.

Capítulo I. Fundamentación Teórica

Java Persistence API (JPA)

Es la interfaz de programación de aplicaciones (API por sus siglas en inglés) de persistencia desarrollada para la plataforma Java EE y está incluida en el estándar EJB3. Esta API busca unificar la manera en que funcionan las utilidades que provee un mapeo objeto-relacional. El objetivo que persigue su diseño es no perder las ventajas de la orientación a objetos al interactuar con una base de datos, como sucedía con EJB2, y permitir usar objetos regulares conocidos como POJOs (Plain Old Java Object). [17]

Enterprise Java Bean 3 (EJB 3)

EJB 3 proporciona un modelo distribuido y estándar de componentes que se ejecutan en el servidor. Su objetivo es dotar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación empresarial (concurrencia, transacciones, persistencia, seguridad, etc.), para centrarse en el desarrollo de la lógica de negocio en sí. El hecho de estar basado en componentes permite que éstos sean flexibles y sobre todo reutilizables. Incorpora el estándar JPA como el principal API de persistencia para aplicaciones EJB3.

Java Platform Enterprise Edition 5 (JavaEE 5)

Java Platform Enterprise Edition o Java versión 5 es una plataforma de programación que parte de la Plataforma Java; para desarrollar y ejecutar software de aplicaciones en lenguaje de programación Java con arquitectura distribuida de n niveles. Se basa ampliamente en componentes de software modulares y se ejecuta sobre un servidor de aplicaciones. [18]

PosgtreSQL 8.3

PostgreSQL es un sistema gestor de base de datos (SGBD) que se destaca por su robustez, escalabilidad y cumplimiento de los estándares SQL. Es un software de código abierto que está bajo la licencia BSD (Berkeley Software Distribution). Presenta alta concurrencia, para esto utiliza la tecnología de Control de Concurrencia Multi-Versión (Multiversion concurrency control (MVCC)), con lo que se logra que ningún lector sea bloqueado por un escritor.

Soporta ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad), o lo que es lo mismo, la realización de transacciones seguras; también cuenta con vistas, uniones y claves extranjeras. Además, implementa internamente lenguajes de consulta de muy alto nivel como son el plpgsql (muy similar al plsql de Oracle), el plperl, el plpiton y el c.

El tamaño máximo de la base de datos es ilimitado; el de una tabla asciende a 32 TB, el de una fila a 1.6 TB y el de un campo de datos a 1 GB; el número de filas en una tabla es ilimitado, pero no el de columnas, que oscila entre 250 y 1600 columnas por tabla. El número de índices por tabla es también ilimitado.

JBoss Application Server (JBoss AS)

JBoss AS es el servidor de aplicaciones de código abierto más ampliamente desarrollado del mercado. Se encuentra bajo la licencia GNU LGPL, por lo que puede libremente usarse sin costo alguno en cualquier aplicación comercial o ser redistribuido. Por ser una plataforma certificada JEE 5, soporta todas las especificaciones correspondientes, incluyendo servicios adicionales como clustering, caching y persistencia.

JBoss es ideal para aplicaciones Java, pues está basado en este lenguaje de programación, y también es propicio para aplicaciones basadas en la web. También soporta Enterprise Java Beans (EJB) 3.0. Los componentes claves son: JBoss AS 4.2, Hibernate 3.2.4, Seam 2.0.

JBoss Tools

Es un conjunto de plug-in para el Eclipse que permite el manejo de diferentes frameworks que facilitan el desarrollo de aplicaciones. Está constituido por varios módulos: RichFaces VE, Seam Tools, Hibernate Tools y JBoss AS Tools. [19]

1.5 Lenguajes de programación

Java

Lenguaje de programación desarrollado por la compañía Sun Microsystems, que utiliza el paradigma de la Programación Orientada a Objetos (POO). Es un lenguaje robusto pues no permite el manejo directo del hardware ni de la memoria. Dentro de sus principales ventajas se encuentra la de ser multiplataforma. Con Java se pueden programar aplicaciones webs dinámicas, con acceso a bases de datos, utilizando XML, con cualquier tipo de conexión de red entre cualquier sistema. Este lenguaje es utilizado de manera horizontal en el desarrollo del sistema, pues puede estar presente en las diferentes capas de la aplicación. [20]

El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple, que elimina herramientas de bajo nivel, las cuales suelen provocar muchos errores, como la manipulación directa de punteros o memoria.

La programación en Java, permite el desarrollo de aplicaciones bajo el esquema de Cliente Servidor y de aplicaciones distribuidas, lo que lo hace capaz de conectar dos o más computadoras, ejecutando tareas simultáneamente.

1.6 Metodologías de desarrollo

Proceso Unificado de Desarrollo (RUP)

RUP es el resultado de varios años de trabajo y uso práctico en el que se han unificado técnicas de desarrollo y trabajo, a través del UML. En RUP se han dividido las actividades en grupos lógicos en los que se definen nueve flujos de trabajo principales. Los seis primeros son conocidos como flujos de ingeniería y los tres últimos como flujos de apoyo. El ciclo de vida de RUP se caracteriza por ser dirigido por caso de uso, centrado en la arquitectura, iterativo e incremental.

Cada fase cambia el foco del equipo de trabajo para alcanzar cada uno de los hitos y es llevada a cabo en forma iterativa. Esto quiere decir que la fase se fragmenta en pequeños proyectos que recorren todas las disciplinas y producen un ejecutable. Esta es la forma más efectiva de verificar el progreso del proyecto y de reducir los riesgos inherentes. [21]

Lenguaje Unificado de Modelado (UML)

UML es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software. Permite la modelación de sistemas con tecnología orientada a objetos. Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software (como RUP), pero no especifica en sí mismo qué metodología o proceso utilizar.

1.7 Herramientas de desarrollo

Eclipse

Esta herramienta es un Entorno Integrado de Desarrollo (IDE, por sus siglas en inglés) multiplataforma, de código abierto y extensible. Ofrece la posibilidad de añadirle módulos o pluggins, que facilita el uso de JBoss Tools, para el desarrollo de aplicaciones con RichFaces, Seam, Hibernate y el JBoss AS. Además, brinda cobertura a otros lenguajes de programación aparte de Java, como pueden ser C/C++, Python, Cobol, Fortran, PHP, entre otros.

Capítulo I. Fundamentación Teórica

Visual Paradigm

Visual Paradigm es una eficaz herramienta para visualizar y diseñar elementos de software, para ello utiliza UML y ofrece una gama de facilidades para el modelado de aplicaciones. Está orientada a la creación de diseños usando el paradigma de programación orientada a objetos.

Provee soporte para la generación de código, tiene integración con diversos IDE's como NetBeans (de Sun Microsystems), JDeveloper (de Oracle), Eclipse (de IBM), JBuilder (de Borland), así como la posibilidad de realizarse la ingeniería inversa para aplicaciones realizadas en JAVA, .NET, XML e Hibernate.

También tiene disponibilidad en múltiples plataformas y soporta BPMN (Business Process Modeling Notation), modelado colaborativo con CVS y Subversion. Además, su diseño se centra en casos de uso.

Java Runtime Environment 6 (JRE 6)

JRE es el entorno en tiempo de ejecución Java y se corresponde con un conjunto de utilidades que permite la ejecución de programas java sobre todas las plataformas soportadas. JVM (máquina virtual Java) es una instancia de JRE en tiempo de ejecución. Este interpreta el código Java y está compuesto además por las librerías de clases que implementan el API de Java. Ambas, JVM y API, deben ser consistentes entre sí, de ahí que sean distribuidas de modo conjunto.

iReport

La herramienta iReport es un constructor / diseñador de informes visual, poderoso, intuitivo y fácil de usar para JasperReports escrito en Java. Este instrumento permite que los usuarios corrijan visualmente informes complejos con cartas, imágenes, subinformes, etc. iReport está además integrado con JFreeChart, una de la biblioteca gráficas OpenSource más difundida para Java. Los datos para imprimir pueden ser recuperados por varios caminos incluso múltiples uniones JDBC, TableModels, JavaBeans, XML, etc.

La lista siguiente describe algunas de las características importantes de iReport:

- 100% escrito en JAVA y además OPENSOURCE y gratuito.
- Maneja el 98% de las etiquetas de JasperReports
- Permite diseñar con sus propias herramientas: rectángulos, líneas, elipses, campos de los textfields, cartas, subreports (subreportes).

- Soporta internacionalización nativamente.
- Browser de la estructura del documento.
- Recopilador y exportador integrados.
- Soporta JDBC.
- Soporta JavaBeans como orígenes de datos (éstos deben implementar la interface JRDataSource).
- Incluye Wizard's (asistentes) para crear automáticamente informes.
- Tiene asistentes para generar los subreportes
- Tiene asistentes para las plantillas.
- Facilidad de instalación.[22]

Conclusiones

En este capítulo se realizó un estudio de los sistemas de información hospitalaria existentes tanto a nivel nacional como internacional, lo que permite concluir que la mayoría están desarrollados sobre software propietario con altos costos de licencia, no son multiplataforma y algunos son aplicaciones desktop, aspecto que dificulta el despliegue a gran escala.

Además, los sistemas que cuentan con el módulo de Emergencias o Urgencias no implementan las funcionalidades correspondientes a la administración de recursos. Algunos de estos sistemas se auxilian de otros módulos para complementar acciones que se realizan durante la estancia al paciente en el área de Observación de Emergencias, pero estas no son suficientes, teniendo en cuenta el amplio conjunto de actividades que se llevan a cabo en esta área.

De igual forma fueron analizadas un conjunto de herramientas, metodologías y tecnologías que constituyen la propuesta tecnológica, para el desarrollo de los procesos Observación y Administración de recursos del Módulo Emergencia para el Sistema de Información Hospitalaria alas HIS.

Capítulo II. DESCRIPCIÓN DE LA ARQUITECTURA

En el presente capítulo se especifican los requerimientos no funcionales y se describe la concepción arquitectónica definida por el departamento Sistemas de Gestión Hospitalaria. Además, se especifica la estrategia de codificación a seguir y se identifican los posibles componentes a reutilizar.

2.1 Requerimientos no funcionales

Los requerimientos no funcionales son aquellos que no se refieren directamente a las funciones específicas del sistema, sino a las propiedades emergentes de éste como la fiabilidad, la respuesta en el tiempo y la capacidad de almacenamiento. De forma alternativa, definen las restricciones del sistema como la capacidad de los dispositivos de entrada/salida y la representación de datos que se utiliza en la interfaz del sistema. [23]

Usabilidad

El sistema estará diseñado de manera que los usuarios adquieran las habilidades necesarias para explotarlo en un tiempo reducido:

Usuarios normales: 20 días

Usuarios avanzados: 30 días

Fiabilidad

En los servidores de los hospitales y en el Centro de Datos Nacional del MPPS se garantizará una arquitectura de máxima disponibilidad, tanto de servidores de aplicación como de base de datos. Se garantizarán además, políticas de respaldo a toda la información, evitando pérdidas en caso de desastres ajenos al sistema.

Los estudios imagenológicos y otros datos que por su tamaño no se puedan replicar hacia el Centro de Datos, se almacenarán localmente en los hospitales; quedando la referencia a dicho estudio en el Centro de Datos, de tal forma que se pueda mediante una transmisión directa entre los hospitales, sin que medie para esto el Centro de Datos Nacional.

Las informaciones médicas relacionadas con los pacientes y que vayan a ser intercambiadas con otros hospitales por la red pública, viajarán cifradas para evitar accesos o modificaciones no autorizadas.

Los mecanismos serán capaces de informar al personal autorizado sobre posibles irregularidades que den indicios sobre la introducción de información falseada.

El sistema implementará un mecanismo de auditoría para el registro de todos los accesos efectuados por los usuarios, proporcionando un registro de actividades (log) de cada usuario en el sistema.

El producto soportará el uso de firmas digitales para la transferencia de información cuya certificación sea imprescindible para validar el uso de la misma.

Se 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 que se le han realizado.

Ninguna información que se haya ingresado en el sistema será eliminada físicamente de la BD, independientemente de que para el sistema, este elemento ya no exista.

La recuperación de la información de la base de datos a partir de los respaldos o salvas realizadas.

Eficiencia

El Centro de Datos permitirá agregar recursos para aumentar el poder de procesamiento y almacenamiento sin afectar los sistemas, garantizando expansiones motivadas por futuros requerimientos. El sistema minimizará el volumen de datos en las peticiones y además optimizará el uso de recursos críticos como la memoria. Para ello se potenciará como regla guardar en la memoria caché datos y recursos de alta demanda.

Soporte

Se permitirá la creación de usuarios, otorgamiento de privilegios y roles, asignación de perfiles y activación de permisos por direcciones IP, la administración remota, monitoreo del funcionamiento del sistema en los centros hospitalarios y detección de fallas de comunicación.

También se podrá realizar copias de seguridad de la base de datos hacia otro dispositivo de almacenamiento externo, recuperar la base de datos a partir de los respaldos realizados, chequear las operaciones y acceso de los usuarios al sistema, establecer parámetros de configuración del sistema y la actualización de nomencladores.

