Universidad Central “Marta Abreu” de Las Villas

LaparoGest
Sistema para la gestión de información acerca de intervenciones de Colecistectomía Laparoscópica basado en plataforma Software Libre.

Tesis Presentada en Opción al Título Académico de
Máster en Computación Aplicada

Autor: Lic. Héctor A. Bernal Suárez
Tutor: Dra. Lydia Rosa Ríos

2010

RESUMEN

El presente trabajo aborda el tema de la gestión de información hospitalaria en la rama de las intervenciones de Colecistectomía Laparoscópica. Se centra en el seguimiento, mediante tecnología informática, de los diferentes estadíos (preoperatorio, transoperatorio y postoperatorio) por los que transita un paciente remitido a dicha especialidad. Tiene como objetivo fundamental: Diseñar e implementar una aplicación informática que gestione la información del seguimiento a paciente de la especialidad de Colecistectomía Laparoscópica e ir nutriendo la base informativa para un futuro almacén de datos que a largo plazo le permitirá tomar decisiones y llegar a conclusiones basado en el conocimiento acumulado. Para la confección del software se utilizó tecnología basada en software libre (Java y PostgreSQL). Permitió concluir que las posibilidades de empleo de las tecnologías de la información y las comunicaciones (TICs) en el campo de la medicina son muy amplias y que el uso de herramientas automatizadas de gestión hospitalaria permite no sólo agilizar el trabajo sino humanizarlo, ganar en confiabilidad y confidencialidad en el manejo de la información.

ABSTRACT

The present paper approaches the topic of hospital data management in the field of Laparoscopic Cholecystectomyc interventions. It is mainly focused in tracking, by means of computer technology, the