Seguridad de acceso y administración de usuarios

Se mantendrá seguridad y control a nivel de usuario, garantizando su acceso sólo a los niveles establecidos de acuerdo con la función que realizan. Las contraseñas podrán cambiarse solo por el propio usuario o por el administrador del sistema.

Se mantendrá un segundo nivel de seguridad a nivel de estaciones de trabajo, garantizando únicamente la ejecución de las aplicaciones que hayan sido definidas para la estación en cuestión. 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 proporcionará un registro de actividades (log) de cada usuario. Ninguna información que se haya ingresado en el sistema será eliminada físicamente de la base de datos.

El sistema permitirá la recuperación de la información de la base de datos a partir de los respaldos o salvas realizadas.

Monitoreo de funcionamiento

Se permitirá administración remota, monitoreo del funcionamiento del sistema en los centros hospitalarios y detección de fallas de comunicación.

Respaldo y recuperación de base de datos

Se permitirá realizar copias de seguridad de la base de datos hacia otro dispositivo de almacenamiento externo, además de recuperar la base de datos a partir de los respaldos realizados.

Auditoría

Se permitirá el chequeo de las operaciones y acceso de los usuarios al sistema, para esto debe existir un registro de trazas que almacene todas las transacciones realizadas en el sistema, indicando para cada caso como mínimo: usuario que realizó la transacción, tipo de operación que se realizó, fecha y hora en que se realizó la operación e información contenida en el registro modificado.

Configuración de parámetros

Se permitirá establecer parámetros de configuración del sistema y actualización de nomencladores.

Réplica

Se permitirá realizar réplica de la base de datos de los hospitales con el Centro de Datos. Esta réplica se podrá hacer de forma manual y automatizada a través de la red.

Restricciones de diseño

La capa de presentación contendrá todas las vistas y la lógica de la presentación. El flujo web se manejará de forma declarativa y basándose en definiciones de procesos del negocio. La capa del negocio mantendrá el estado de las conversaciones y procesos del negocio que concurrentemente pueden estar siendo ejecutados por cada usuario.

La capa de acceso a datos contendrá 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.

Requisitos para la documentación de usuarios en línea y ayuda del sistema

Se posibilitará el uso de ayudas dinámicas y tutoriales en línea sobre el funcionamiento del sistema.

Capítulo II. Descripción de la arquitectura

Interfaz

Interfaces de usuario

Las ventanas del sistema contendrán los datos claros y bien estructurados, además de permitir la interpretación correcta de la información. La interfaz contará con teclas de función y menús desplegables que faciliten y aceleren su utilización. La entrada de datos incorrecta será detectada claramente e informada al usuario. Todos los textos y mensajes en pantalla aparecerán en idioma español.

Interfaces de comunicación

Para el intercambio electrónico de datos entre aplicaciones se usará el protocolo HL7 (Health Level Seven). El sistema usará el formato estándar WSDL (Web Services Desciption Language) para la descripción de los servicios web. El sistema implementará mecanismos de encriptación de datos para el intercambio de información con sistemas externos. El sistema utilizará mecanismos de compactación de los datos que se intercambiarán con sistemas externos con el objetivo de minimizar el tráfico en la red y economizar el ancho de banda.

Rendimiento

El sistema minimizará el volumen de datos en las peticiones y además optimizará el uso de recursos críticos como la memoria.

El sistema respetará buenas prácticas de programación para incrementar el rendimiento en operaciones costosas para la máquina virtual como la creación de objetos.

Hardware

- Estaciones de trabajo**.**

En la solución se incluyen estaciones de trabajo para las consultas del Sistema de Información Hospitalaria alas HIS, las que necesitan capacidad de hardware que soporte un sistema operativo que cuente con un navegador actualizado y que siga los estándares web, se recomienda Internet Explorer 7, Firefox 2 o versiones superiores. Por lo que se escogieron estaciones de trabajo de 256 Mb de memoria RAM y un microprocesador de 2.0 Hz con sistema operativo Linux.

- Servidores.

La solución estará conformada, fundamentalmente, por servidores de alta capacidad de procesamiento y redundancia, que permitan garantizar movilidad y residencia de la información y las aplicaciones bajo esquemas seguros y confiables. Servidores de Base de datos: 1 DL380 G5, Procesador Intel® Xeon® 5140 Dual - Core 4GB de memoria y 2x72GB de disco y sistema operativo Linux. Servidores de

Aplicaciones: 2 DL380 G5, Procesador Intel® Xeon® 5140 Dual - Core 4GB de memoria y 2x72GB de disco y sistema operativo Linux. Servidores de Intercambio: 1 DL380 G5, Procesador Intel® Xeon® 5140 Dual - Core 2 GB de memoria y 2x72GB de disco y sistema operativo Linux.

Software

El sistema debe correr en sistemas operativos Windows, Unix y Linux, utilizando la plataforma JAVA (Java Virtual Machine, JBoss AS y Postgre SQL). El sistema deberá disponer de un navegador web, estos pueden ser IE 7, Opera 9, Google chrome 1 y Firefox 2 o versiones superiores de estos.

2.2 Descripción de la arquitectura

La arquitectura de software es un conjunto de patrones que sirven de guía para la elaboración de un sistema, ofrece además la descripción del software en subsistemas y componentes, dejando bien claro la forma en que estos interactúan. Su objetivo principal es aportar elementos que ayuden a la toma de decisiones y al mismo tiempo, proporcionar conceptos y un lenguaje común que permitan la comunicación entre los equipos que participen en un proyecto.

Para el desarrollo del software se define como parte de la línea base de la arquitectura el patrón de diseño Modelo Vista Controlador (MVC). El mismo separa la interfaz de usuario, la lógica de control y los datos de una aplicación, en tres componentes distintos: el modelo, que abarca los datos y las reglas del negocio; la vista, que muestra la información del modelo al usuario; y el controlador, que gestiona las entradas del mismo. Esto ofrece varias ventajas, como la reusabilidad de componentes, además permite que modificaciones en las vistas afecten lo menos posible en la lógica de negocio o de datos.

Con el objetivo de lograr una mejor organización de los componentes que conforman los procesos *dejar en observación* y *administración de recursos*, y siguiendo las características del patrón MVC, se agruparon dichos componentes en tres capas fundamentales: presentación, negocio y acceso a datos. Esta estructura garantiza que el cambio o sustitución de un elemento correspondiente a una de las capas, no altere el funcionamiento de otro que lo utilice y sea parte de alguna de las capas restantes.

La capa de presentación está formada por páginas XHTML, las cuales están compuestas por formularios y controles JSF y RichFaces, estos se encargan de validar y mostrar los datos haciendo uso del componente Ajax4jsf.

La capa de negocio está compuesta por clases controladoras, que definen la lógica del negocio conjuntamente con los datos que se validan en la capa de presentación. Todo esto se desarrolla mediante el framework Seam y permite a través de las reglas (Drools), facilitar las funcionalidades del sistema.

La capa de acceso a datos realiza el manejo de los datos mediante Hibernate, lo cual permite seleccionar, modificar, eliminar y persistir la información en la base de datos. Esto permite llevar las consultas a un lenguaje orientado a objeto.

2.3 Análisis de posibles implementaciones, componentes o módulos ya existentes y que puedan ser rehusados

La reutilización se apoya en el uso de componentes de software, vistos como colecciones de código reutilizables que facilitan el desarrollo de las aplicaciones, los cuales se pueden encontrar en el mismo sistema que se está desarrollando o en sistemas externos que incluyan la solución que se desea desarrollar. Esta reduce los costes de diseño, codificación y comprobación.

El Sistema de Información Hospitalaria alas HIS, el cual se encuentra en desarrollo, está compuesto por diferentes módulos que comparten componentes entre sí. Para la implementación de los procesos enmarcados en el campo de acción de la presente investigación, se identificaron los componentes a reutilizar a partir de un estudio realizado a las funcionalidades de los módulos relacionados con la atención al paciente y los servicios de apoyo al diagnóstico.

Para conformar el diagnóstico en el proceso de observación se reutilizó del módulo de Emergencias la funcionalidad seleccionar enfermedad y para visualizar con claridad la lesión que presenta el paciente y qué la ocasionó, las funcionalidades dibujar lesiones externas e internas.

Del módulo Bloque Quirúrgico se reutilizó la funcionalidad notificar solicitud de intervención, para los pacientes que requieren ser intervenidos quirúrgicamente de forma inmediata. A los pacientes que son atendidos en el área de observación se les puede indicar exámenes complementarios con el objetivo de esclarecer su enfermedad y llegar un diagnóstico acertado, de ahí que del módulo Laboratorio se utilice el crear solicitud de análisis de laboratorio. Por otra parte del módulo Banco de Sangre se pueden reutilizar las funcionalidades relacionadas con la creación de solicitudes y registros de transfusión.

Crear solicitud de interconsulta y crear orden médica son otras de las funcionalidades que se pueden realizar durante la atención al paciente en el área de observación por lo que estas se pueden reutilizar de módulo de Hospitalización. Asimismo, del módulo de Enfermería se utilizan los componentes relacionados con las funcionalidades buscar control hidromineral, buscar control de tratamientos y buscar evoluciones de enfermería con el objetivo de que el médico tenga conocimiento de la evolución del paciente ante los medicamentos y tratamientos indicados durante la estancia en el área de observación.

Por otra parte, existen algunos componentes que se definen de manera general para ser utilizados por todos los módulos del sistema alas HIS, los mismos brindan un conjunto de facilidades y simplifican el trabajo de los desarrolladores. Dentro estos están: la clase Bitácora para el control de las trazas de todas las acciones que se realizan con la aplicación; la clase User para saber qué usuario está trabajando con la aplicación en tiempo real; y la clase ActiveModule para conocer qué modulo está activo y en qué entidad.

2.4 Seguridad

Para obtener un sistema seguro se debe garantizar la confidencialidad, integridad y disponibilidad de su información. Por lo que se debe tener un gran control de la misma para garantizar que no sea accedida ni modificada por personal no autorizado. A continuación se mencionan algunas características que deben cumplir el software para que sea seguro:

El sistema debe contar con secciones las cuales les permiten a los usuarios acceder a sus permisos establecidos, además de registrar trazas para ser verificadas por el administrador en caso de ser necesario. Las contraseñas solo podrán ser cambiadas por el administrador del sistema además de los permisos para acceder a los datos.

La información que se intercambia entre el sistema y otros sistemas debe ser cifrada, pues de esta forma se asegura que no se lean o modifiquen los datos confidenciales que se manejan. Además, todo lo que se haya ingresado en el sistema no será eliminado físicamente de la base de datos.

2.5 Vista de despliegue

En la construcción de aplicaciones usando UML existen diferentes vistas para visualizar, especificar, construir y documentar la arquitectura del software. Este lenguaje permite representar cada vista mediante un conjunto de diagramas, un ejemplo de ello es la vista de despliegue o implantación que cuenta con el diagrama de despliegue, el cual se usa para ilustrar la vista estática de implantación de un sistema a través de un conjunto de nodos unidos por conexiones de comunicación.

La arquitectura en tiempo de ejecución del Sistema de Información Hospitalaria alas HIS se modeló utilizando tres elementos de hardware: una computadora cliente y dos servidores (servidor de aplicación y servidor de base de datos), además se dispuso de la conexión con un dispositivo, en este caso la impresora.

La comunicación entre los nodos se realiza por los protocolos HTTP para asociar la computadora cliente y el servidor de aplicaciones y por TCP/IP para establecer la conexión entre los dos servidores. La computadora cliente se conecta a la impresora a través de los puertos USB o LPT.

Diagrama de despliegue

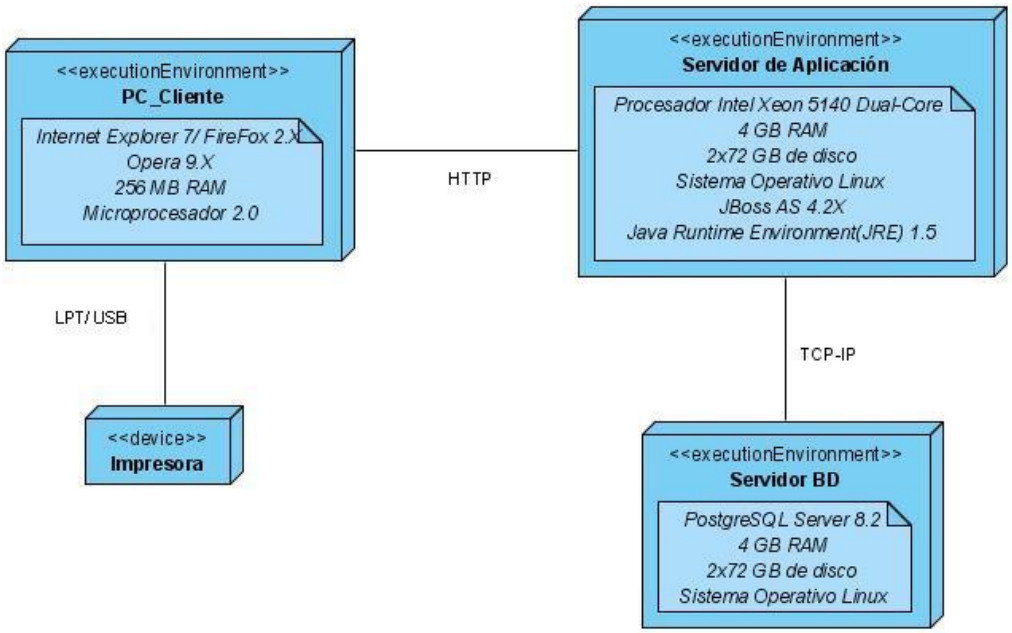


Figura 2.1 Diagrama de despliegue

2.6 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 prudente, práctico y entendible para todos los programadores. Usar técnicas de codificación sólidas y realizar buenas prácticas de programación es de gran importancia para la calidad del software y para obtención de un buen rendimiento.

Para el desarrollo del sistema presentado, se utiliza un estándar de codificación, garantizando la homogeneidad del estilo de programación durante la implementación del sistema. A continuación se describen algunos de los elementos que conformar el estándar de codificación:

En cuanto al *idioma* todas las palabras se deben usar en idioma español, pero sin acentuarse.

Lograr una estructura uniforme para los bloques de código así como para los diferentes niveles de anidamiento es el objetivo de la *identación*, para esta se definieron los siguientes aspectos:

- Inicio y fin de bloque: se deben dejar dos espacios en blanco desde la instrucción anterior para el inicio y fin de bloque {}, asimismo para el caso de las instrucciones if, else, for, while, do while, switch, foreach.

- Aspectos generales: evitar el uso del tabulador; pues que este puede variar según la PC o la configuración de dicha tecla. Los inicios ({) y cierre (}) de ámbito deber estar alineados debajo de la declaración a la que pertenecen y deben evitarse si hay sólo una instrucción. No se debe colocar ({) en la línea de un código cualquiera, pues esto requiere una línea propia.

Los comentarios, separadores, líneas, espacios en blanco y márgenes fueron otros de los elementos a considerar en el estándar, con estos se logra establecer un modo común para comentar el código de forma tal que se comprenda con sólo leerlo una vez.

- Ubicación de comentarios: estos se ubican al inicio de cada clase o función y al final de cada bloque de código, especificando el objetivo de la misma así como el tipo de dato y el objetivo de cada parámetro.
- Líneas en blanco: esta se emplean antes y después de métodos, clases y estructuras.
- Espacios en blanco: se recomienda entre operadores lógicos y aritméticos para lograr una mayor legibilidad en el código.
- Aspectos generales:
  - ✓ No comentar cada línea de código. Cuando el comentario se aplica a un grupo de instrucciones debe estar seguido de una línea en blanco. Cuando se comente una sola instrucción se suprime la línea en blanco o se escribe a continuación de la instrucción.
  - ✓ No se usa un espacio en blanco después del corchete abierto y antes del cerrado de un arreglo, después del paréntesis abierto y antes del cerrado y antes de un punto y coma.

En cuanto al uso de variables y contantes se tuvo en cuenta los siguientes elementos:

- Apariencia de variables: el nombre de las variables debe comenzar con la primera letra en minúscula, en caso de que sea un nombre compuesto se empleará la notación CamellCasing[1].
- Apariencia de constantes: se declaran con todas sus letras en mayúscula.
- Aspectos generales: los nombres de las variables y constantes deben tener nombres que con sólo leerlas se conozca el propósito.

Para el uso de clases y objetos se definieron un conjunto de elementos que garantizan nombrar las clases e instancias de la misma forma para toda la aplicación:

[1] **Notación CamellCasing:** Especifica que la palabra de inicio del identificador comienza con minúscula. Si el identificador está compuesto por más de una palabra entonces estás deben comenzar con mayúscula.

- Apariencia de clases y objetos: 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[2]. Para el caso de las instancias se comenzará con un prefijo que identificará el tipo de dato, este se escribirá en minúscula.
- Apariencia de atributos: en caso de que sea un nombre compuesto se empleará notación CamellCasing.
- Apariencia de las funciones: el nombre de las funciones con verbos que denoten la acción que realiza y se empleará la notación PascalCasing. Las funcionen que obtienen algún dato emplean el prefijo get y si fijan algún valor se emplea el prefijo set
- Declaración de parámetros en funciones: se declaran agrupados por tipos, defiendo primero los string y luego los numéricos, además se agrupan tenido en cuenta los valores por defecto.
- Aspectos generales: el nombre que se emplea para las clases, objetos, atributos y funciones debe permitir que con sólo leerlo se conozca el propósito de los mismos.

La base de datos, tablas, esquemas y campos fueron otros de los elementos que se tuvieron en cuenta en la definición del estándar.

- Apariencia de la base de datos: el nombre debe comenzar con el prefijo bd a continuación un underscoard[3] y luego el nombre completamente en minúscula. Los nombres serán cortos y descriptivos.
- Apariencia de las tablas: el nombre comienza con el prefijo tb seguido de underscore y luego se escriben todas las letras en minúscula, en caso de ser un nombre compuesto se utiliza underscore para separarlo.
- Tablas que representen relaciones: el nombre empleado para estas tablas comienza con el prefijo tr seguido de underscore y la concatenación del nombre de las dos tablas que la generaron separados por underscore, todo en minúscula.
- Tablas que representen nomencladores: el nombre a emplear para estas tablas de relación debe comenzar con el prefijo tn seguido de underscoard, debe ser corto, descriptivo y todo en minúscula.
- Apariencia de los campos: el nombre de los campos se escribe con todas las letras en minúscula para evitar problemas con el Case Sensitive del gestor

[2] **Notación PascallCasing:** Especifica que la palabra de inicio del identificador comienza con mayúscula. Si el identificador está compuesto por más de una palabra entonces estás deben comenzar con mayúscula.
[3] **Underscoard:** Carácter ASCII representado como "_".

- Nombre de los campos: todos los campos: los campos identificadores comienzan con el identificador id seguido de underscore y posteriormente el nombre del campo.
- Aspectos generales: el nombre empleado para las bases de datos, tablas y campos, con sólo leerlos se conoce el propósito de los mismos.

Además, se tuvo en cuenta aspectos relacionados con los controles, los cuales se muestran a continuación:

- Apariencia de los controles: el nombre que se le asigna a los controles comienza con las primeras letras en minúscula, las cuales identifican el tipo de datos al que se refiere. Para los nombres compuestos se emplea la notación CamellCasing.

Conclusiones

En este capítulo se describió la concepción arquitectónica y la seguridad del sistema. También se definió la estrategia de codificación enmarcada en el estándar a utilizar, se especificaron los requerimientos no funcionales y componentes a reutilizar y además se elaboró el diagrama de despliegue con el objetivo de visualizar la estructura física del sistema.

Capítulo III. DESCRIPCIÓN Y ANÁLISIS DE LA SOLUCIÓN PROPUESTA

En el presente capítulo se realiza el análisis y descripción de la solución propuesta, para lo que se definen claramente los diagramas de clases y de interacción, que son las entradas para el modelo de implementación; el modelo de datos y diagramas de componentes; estos últimos con el objetivo de mostrar las dependencias entre las partes del código del sistema propuesto. Además, se describen las clases y entidades asociadas a cada diagrama representado.

3.1 Valoración crítica del diseño propuesto por el analista

En el desarrollo de un software el diseño es el encargado de traducir los requisitos a una estructura que describe como implementar el sistema, siendo lo suficientemente específica para que no existan ambigüedades, y ofreciendo la posibilidad de descomponer los trabajos de implementación en componentes más manejables, visualizándolos mediante una notación común.

Generalmente, para elaborar el diseño se utiliza un grupo de patrones, o modelos para lograr los objetivos esperados. Estos se definen como soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos, basados en la experiencia y que se ha demostrado que funcionan.

Para definir el diseño de la solución propuesta, se tuvieron en cuenta varios patrones, dentro de los que se destacan los GRASP. Estos se usaron con el fin de asignar a cada clase o componente las responsabilidades que debe cumplir, teniendo en cuenta la información que posee o su comportamiento, además de crear las instancias de otras clases en correspondencia con la responsabilidad dada, poniéndose de manifiesto los patrones *Experto* y *Creador*.

Dentro de esta misma clasificación de patrones que se usaron, también se encuentran el de *Bajo acoplamiento* y *Alta cohesión* permitiendo la colaboración entre los elementos del diseño (clases), sin afectarse la reutilización de los mismos y el entendimiento de estos cuando se encuentran separados. La creación de clases controladoras facilitó realizar las operaciones del sistema, debido a que estas operaciones reflejan los procesos de las instituciones y no es factible manejarse en la capa de interfaz o presentación.

Otros de los patrones utilizados fueron los de diseño a objetos, tal es el caso del patrón de creación *Abstract Factory* el cual permite el acceso a la base de datos a través de EntityManager. Esta es la interfaz principal de JPA utilizada para la persistencia de las aplicaciones, donde cada instancia puede realizar operaciones como inserción, lectura, modificación y eliminación (CRUD por sus siglas en inglés) sobre un conjunto de objetos persistentes.

Hibernate implementa también el patrón *Active record* donde una fila en la tabla de la base de datos se envuelve en una clase, de manera que se asocian filas únicas de la base de datos con objetos del lenguaje de programación usado. También implementa los patrones de mapeo: *Identity field*, *Foreign Key Mapping* y *Association Table Mapping*. El primero consiste básicamente en la generación de un único ID para definir la identidad de una entidad. El segundo es utilizado para mapear relaciones de uno a muchos y el tercero para mapear relaciones de muchos a muchos.

Hibernate implementa además los patrones de comportamiento *Identity map* y *Lazy Load*. El primero es utilizado para evitar tener en memoria dos representaciones distintas del mismo objeto en una transacción de negocio. El segundo resuelve problemas de carga desmesurada y dependencias circulares o sea optimiza la carga de datos de la base de datos manteniendo en memoria sólo los que se invoquen en cada momento.

El patrón *Query object* es uno de los más importantes, pues se encarga de interpretar la estructura de objetos y traducirlos a consultas (querys) SQL. Mediante su utilización se pueden crear estas consultas, haciendo referencia a las clases y sus campos, en lugar de tablas y columnas independientemente del esquema en que puedan estar.

Existen varios elementos que componen al modelo de diseño entre los que se destacan los diagramas de clases de diseño y los diagramas de interacción. Los diagramas de clases se utilizan para visualizar las clases involucradas en el sistema y las relaciones que existen entre ellas.

Los diagramas de interacción muestran gráficamente las interacciones existentes entre instancias y clases y se utilizan para modelar aspectos dinámicos. Existen dos tipos de diagramas de interacción, el de colaboración y el de secuencia, este último muestra el intercambio de mensajes en un momento dado teniendo en cuenta el orden y el momento en que ocurren.

A partir de los elementos a tener en cuenta para la elaboración del modelo de diseño, se define una estructura de paquetes y subpaquetes que permite dividir el sistema en fracciones manejables para su futura implementación. Se utilizan los criterios de empaquetamiento: por proceso y por clases. Los paquetes que hacen referencia a los procesos se componen de subpaquetes que responden a las realizaciones de los casos de uso y los cuales contienen un diagrama de clases y los respectivos diagramas de secuencia teniendo en cuenta los escenarios. Los diagramas de clases se realizan utilizando estereotipos web.

Se crea un paquete repositorio de clases que contiene tres subpaquetes, el de las vistas con los contenidos webs referentes a las paginas clientes y los formularios que las componen, el de las sesiones compuesto por las clases controladoras autogeneradas por el entorno de desarrollo, las

clases controladoras personalizadas y las controladoras del proceso y el paquete de las entidades que contiene las clases entidades autogeneradas y personalizadas.

Estos paquetes y subpaquetes guardan relaciones entre sí, estableciendo las dependencias entre las distintas clases que contienen. A continuación se presenta el diagrama de paquetes del diseño, donde se evidencia lo antes mencionado:

Diagrama de paquetes

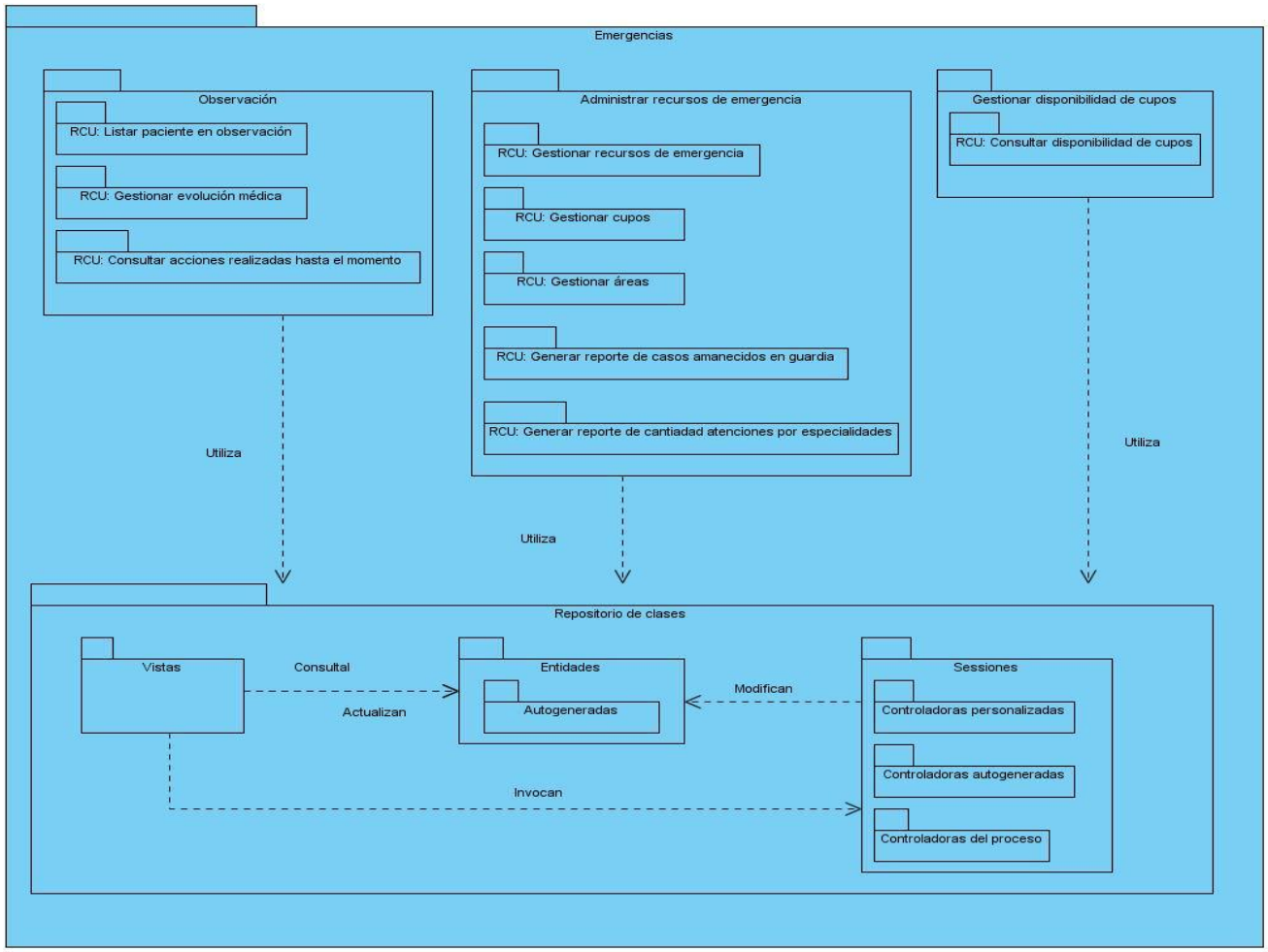
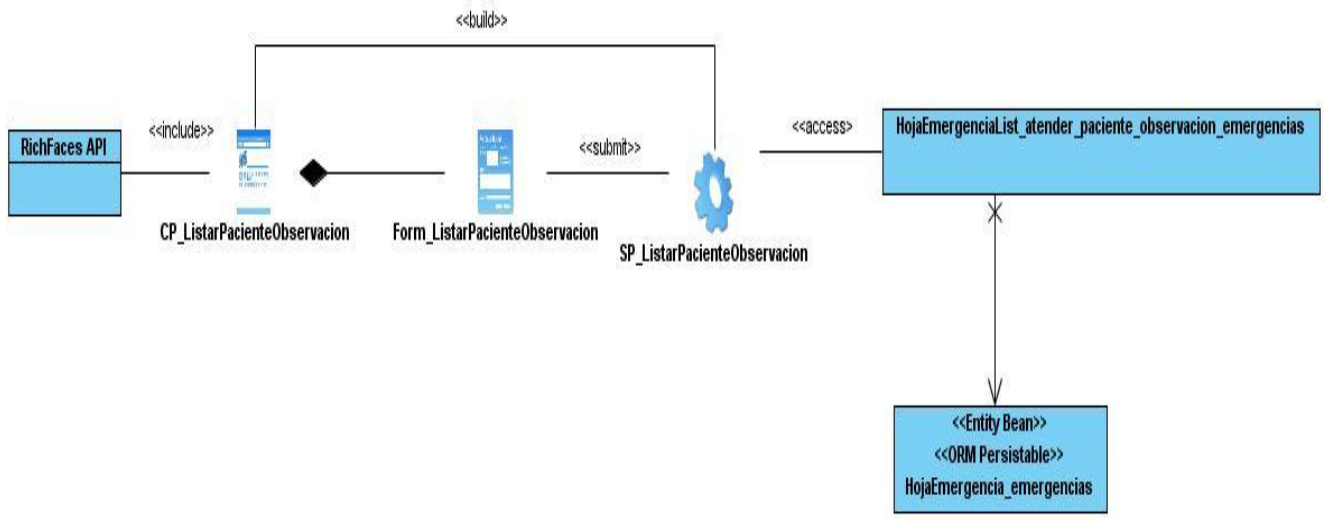


Figura 3.2.1 Diagrama de paquetes

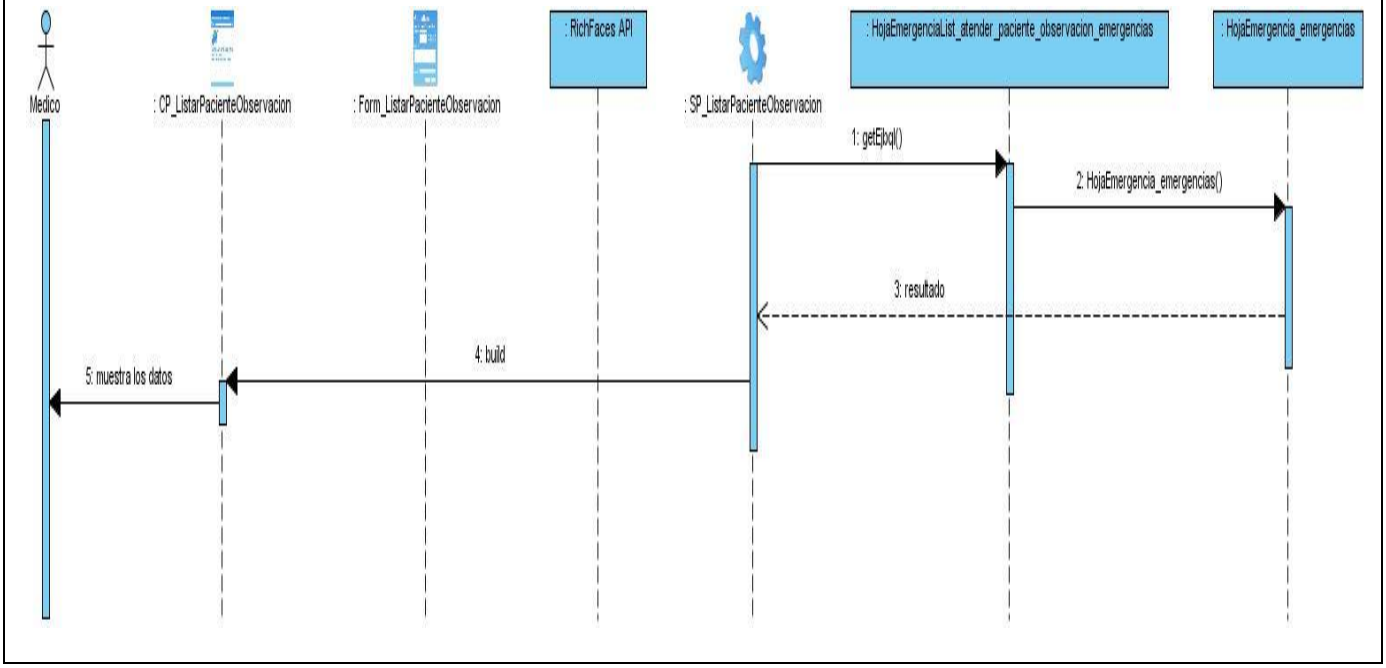
Capítulo III. Descripción y análisis de la solución propuesta

A continuación se muestran algunas de las realizaciones de los casos de usos del sistema propuesto, relacionados con los procesos Observación y Administrar recursos de emergencia:

DCD_Listar pacientes en observación



DS_Listar pacientes en observación



3.5 Descripción de las nuevas clases u operaciones necesarias

Nombre: EvolucionEmergenciaList_custom_emergencias	
Tipo de clase: Controladora	
Atributo	**Tipo**
RESTRITIONS	String
fechaDesde	Date
fechaHasta	Date
hora	String
evolucionEmergencia	EvolucionEmergencia_emergencias
evolucionEmergencia2	EvolucionEmergencia_emergencias
attribute	EvolucionEmergencia_emergencias
Para cada responsabilidad:	
Nombre:	EvolucionEmergenciaList_custom_emergencias
Descripción:	Permite listar los pacientes que se encuentran en evolución

Tabla 3.5.1 EvolucionEmergenciaList_custom_emergencias

Nombre: AtencionPacienteControlador_emergencias	
Tipo de clase: Controladora	
Atributo	**Tipo**
idPac	Int
idSolCreada	Int
idSolInterconsulta	Int
idSolTransfusion	Int
idSolAnalisisLab	Int
idOrdenMedica	Int
registrarSVitales	Boolean
registrarExamenGin	Boolean

creandoAtencionMedica	Boolean
modificando	Boolean
primeraVez	Boolean
esModif	Boolean
enfSelected	Hashtable
idEnfSelect	Intiger
idParteSeleccionada	Intiger
enfSelected2	Hashtable
idEnfSelect2	Intiger
tonoVaginaSeleccionado	String
listaTonosVaginas	List
temperaturaVaginaSeleccionada	String
listTemperaturasVaginas	List
cuPosicionSeleccionada	List
listaCUPosiciones	List
cuPresentacionSeleccionada	String
listaCUPresentaciones	List
cuPlanoSeleccionada	String
listaCUPlanos	String
cuMembranaOvularSeleccionada	String
listaMembranasOvulares	List
cuCaracteristicaLiqSeleccionada	String
listaCaracteristicasLiq	List
ubicacionSel	String
listaUbicaciones	List
caracteristicaPulsoSel	String
listaCaracteristicasPulso	List

caracteristicaFrecSel	String
listaCaracteristicasFrec	String
situacionFetoSel	String
listaSituacionesFetos	List
posicionFetoSel	String
listaPosicionesFetos	List
presentacionFetoSel	String
listaPresentacionesFetos	List
listaEspecialidades	List
parteRegionSeleccionada	Hashtable
herramientasImagenes	HerramientasImagenes
imagenUrlLesIntVieja	String
imagenUrlLesExtVieja	String
facesMessages	FacesMessages
entityManager	EntityManager
verInformacionPaciente_Controlador	VerInformacionPaciente_Controlador_emergencias
bitacora	IBitacora
-cid	Int
identity	Identity
hojaEmergencia	HojaEmergencia_emergencias
solicitudAnalisisLab	SolicitudAnalisisLab_emergencias
listaSolInterconsulta	listaSolInterconsulta_emergencias
listaSolTransfusiones	listaSolTransfusion_emergencias
signosVitales	SignosVitales_emergencias
interrogatorioEmbarazada	InterrogatorioEmbarazada_emergencias
signosVitalesFetos	SignosVitalesFetos_emergencias
examenGinecologico	ExamenGinecologico_emergencias
descAbdomenEmb	DescAbdomenEmb_emergencias

solicitudTransfusion	SolicitudTransfusion_emergencias
atencionMedica	AtencionMedica_emergencias
listaCie	Cie_emergencias
enfermedadesSeleccionadas	DiagnosticoMedicoEnfermedad_emergencias
diagAEliminar	DiagnosticoMedicoEnfermedad_emergencias
aux	DiagnosticoMedicoEnfermedad_emergencias
diagnostico	DiagnosticoMedicoEnfermedad_emergencias
listaCie2	Cie_emergencias
enfermedadesSeleccionadas2	DiagnosticoMedicoEnfermedad_emergencias
diagAEliminar2	DiagnosticoMedicoEnfermedad_emergencias
Diagnostico2	DiagnosticoMedicoEnfermedad_emergencias
listaPartesASeleccionar	ParteRegionLesionada_emergencias
listaPRL	ParteRegionLesionada_emergencias
dPRLAEliminar	DescParteRegionLesionada_emergencias
auxParteLecionada	DescParteRegionLesionada_emergencias
partesLecionadasSeleccionadas	DescParteRegionLesionada_emergencias
ordenMedica	OrdenMedica_emergencias
casoMedicoLegal	CasoMedicoLegal_emergencias
registrosTransfusionesCreadas	registroTransfusiones_emergencias
Para cada responsabilidad:	
Nombre:	AtencionPacienteControlador_emergencias
Descripción:	Constructor de la clase

Tabla 3.5.2 AtencionPacienteControlador_emergencias

Nombre: VerLesionesPacienteControlador_emergencias	
Tipo de clase: Controladora	
Atributo	**Tipo**
imagenUrlLestExt	String
imagenUrlLestInt	String
Para cada responsabilidad:	
Nombre:	VerLesionesPacienteControlador_emergencias
Descripción:	Permite ver las lesiones que presenta el paciente

Tabla 3.5.3 VerLesionesPacienteControlador_emergencias

Nombre: EvolucionPacienteControlador_emergencias	
Tipo de clase: Controladora	
Atributo	**Tipo**
-cieList	CieList_seleccionar_enfermedad_emergencias
identityManager	IdentityManager
Identity	Identity
atencionPacienteControlador	AtencionPacienteControlador_emergencias
verLecionesPacienteControlador	VerLecionesPacienteControlador_emergencias
Para cada responsabilidad:	
Nombre:	EvolucionPacienteControlador_emergencias
Descripción:	Permite realizar la evolución al paciente

Tabla 3.5.4 EvolucionPacienteControlador_emergencias

Nombre: HojaEmergenciaList_comun_emergencias	
Tipo de clase: Controladora	
Atributo	**Tipo**
RESTRITIONS	String

-hojaEmergencia	HojaEmergencia
-currentDate	Calendar
fechaDesde	Date
fechaHasta	Date
busquedaSimple	Boolean
idPacSeleccionado	Int
-listaNiveles	List
-nivelSel	String
-listaSexo	List
-sexoSel	String
Para cada responsabilidad:	
Nombre:	HojaEmergenciaList_comun_emergencias
Descripción:	Permite listar las hojas de emergencia

Tabla 3.5.5 HojaEmergenciaList_comun_emergencias

Nombre: HojaEmergenciaDesconocidoList_comun_emergencias	
Tipo de clase: Controladora	
Atributo	**Tipo**
RESTRITIONS	String
-hojaEmergencia	HojaEmergencia
-currentDate	Calendar
fechaDesde	Date
fechaHasta	Date
busquedaSimple	Boolean
idPacSeleccionado	Int
-listaNiveles	List
-nivelSel	String

-listaSexo	List
-sexoSel	String
Para cada responsabilidad:	
Nombre:	HojaEmergenciaDesconocidoList_comun_emergencias
Descripción:	Permite listar las hojas de emergencia

Tabla 3.5.6 HojaEmergenciaDesconocidoList_comun_emergencias

Nombre: VerDetallesEvolucionPacienteControlador_emergencias	
Tipo de clase: Controladora	
Atributo	**Tipo**
-cid	Int
-idEvoSeleccionada	Int
-notificarENOControlador	NotificarENOControlador_emergencias
-entityManager	EntityManager
-bitacora	IBitacora
-evolucionEmergencia	EvolucionEmergencia_emergencias
-enfDiag	DiagnosticoMedicoEnfermedad_emergencias
-verInformacionPacienteControlador	VerInformacionPacienteControlador_emergencias
Para cada responsabilidad:	
Nombre:	VerDetallesEvolucionPacienteControlador_emergencias
Descripción:	Permite ver los detalles de la evolución del paciente

Tabla 3.5.7 VerDetallesEvolucionPacienteControlador_emergencias

Nombre: VerInformacionPacienteControlador_emergencias	
Tipo de clase: Controladora	
Atributo	**Tipo**
-idHojaFrontal	Int

-idHojaEmergencia	Int
-idParto	NotificarENOControlador_emergencias
-mostrarAntecedentesHabitos	Boolean
-entityManager	EntityManager
-antecedentesFamiliaresList	AntecedentesFamiliaresList_gestion_antecedentes_ emergencias
-antecedentesPersonalesList	AntecedentesPersonalesList_gestion_antecedentes_ emergencias
-habitosPsicobiolList	HabitosPsicobiolList_gestion_antecedentes_ emergencias
-hojaFrontal	HojaFrontal_ emergencias
-hojaEmergencia	HojaEmergencia_ emergencias
-parto_emergencia	Parto_emergencias
-inmunizacionesList	InmunizacionesList gestion_antecedentes_ emergencias
Para cada responsabilidad:	
Nombre:	VerInformacionPacienteControlador_emergencias
Descripción:	Permite ver la información del pacientes

Tabla 3.5.8 VerInformacionPacienteControlador_emergencias

Nombre: NotificarENOControlador_emergencias	
Tipo de clase: Controladora	
Atributo	**Tipo**
-entityManager	EntityManager
-facesMessages	FacesMessages
-codigo	String
-cid	Int
-notificacion	Notificacion_emergencias
-diagnosticoSeleccionado	EpDiagnostico_emergencias
-hojaFrontalSeleccionada	HojaFrontal_ emergencias

-hojaFrontal		HojaFrontal_ emergencias
Para cada responsabilidad:		
Nombre:	NotificarENOControlador_emergencias	
Descripción:	Constructor de la clase	

Tabla 3.5.9 NotificarENOControlador_emergencias

Nombre: ModificarEvolucionPacienteControlador_emergencias	
Tipo de clase: Controladora	
Atributo	**Tipo**
-idEvoSeleccionada	Int
-cieList	CieList_seleccionar_enfermedad_emergencias
-entityManager	EntityManager
-identity	Identity
-atencionPacienteControlador	AtencionPacienteControlador_emergencias
-verLecionesPacienteControlador	VerLecionesPacienteControlador_emergencias
Para cada responsabilidad:	
Nombre:	ModificarEvolucionPacienteControlador_emergencias
Descripción:	Permite modificar la evolución del paciente

Tabla 3.5.10 ModificarEvolucionPacienteControlador_emergencias

Nombre: PacientesConocidosCustom	
Tipo de clase: Controladora	
Atributo	**Tipo**
-fecha	String
-cedula	String
-nombres	String
-apellido1	String

-apellido2	String
-subtotal	String
Para cada responsabilidad:	
Nombre:	PacientesConocidosCustom
Descripción:	Constructor de la clase

Tabla 3.5.11 PacientesConocidosCustom

Nombre: PacientesDesconocidosCustom	
Tipo de clase: Controladora	
Atributo	**Tipo**
-fecha	String
-noindocumentado	String
-sexo	String
-descripcion	String
-subtotal	String
Para cada responsabilidad:	
Nombre:	PacientesDesconocidosCustom
Descripción:	Constructor de la clase

Tabla 3.5.12 PacientesDesconocidosCustom

Nombre: ReportePacientesAmanecidosControlador	
Tipo de clase: Controladora	
Atributo	**Tipo**
-entityManager	EntityManager
-reportManager	ReportManager
-pathToReport	String
-sinResultados	Boolean

-fechaSeleccionada	Date
-formato	SimpleDateFormat
-subReportsList	List
-subReportsDataSourceList	List
- subReportsParameters	List
-listaPacientesAmanecidos	List
- listaPacientesDesAmanecidos	List
Para cada responsabilidad:	
Nombre:	ReportePacientesAmanecidosControlador
Descripción:	Permite generar reportes de los pacientes amanecidos en guardias dado una fecha específica

Tabla 3.5.13 ReportePacientesAmanecidosControlador

Nombre: ReporteAXEspecialidadesControlador_emergencias	
Tipo de clase: Controladora	
Atributo	**Tipo**
-pathToReport	String
-mesDesde	String
--mesHasta	String
-diaDesde	String
-diaHasta	String
-annoSeleccionado	String
-ultimoDia	Int
-ejecutar	Boolean
- mostrarSinResultados	Boolean
-listaAnnos	List
- meses	List
-subReportsList	List

-listaMeses	List
-listaEspecialidadesQuirurgicas	List
-listaEspecialidadesClinicas	List
-listaAtencionesClinicas	List
- listaAtencionesQuirurgicas	List
-subReportsDataSourceList	List
-subReportsParameters	List
-entityManager	EntityManager
-facesMessages	FacesMessages
-reportManager	ReportManager
Para cada responsabilidad:	
Nombre:	ReporteAXEspecialidadesControlador_emergencias
Descripción:	Permite generar reportes que informa la cantidad de atenciones por cada una de las especialidades en una fecha determinada

Tabla 3.5.14 ReporteAXEspecialidadesControlador_emergencias

Nombre: HojaEmergenciaList_observacion_emergencias	
Tipo de clase: Controladora	
Atributo	**Tipo**
RESTRITIONS	String
-busquedaSimple	Boolean
-idPacSeleccionado	Int
-maxResul	Int
-fechaDesde	Date
-fechaHasta	Date
-hojaEmergencia	HojaEmergencia_emergencias
Para cada responsabilidad:	
Nombre:	HojaEmergenciaList_observacion_emergencias

Descripción:	ver

Tabla 3.5.15 HojaEmergenciaList_observacion_emergencias

Nombre: HojaEmergenciaDesconocidoList_observacion_emergencias	
Tipo de clase: Controladora	
Atributo	**Tipo**
RESTRITIONS	String
-busquedaSimple	Boolean
-idPacSeleccionado	Int
-maxResul	Int
-fechaDesde	Date
-fechaHasta	Date
-listaSexos	List
-sexoSel	String
-hojaEmergencia	HojaEmergencia_emergencias
Para cada responsabilidad:	
Nombre:	HojaEmergenciaDesconocidoList_observacion_emergencias
Descripción:	ver

Tabla 3.5.16 HojaEmergenciaDesconocidoList_observacion_emergencias

Nombre: GestionRecursosEmergenciasControlados_emergencias	
Tipo de clase: Controladora	
Atributo	**Tipo**
-cuposDisponibles	Boolean
-cuposDesocupados	Boolean
-creado	Boolean
-modalPanelAbierto	String

-filtro	String
-treeAreasCupos	TreeNode
-activeModule	IActiveModule
-entityManager	EntityManager
-hojaEmergencia	HojaEmergencia_emergencias
-areaSeleccionada	AreaEmergencia_emergencias
-nuevaArea	AreaEmergencia_emergencias
-nuevoCupo	CupoEmergencia_emergencias
-cupoSeleccionado	CupoEmergencia_emergencias
Para cada responsabilidad:	
Nombre:	GestionRecursosEmergenciasControlados_emergencias
Descripción:	ver

Tabla 3.5.17 GestionRecursosEmergenciasControlados_emergencias

3.6 Modelo de datos

Es el modelo que describe de manera abstracta cómo se representan los datos de una aplicación o sistema de información. Consiste en una descripción de algo conocido como contenedor de datos, así como de los métodos para almacenar y recuperar información de esos contenedores. El Modelo de Datos tiene gran importancia en el ciclo de desarrollo de software, y de manera particular para la fase de implementación, pues define formalmente las estructuras permitidas y las restricciones que se aplican con el fin de representar los datos del dominio de la aplicación. Está compuesto por objetos que son las entidades que existen y se manipulan; y atributos que son las características básicas de dichos objetos y relaciones que es la forma en que se enlazan los objetos entre sí. [24]

A continuación se muestra parte del modelo de datos propuesto:

Modelo de datos

Figura 3.6.1 Modelo de datos.

Capítulo III. Descripción y análisis de la solución propuesta

Descripción de las tablas

Nombre: emergencias.area_emergencia		
Descripción: Tabla para registrar los datos pertenecientes a los usuarios del sistema		
Atributo	**Tipo**	**Descripción**
Id	Integer	Identificador del área
Nombre	varchar	Nombre del área
Descripción	varchar	Descripción del área
id_area	Integer	Identificador del área
Versión	Integer	Indica con que versión de la entidad se está trabajando
Eliminado	Boolean	Permite saber si el área ha sido eliminada
cid	Integer	Identificador de modificaciones registradas en la bitácora
id_funcionalidad	Integer	Identificador de la funcionalidad

Tabla 3.6.1 Emergencias.area_emergencia

Nombre: emergencias.cupo_emergencia		
Descripción: Tabla para registrar los datos pertenecientes a los usuarios del sistema		
Atributo	**Tipo**	**Descripción**
Id	Integer	Identificador del cupo
Nombre	Varchar	Nombre del cupo
Descripción	Varchar	Descripción del cupo
id_area	Integer	Identificador del área a la que pertenece el cupo
id_hoja_emergencia	Integer	Identificador de la hoja de emergencia a la que pertenece el cupo
Versión	Integer	Indica con que versión de la entidad se está trabajando

Eliminado	Boolean	Permite saber si el área ha sido eliminada
cid	Integer	Identificador de modificaciones registradas en la bitácora

Tabla 3.6.2 Emergencias.cupo_emergencia

Nombre: hc_local.evolucion_emergencia		
Descripción: Tabla para registrar los datos pertenecientes a los usuarios del sistema		
Atributo	**Tipo**	**Descripción**
Id	Integer	Identificador de la evolución
Fecha	Date	Fecha de creada la evolución
Hora	Time	Hora de creada la evolución
id_medico	Integer	Identificador del médico que realiza la evolución
id_orden_medica	Integer	Identificador de la orden médica realizada en la evolución
id_observacion	Integer	Identificador de la observación realizada en la evolución
id_signos_v	Integer	Identificador de los signos vitales realizados en la evolución
id_diagnostico	Integer	Identificador del diagnostico realizado en la evolución
id_solicitud_examen_laboratorio	Integer	Identificador de la solicitud de examen de laboratorio realizada en la evolución
id_cita_rad_imag	Integer	Identificador de las citas radiología e imagenología realizada en la evolución
Versión	Integer	Indica con que versión de la entidad se está trabajando
Eliminado	Boolean	Permite saber si el área ha sido eliminada
cid	Integer	Identificador de modificaciones

		registradas en la bitácora
Observaciones	Text	Observaciones realizadas en la observación
id_especialidad	Integer	Identificador de la especialidad
imagen_url_les_int	Text	Url de la imagen que muestra las lesiones internas del paciente
imagen_url_les_ext	Text	Url de la imagen que muestra las lesiones externas del paciente

Tabla 3.6.3 Hc_local.evolucion_emergencia

Nombre: hc_local.hoja_emergencia		
Descripción: Tabla para registrar los datos pertenecientes a los usuarios del sistema		
Atributo	**Tipo**	**Descripción**
Id	Integer	Identificador de la evolución
id_hoja_frontal	Integer	Identificador de la hoja frontal
Fecha	Date	Fecha de creada la evolución
id_trido_por	Integer	Identificador de la entidad que trajo al paciente
id_nivel_gravedad	Integer	Identificador de la gravedad que presenta el paciente
motivo_consulta	Varchar	Motivo por el cual el paciente se presenta en emergencia
id_caso_medico_legal	Integer	Identificador del caso médico legal
Hora	Time	Hora de creada la hoja emergencia
id_atencion_medica	Integer	Identificador de la atención médica
id_egreso	Integer	Identificador del egreso
id_observacion	Integer	Identificador de la observación realizada en la hoja de

		emergencia
id_fallecimiento	Integer	Identificador del fallecimiento realizada en la hoja de emergencia
id_orden_admision	Integer	Identificador de la orden de admisión realizada en la hoja de emergencia
Versión	Integer	Indica con que versión de la entidad se está trabajando
Eliminado	Boolean	Permite saber si el área ha sido eliminada
cid	Integer	Identificador de modificaciones registradas en la bitácora
id_diagnostico_final	Integer	Identificador del diagnostico realizado en la hoja de emergencia
id_notificacion_intervencion	Integer	Identificador de la notificación de la intervención realizada en la hoja de emergencia
id_diagnostico_final_recodificado	Integer	Identificador del diagnóstico final realizado en la hoja de emergencia
id_medico_diagnostica	Integer	Identificador del médico que realizada el diagnóstico en la hoja de emergencia
id_usuario_recodificador	Integer	Identificador del usuario recodificador que realiza la hoja de emergencia
id_servicio_emergencia	Integer	Identificador del servicio de emergencia que se le presta al paciente
fecha_salida_emergencias	Date	Fecha en que el paciente sale de emergencia
hora_salida_emergencias	Timetz	Hora en que el paciente sale de emergencia
id_referencia	Integer	Identificador de la entidad de la cual se refiere el paciente

Tabla 3.6.4 Hc_local.hoja_emergencia

Nombre: hc_local.hoja_frontal		
Descripción: Tabla para registrar los datos pertenecientes a los usuarios del sistema		
Atributo	**Tipo**	**Descripción**
Id	Integer	Identificador de la hoja frontal
numero_hc	Varchar	Número de la historia clínica
Foto	Varchar	Foto que identifica al paciente
Nombres	Varchar	Nombre del paciente
apellido1	Varchar	Primer apellido del paciente
Cedula	Varchar	Identificador personal del paciente
id_grupo_sanguineo	Integer	Identificador del grupo sanguíneo del paciente
fecha_nacimiento	Date	Fecha de nacimiento del paciente
id_raza	Integer	Identificador de la raza del paciente
id_etnia	Integer	Identificador del la etnia
id_nacion	Integer	Identificador de la nación
id_estado_civil	Integer	Identificador del estado civil del paciente
correo_electronico	Varchar	Correo electrónico del paciente
avisar_en_emergencia	Varchar	Lugar o persona que se puede avisar en caso de emergencia
telf_emergencia	Varchar	Teléfono que se puede avisar en caso de emergencia
id_lugar_nacimiento	Integer	Lugar de nacimiento del paciente
id_direccion_particular	Integer	Dirección particular del paciente
id_lugar_residencia	Integer	Identificador del lugar de residencia del paciente
telf_fijo	Varchar	Teléfono del paciente
telf_celular	Varchar	Teléfono celular del paciente

punto_referencia	Varchar	Punto referencia
tiempo_residencia	Integer	Tiempo en que el paciente reside
id_otra_direccion_residencia	Intiger	Identificador de otra dirección de residencia
id_ant_prenat_obst	intiger	Identificador de antecedentes prenatales obstetricia
id_periodo_neonatal	Integer	Identificador del período neonatal
id_alimentacion	Integer	Identificador de la alimentación
id_desarrollo	Integer	Identificador del desarrollo
id_ant_sex_reprod	Integer	Identificador de antecedentes
id_datos_madre	Integer	Identificador de los datos de la madre
id_datos_padre	Integer	Identificador de los datos de la padre
id_datos_representante	Integer	Identificador de los datos del representante
id_factor_sanguinea	Integer	Identificador del factor sanguíneo
apellido2	Varchar	Segundo apellido del paciente
Versión	Integer	Indica con que versión de la entidad se está trabajando
Eliminado	Boolean	Permite saber si la hoja frontal ha sido eliminada
cid	Integer	Identificador de modificaciones registradas en la bitácora
id_sexo	Integer	Identificador del sexo del paciente
id_ant_gineco_obstetrico	Integer	Identificador de antecedentes gineco-obstétrico
Profesión	Varchar	Profesión del paciente
id_tipo_hoja_frontal	Integer	Identificador del tipo de hoja frontal

id_ant_hemato_onc	Integer	Identificador de antecedente
esta_embarazada	Boolean	Para saber si se está embarazada
descripcion_desconocido	Varchar	Muestra la descripción del paciente si es desconocido
nombres_foneticos	Varchar	Nombre fonético
apellido1_foneticos	Varchar	Primer apellido fonético
apellido2_foneticos	Varchar	Segundo apellido fonético
fecha_creada	Date	Fecha en que fue creada la hoja frontal
Desconocido	Boolean	Si es desconocido el paciente

Tabla 3.6.5 Hc_local.hoja_frontal

3.7 Valoración de las técnicas de validación

Validar datos es el proceso de confirmar que los valores que se especifican en los objetos de datos son compatibles con las restricciones dentro de un esquema del conjunto de datos, al igual que las reglas establecidas para su aplicación. La validación de los datos antes de enviar actualizaciones a la base de datos subyacente es una buena práctica que reduce los errores y la cantidad potencial de acciones de ida y vuelta entre una aplicación y la base de datos.

Para confirmar que son válidos los datos que se escriben en un conjunto de datos, se puede construir comprobaciones de validación en el propio conjunto de datos. El conjunto de datos puede comprobar los datos independientemente de cómo se esté realizando la actualización, ya sea directamente mediante los controles de un formulario, desde dentro de un componente o de alguna otra manera. Dado que el conjunto de datos forma parte de la aplicación, es lógico construir una validación específica de la aplicación (a diferencia de integrar las mismas comprobaciones en el servidor de bases de datos). [25]

Hibernate Validator define una única vez las validaciones en los objetos de negocio, JBoss Seam hace uso de esta capacidad y la integra con JSF, se invocan dichas validaciones desde el punto que interese. Está plenamente internacionalizado y sus principales características son:

- ✓ Definimos las validaciones muy fácilmente con anotaciones (es posible anotar tanto los atributos como los getters).

- ✓ Trae un conjunto predefinido de validaciones típicas, conjunto que podemos extender fácilmente con nuestras propias validaciones.
- ✓ El sistema soporta internacionalización: Ya trae mensajes de error traducidos a diez idiomas. Estos mensajes los podemos cambiar fácilmente, simplemente escribiendo nuestro propio fichero de propiedades y sobrescribiendo los mensajes que nos interese.
- ✓ Se integra directamente con Hibernate, y en general con cualquier OR Mapping, de forma que antes de hacer una inserción o actualización se validarán los objetos.
- ✓ Si usamos Hibernate, las validaciones que indiquemos se tendrán en cuenta a la hora de generar el DDL (los scripts de creación de la base de datos).

3.8 Vista de implementación

La vista de implementación muestra el empaquetado físico de las partes reutilizables del sistema en unidades sustituibles, llamadas componentes, que no son más que piezas reutilizables de alto nivel a partir de las cuales se pueden construir diferentes sistemas. Esta vista tiene como propósito definir la organización del código, planificar las integraciones de sistema necesarias en cada iteración e implementar las clases y subsistemas definidos durante el diseño.

Para ilustrar la vista de implementación estática se construyen diagramas de componentes haciendo uso del lenguaje UML. Un diagrama de componentes describe la descomposición física del software en componentes, a efectos de construcción y funcionamiento. Este tiene un nivel de abstracción más elevado que un diagrama de clase, usualmente un componente se implementa por una o más clases (u objetos) en tiempo de ejecución. Estos son bloques de construcción, así eventualmente un componente puede comprender una gran porción de un sistema.

A continuación se presenta el diagrama de componentes correspondiente al sistema presentado, implementado con tres componentes fundamentales: Vista, Controlador y Modelo, que a su vez incluyen otros componentes.

Diagrama de componentes

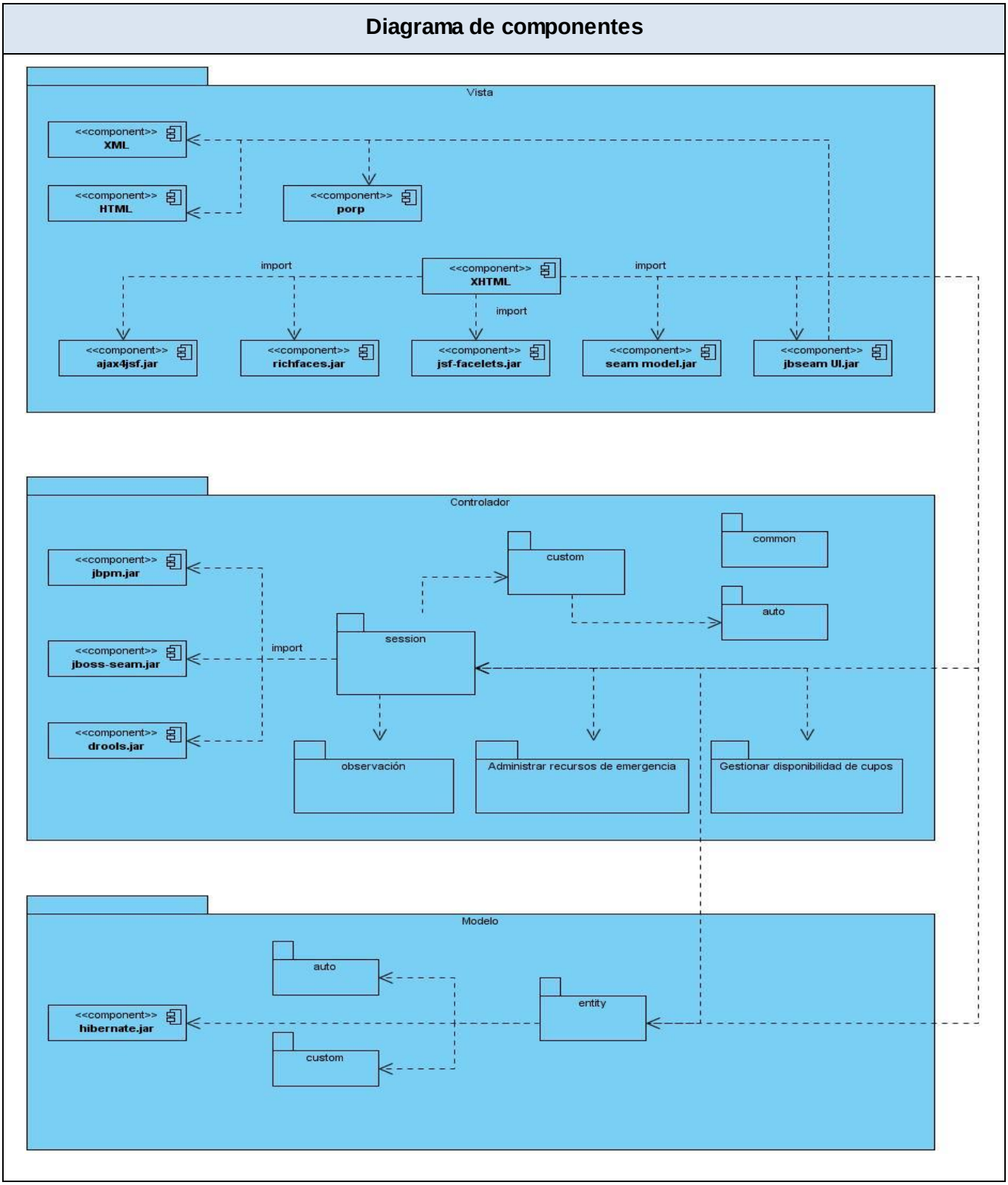


Figura 3.7.1 Diagrama de componentes

Capítulo III. Descripción y análisis de la solución propuesta

Descripción de los algoritmos no triviales a implementar

Un algoritmo puede definir como una secuencia de instrucciones que representan un modelo de solución determinado tipo de problemas. O bien como un conjunto de instrucciones que realizadas en orden conducen a obtener la solución. Estos pueden escribirse y luego ejecutarse son independientes de los lenguajes de programación, sin importar el tipo de lenguaje que se utilice.

Para solucionar problemas mediante la realización de un algoritmo, normalmente, se analiza el problema desde distintos puntos de vista, aplicando diferentes estrategias, y por tanto, llegando a soluciones algorítmicas distintas.

Cuando se desea resolver un problema de manera computacional, se comparan estas soluciones algorítmicas para conocer cómo se comportarán cuando se implementen, especialmente si son problemas grandes. La complejidad algorítmica es una métrica teórica que se aplica a los algoritmos en este sentido. Es un concepto fundamental para todos los programadores.

Para lograr el desarrollo de los procesos observación y administración de recursos, del módulo emergencias del sistema de información hospitalaria alas HIS se implementaron algoritmos de alta complejidad como son:

- ✓ Listar paciente en observación.
- ✓ Crear evolución.
- ✓ Seleccionar hoja de emergencia de paciente en observación.
- ✓ Consultar acciones realizadas en evolución.
- ✓ Modificar hoja de evolución.
- ✓ Generar informe de estadísticas.
- ✓ Generar reporte de casos amanecidos en guardia.
- ✓ Ver libro de emergencia.
- ✓ Administrar recursos de la emergencia.

Capítulo III. Descripción y análisis de la solución propuesta

Selección de las estructuras de datos apropiadas para la implementación de estos algoritmos

Una estructura de datos es una forma de organizar un conjunto de datos elementales con el objetivo de facilitar su manipulación. Lo que se pretende es facilitar un esquema lógico para manipular los datos en función del problema que haya que tratar y el algoritmo para resolverlo. En ella se identifican dos niveles, en el primero se identifican la colección de elementos a agrupar y en el segundo se piensa en el lenguaje de programación en específico que se va a utilizar.

El lenguaje de programación java contiene un paquete java.util que consta de 34 clases y 13 interfaces que implementan algunas de las estructuras de datos más comunes. Una de estas estructuras es el Arraylist que es una lista con las que se ordena correctamente un conjunto de elementos, la misma tiene varios métodos y es una clase que implementa le interfaz.

Descripción de las clases que se utilicen para representar computacionalmente dicha estructura

Las aplicaciones frecuentemente necesitan almacenar un grupo de datos en un sólo objeto. Los arrays sirven bien para este propósito, pero algunas veces necesitamos incrementar o reducir dinámicamente el número de elementos del array, o hacer que contenga distintos tipos de datos. Estos tienen varios constructores, dependiendo de cómo necesitemos construir, sólo contiene referencias a objetos.

Para almacenar tipos primitivos como double, long, o float, se utiliza una clase envoltura. ArrayList es una de las muchas clases del Collection Framework, que proporciona un conjunto de interfaces y clases bien diseñados para almacenar y manipular grupos de datos como una sola unidad, una colección. En caso de que sea necesario circular a través de los elementos del ArrayList, se realiza a través de la clase Iterator y sus métodos hasNext y next.

Esta clase provee una serie de métodos para manipular la lista, seguidamente se explican algunos de estos métodos:

- ✓ Size: devuelve el número de elementos en esta lista.
- ✓ toArray: Devuelve una matriz que contiene todos los elementos en esta lista en el orden correcto.
- ✓ Remove: Elimina el elemento en la posición especificada en esta lista. Cambios de todos los elementos posteriores a la izquierda.
- ✓ isEmpty: Comprueba si esta lista no tiene elementos.
- ✓ indexOf: Las búsquedas de la primera aparición de los argumentos dados, las pruebas para la igualdad mediante el método equals.

- ✓ get: Devuelve el elemento en la posición especificada en esta lista.
- ✓ add: Inserta el elemento especificado en la posición especificada en esta lista. Cambia el elemento actualmente en esa posición y todos los elementos posteriores a la derecha.
- ✓ hasNext: Devuelve verdadero si existen más objetos en la lista y falso en caso contrario. Siempre, antes de invocar la recuperación del siguiente objeto, debe consultarse si existen más objetos en la lista en dicha dirección
- ✓ next: Recupera el siguiente objeto de la lista. Invocando repetidamente este método se visitan todos los objetos de la lista, uno a uno.

Conclusiones

En este capítulo se realizó una valoración del diseño propuesto por el analista de sistemas y se especificó la estructura de este teniendo en cuenta las realizaciones de los casos de usos con sus clases correspondientes. De cada clase se realizó una breve descripción especificando los atributos y métodos asociados. Además, se obtuvo el modelo de datos del sistema presentado y se elaboró el diagrama de componentes estrechamente relacionado con el modelo de diseño.

Capítulo IV. MODELO DE PRUEBA

En el presente capítulo se describe el método de caja negra para probar las funcionalidades del sistema propuesto. Además, se definen y detallan los casos de pruebas asociados a las funcionalidades a implementar.

4.1 Pruebas de caja negra

Para obtener un software excelente es necesario que se tenga un buen modelo de pruebas. Con las pruebas no se puede asegurar la ausencia de defectos en un software, pero si se puede demostrar que presenta defectos.

Un buen diseño de prueba, sistemáticamente saca a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo. Para garantizar la efectividad de las pruebas, deben ser realizadas por un equipo independiente al que realizó el sistema. [26]

Las pruebas de **caja negra** se llevan a cabo sobre la interfaz del software y son completamente indiferentes al comportamiento interno y la estructura del programa. Con estas se pretende demostrar que las funcionalidades del sistema son operativas, que la entrada se acepta de forma adecuada, que se produce un resultado correcto y que se mantiene la integridad de la información externa.

Para desarrollar la prueba de caja negra existen varias técnicas, entre ellas están:

- ✓ Técnica de la partición de equivalencia.
- ✓ Técnica del análisis de valores límites.
- ✓ Técnica de grafos de causa-efecto. [27]

Técnica de partición de equivalencia

Divide el dominio de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. El diseño de estos casos de prueba para esta partición se basa en la evaluación de las clases de equivalencia para una condición de entrada. Una clase de equivalencia representa un conjunto de estados válidos o inválidos para condiciones de entrada y regularmente estas condiciones pueden ser un valor numérico, un rango de valores, un conjunto de valores relacionados o una condición lógica.

Técnica del análisis de valores límites

Los casos de prueba que exploran las condiciones límite producen mejor resultado que aquellos que no lo hacen. Las condiciones límite son aquellas que se hallan en los márgenes de la clase de equivalencia, tanto de entrada como de salida. La técnica del análisis de valores límites complementa

a la de partición equivalente, pues en lugar de centrarse solamente en las condiciones de entrada, deriva los casos de prueba también para el campo de salida.

Técnica de grafos de causa-efecto

En este método se debe entender los objetos (objetos de datos, objetos de programa tales como módulos o colecciones de sentencias del lenguaje de programación) que se modelan en el software y las relaciones que conectan a estos objetos. Una vez que se ha llevado a cabo esto, el siguiente paso es definir una serie de pruebas que verifiquen que todos los objetos tienen entre ellos las relaciones esperadas. En este método:

- ✓ Se crea un grafo de objetos importantes y sus relaciones.
- ✓ Se diseña una serie de pruebas que cubran el grafo de manera que se ejerciten todos los objetos y sus relaciones para descubrir errores.

4.2 Descripción de los casos de prueba

Caso de uso: Listar paciente en observación

Escenarios del listar pacientes en observación	**Descripción de la funcionalidad**	**Flujo Central**
EC 1: Listar pacientes en observación	Mostrar el listado de los pacientes en observación satisfactoriamente.	Muestra un listado de los pacientes que cumplen con los criterios de búsqueda.
EC 2: No se encuentra información	No se encuentra información que cumpla con los criterios de búsqueda.	Muestra el mensaje de información “No se encontró información que cumpla con los criterios de búsqueda.”
EC 3: Ordenar	Permite ordenar el resultado ascendente o descendentemente por un atributo.	Muestra un listado de los pacientes. Se ordenan los campos del listado por atributos.
EC 4: Realizar búsqueda	Buscar un paciente dado criterios.	Muestra la interfaz Listar pacientes en observación. Se introducen los datos correspondientes a la búsqueda simple. Se selecciona la opción Buscar.

		Muestra un listado de los pacientes que cumplen con los criterios de búsqueda.
EC 5: Cancelar operación	Cancelar la opción de Buscar pacientes.	Muestra la interfaz Listar pacientes en observación. Se introducen los datos correspondientes a la búsqueda simple. Se selecciona la opción Cancelar. Se regresa a la vista anterior.
EC 6: Seleccionar paciente	Seleccionar un paciente satisfactoriamente.	Muestra un listado de los pacientes que cumplen con los criterios de búsqueda. Se selecciona la opción Seleccionar paciente. Ver DCP **Seleccionar hoja de emergencia de paciente en observación.**
EC 7: Adicionar evolución	Permite acceder al caso de uso registrar evolución.	Muestra la interfaz para registrar la evolución. Se introducen los datos correspondientes. Se selecciona la opción Adicionar. Ver DCP **Registrar evolución**.

Sección: Listar paciente en observación.

Id del escenario	EC 1	EC 2	EC 3	EC 4	EC 5	EC 6	EC 7
Escenario	Listar pacientes en observación	No se encuentra información	Ordenar	Realizar búsqueda	Cancelar operación	Seleccionar paciente	Adicionar evolución

Variable 1 (Carné Identidad)	NA	NA	NA	V			
Variable 2 (Nombre)	NA	NA	NA	V			
Variable 3 (Primer apellido)	NA	NA	NA	V			
Variable 4 (Segundo apellido)	NA	NA	NA	V			
Variable 5 (Fecha desde)	NA	NA	NA	V			
Variable 6 (Fecha hasta)	NA	NA		V			
Botón 1 (Buscar)				NA			
Botón 2 (Cancelar)					NA		
Botón 3 (Seleccionar)						NA	
Botón 4 (Adicionar)							NA
Respuesta del Sistema	Muestra: El listado de pacientes en observación	Muestra el mensaje de información: "No se encuentra información que cumpla con los criterios de búsqueda.".	Muestra los atributos del listado ordenados	Muestra: El listado de paciente que coinciden con el criterio de búsqueda • Cédula (código)	Regresa a la vista anterior	Muestra la interfaz para seleccionar el paciente	Muestra la interfaz para adicionar evolución
Resultado de la Prueba							

Capítulo IV. Modelo de prueba

Conclusiones

En este capítulo se realizó el modelo de prueba. Atendiendo al método prueba, caja negra, obteniéndose los casos de prueba como artefactos generados del flujo de trabajo.

Conclusiones

CONCLUSIONES

Con el desarrollo de los procesos de observación y administración de recursos del módulo Emergencias se arribaron a las siguientes conclusiones:

- ✓ Los sistemas analizados no contemplan todas las funcionalidades que se llevan a cabo durante la observación al paciente y la administración de los recursos en el área de Emergencias.
- ✓ El análisis de los procesos y la evaluación de las funcionalidades definidas posibilitó realizar el diseño de las clases asociadas a la aplicación.
- ✓ El empleo de una arquitectura definida permitió el desarrollo de un sistema flexible y robusto.
- ✓ La utilización de las pautas definidas garantizan uniformidad visual de todas las interfaces del sistema.
- ✓ La realización del diseño, en correspondencia con la arquitectura facilitó la implementación de los procesos observación y administración de los recursos del área de Emergencias.

De esta manera, se ha cumplido con el objetivo y las tareas trazadas en el trabajo de diploma, por lo que se obtuvo un sistema informático, que permite mejorar los procesos de gestión de la información en el área de Emergencias en las instituciones hospitalarias.

Recomendaciones

RECOMENDACIONES

Los autores recomiendan para futuras versiones del sistema:

Implementar funcionalidades que permitan generar reportes estadísticos de forma genérica, donde el personal médico pueda decidir la información que desea incluir en ellos.

Referencias bibliográficas

REFERENCIAS BIBLIOGRÁFICAS

1. Kerdel-Vegas, Francisco. Aplicaciones de las TIC en el sector de la salud del futuro. Fundación Telefónica. Julio de 2009. Disponible en: http://bitacoramedica.com/weblog/wp-content/uploads/2009/09/Telefonica_TIC_Cap_II.pdf.

2. Sistema de Información Hospitalaria. México D.F.: Universidad Autónoma de México. D. R. Facultad de Medicina, 2003. p 7. Disponible en: http://www.facmed.unam.mx/emc/computo/ssa/HIS/his.pdf

3. CNT Sistema de Información S.A. Disponible en: http://www.cnt.com.co/pagina/index.asp?id=177

4. Sistema de Información para Gerencia Hospitalaria. Disponible en: http://www.sigho.gob.mx/

5. PRIDES. Disponible en: http://www.prides.net/salud.aspx?descpro=em

6. ISOFT. Disponible en: http://www.isoftsanidad.es/dummy/xHIS_080108.pdf

7. Diaz Lauzurica, Belkis. Mayo 25 de 2001. Sistema Informático para el control de la urgencia médica. Memorias II Congreso Latinoamericano de Ingeniería Biomédica.

8. Buschmann, Frank; Meunier, Regine; Rohnert Hans, Sommerlad Peter; Stal, John Michael. Pattern-Oriented Software Architecture. 1996.

9. Márquez Avendaño, B. M., Zulaica Rugarcía. Implementación de un reconocedor de voz gratuito al sistema de ayuda a invidentes Dos-Vox en español. Tesis Licenciatura. Ingeniería en Sistemas Computacionales. Departamento de Ingeniería en Sistemas Computacionales, Escuela de Ingeniería, Universidad de las Américas Puebla. Enero 2004. Disponible en: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/marquez_a_bm/

10. Reynoso, Carlos y Kicillof, Nicolás. Marzo del 2004. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. 2004. Disponible en: http://www.willydev.net/descargas/prev/Estiloypatron.pdf

11. UI Develoment with JaveServer Faces. Disponible en: http://www.itk.ilstu.edu/faculty/bllim/itk353/j-jsf-ltr.pdf

12. Katz, Max. Practical Richfaces. Apress. 2008. p 12.

Referencias bibliográficas

13. Ajax4jsf Developer Guide. 2007. Disponible en: http://www.jboss.org/file-access/default/members/jbossajax4jsf/freezone/docs/devguide/en/html_single/index.html

14. Hookom, Jacob. (17 de Agosto de 2005). JSF Central tm. Inside Facelets Part 1: An Introduction. Disponible en: http://www.jsfcentral.com/articles/facelets_1.html

15. Allen, Dan. Seam in action. Manning Early Access Program. Manning Publications Co. 2008. p 4, p 5, p 6. Disponible en: http://www.docstoc.com/docs/5134034/ManningSeaminActionSep2008

16. JavaHispano (7 de Julio de 2006). Disponible en: http://www.javahispano.org/contenidos/es/liberado_drools_3_0/

17. Ort, R. B. (Mayo de 2006). Sun microsystems. The Java Persistence API - A Simpler Programming Model for Entity Persistence. Disponible en: http://java.sun.com/developer/technicalArticles/J2EE/jpa/

18. Franky, María Consuelo. (Abril de 2007). Java EE 5 (sucesor de J2EE). Disponible en: http://www.acis.org.co/fileadmin/Conferencias/ConfConsueloFranky_Abr19.pdf

19. Ottinger, Joseph. (25 de Junio de 2007). *TheServerSide.com*. JBoss releases JBoss Tools, Eclipse Plugins including Exadel. Disponible en: http://www.theserverside.com/news/thread.tss?thread_id=45933

20. García de Jalón Javier, Rodríguez Iñigo Mingo José Ignacio, Alfonso Brazález Aitor Imaz, Larzabal Alberto, Calleja Jesús, García Jon. Aprenda Java como si estuviera en primero. Escuela Superior de Ingenieros Industriales. Universidad de Navarra. España. Enero 2000.

21. Itera. (29 de Julio de 2008). Rational Unified Process. Disponible en: http://www.iteraprocess.com/index.php?option=com_content&task=view&id=18&Itemid=42

22. Informática Profesional. Disponible en: http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=ireport

23. Especificaciones de requerimientos. Disponible en: http://www.mitecnologico.com/Main/EspecificacionesDeRequerimientos

24. Presman Roger S. Ingeniería de Software, un enfoque práctico. McGraw-Hill. 2001. http://www.hl7spain.org/VerPagina.asp?IDPage=0

Referencias bibliográficas

25. Información general sobre validación de datos. Disponible en: http://msdn.microsoft.com/es-es/library/kx9x2fsb%28VS.80%29.aspx

26. Juristo Natalia, Moreno Ana M., Vegas Sira. TÉCNICAS DE EVALUACIÓN DE SOFTWARE. 17 de octubre de 2006.

27. R. S. Pressman. Ingeniería del software. Un enfoque práctico. 1997

28. La prueba del software. Disponible en: http://eisc.univalle.edu.co/materias/Material_Desarrollo_Software/Pruebas.pdf

Bibliografía

BIBLIOGRAFÍA

- Ajax4jsf Developer Guide. 2007. Disponible en: http://www.jboss.org/file-access/default/members/jbossajax4jsf/freezone/docs/devguide/en/html_single/index.html
- Allen, Dan. Seam in action. Manning Early Access Program. Manning Publications Co. 2008. p 4, p 5, p 6. Disponible en: http://www.docstoc.com/docs/5134034/ManningSeaminActionSep2008
- ARNOLD, KEN; GOSLING, JAMES; HOLMES, DAVID. El lenguaje de Programación Java. Addison Wesley. 2009.Disponible en: http://www.lenguajes-de-programacion.com/programacion-java.shtml
- Buschmann, Frank; Meunier, Regine; Rohnert Hans, Sommerlad Peter; Stal, John Michael. Pattern-Oriented Software Architecture. 1996.
- Departamento de Lenguajes y Sistemas Informáticos. Tema 1: Patrones Arquitectónicos. Escuela Técnica Superior de Ingeniería Informática. Universidad de Sevilla. Disponible en: http://www.lsi.us.es/docencia/get.php?id=1891
- Diaz Lauzurica, Belkis. Mayo 25 de 2001. Sistema Informático para el control de la urgencia médica. Memorias II Congreso Latinoamericano de Ingeniería Biomédica.
- Especificaciones de requerimientos. Disponible en: http://www.mitecnologico.com/Main/EspecificacionesDeRequerimientos
- Franky, María Consuelo. (Abril de 2007). Java EE 5 (sucesor de J2EE). Disponible en: http://www.acis.org.co/fileadmin/Conferencias/ConfConsueloFranky_Abr19.pdf
- GARCÍA, JOAQUÍN. Patrones de diseño. 27 de Mayo de 2005 Disponible en: http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php
- García de Jalón Javier, Rodríguez Iñigo Mingo José Ignacio, Alfonso Brazález Aitor Imaz, Larzabal Alberto, Calleja Jesús, García Jon. Aprenda Java como si estuviera en primero. Escuela Superior de Ingenieros Industriales. Universidad de Navarra. España. Enero 2000.
- Gutiérrez, Rodolfo. El uso imprescindible del modelo cliente/servidor. La revista del empresario cubano. Disponible en: http://www.betsime.disaic.cu/secciones/tec_abr_02.htm#1

Bibliografía

- Hennebrueder, Sebastian. (15 de Marzo de 2006). Lalíluna. Java tutorials and development. First EJB 3 Tutorial showing a session and entity beans with annotations and JBoss. Disponible en: http://www.lalíluna.de/ejb-3-tutorial-jboss.html

- Hookom, Jacob. (17 de Agosto de 2005). JSF Central tm. Inside Facelets Part 1: An Introduction. Disponible en: http://www.jsfcentral.com/articles/facelets_1.html

- Información general sobre validación de datos. Disponible en: http://msdn.microsoft.com/es-es/library/kx9x2fsb%28VS.80%29.aspx

- Informática Profesional. Disponible en: http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=ireport

- Itera. (29 de Julio de 2008). Rational Unified Process. Disponible en: http://www.iteraprocess.com/index.php?option=com_content&task=view&id=18&Itemid=42

- Javid Jamae, Peter Johnson. JBoss in Action. s.l.: Manning Publications, enero 2009. 1933988029.

- JavaHispano (7 de Julio de 2006). Disponible en: http://www.javahispano.org/contenidos/es/liberado_drools_3_0/

- Juristo Natalia, Moreno Ana M., Vegas Sira. TÉCNICAS DE EVALUACIÓN DE SOFTWARE. 17 de octubre de 2006.

- Katz, Max. Practical Richfaces. Apress. 2008. p 12.

- Kerdel-Vegas, Francisco. Aplicaciones de las TIC en el sector de la salud del futuro. Fundación Telefónica. Julio de 2009. Disponible en: http://bitacoramedica.com/weblog/wp-content/uploads/2009/09/Telefonica_TIC_Cap_II.pdf.

- La prueba del software. Disponible en: http://eisc.univalle.edu.co/materias/Material_Desarrollo_Software/Pruebas.pdf

- Márquez Avendaño, B. M., Zulaica Rugarcía. Implementación de un reconocedor de voz gratuito al sistema de ayuda a invidentes Dos-Vox en español. Tesis Licenciatura. Ingeniería en Sistemas Computacionales. Departamento de Ingeniería en Sistemas Computacionales, Escuela de Ingeniería, Universidad de las Américas Puebla. Enero 2004. Disponible en: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/marquez_a_bm/

Bibliografía

- Ort, R. B. (Mayo de 2006). Sun microsystems. The Java Persistence API - A Simpler Programming Model for Entity Persistence. Disponible en: http://java.sun.com/developer/technicalArticles/J2EE/jpa/

- Ottinger, Joseph. (25 de Junio de 2007). TheServerSide.com. JBoss releases JBoss Tools, Eclipse Plugins including Exadel. Disponible en: http://www.theserverside.com/news/thread.tss?thread_id=45933

- Presman Roger S. Ingeniería de Software, un enfoque práctico. McGraw-Hill. 2001. Disponible en: http://www.hl7spain.org/VerPagina.asp?IDPage=0

- R. S. Pressman. Ingeniería del software. Un enfoque práctico. 1997

- Reynoso, Carlos y Kicillof, Nicolás. Marzo del 2004. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. 2004.

  Disponible en: http://www.willydev.net/descargas/prev/Estiloypatron.pdf
- SAAVEDRA, JORGE. PATRONES GRASP (Patrones de Software para la asignación General de Responsabilidad). Parte 2. Mayo 2007.
  Disponible en: http://jorgesaavedra.wordpress.com/category/patrones-grasp/

- The postgreSQL Global Develoment Group. PostgreSQL 8.3.6 Documentation. Disponible en: http://www.enterprisedb.com/docs/en/8.3/pg/index.html

- Sistema de Información Hospitalaria. México D.F.: Universidad Autónoma de México. D. R. Facultad de Medicina, 2003. p 7. Disponible en: http://www.facmed.unam.mx/emc/computo/ssa/HIS/his.pdf

- UI Develoment with JaveServer Faces. Disponible en: http://www.itk.ilstu.edu/faculty/bllim/itk353/j-jsf-ltr.pdf

Glosario de términos

GLOSARIO DE TÉRMINOS

AJAX: Acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

API: Interfaz de Programación de Aplicaciones, cuyo acrónimo en inglés es API (Application Programming Interface), es un conjunto de funciones residentes en bibliotecas generalmente dinámicas. Permiten que una aplicación corra bajo un determinado sistema operativo.

Bean: Es un componente hecho en software que se puede reutilizar y que puede ser manipulado visualmente por una herramienta de programación en lenguaje Java.

CIE10: Es la décima versión de la Clasificación Estadística Internacional de Enfermedades y otros Problemas de Salud. Provee los códigos para clasificar las enfermedades y una amplia variedad de signos, síntomas, hallazgos anormales, denuncias, circunstancias sociales y causas externas de daños y/o enfermedad. Cada condición de salud puede ser asignada a una categoría y darle un código de hasta seis caracteres de longitud.

CIE-9-MC: Es la traducción oficial de ICD-9-CM (International Classification of Diseases, Ninth Revision, Clinical Modification) que a su vez es una adaptación de la ICD-9 (International Classification of Diseases, Ninth Revision). La ICD-9-CM fue creada para facilitar la codificación de morbimortalidad en los hospitales. Se trata de una clasificación de enfermedades y procedimientos utilizada en la codificación de información clínica derivada de la asistencia sanitaria, principalmente en el entorno de hospitales y centros de atención médica especializada.

Framework: Es una estructura de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado.

HL7: (Health Level Seven) es un conjunto de estándares para el intercambio electrónico de información médica. Los estándares HL7 son desarrollados por la organización ANSI del mismo nombre.

Glosario de términos

Multiplataforma: Es un término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de software, que puedan funcionar en diversas plataformas.

WSDL: Es un lenguaje que está basado en XML y que permite la descripción de los servicios web desplegados. WSDL se utiliza también para la localización y ubicación de estos servicios en Internet.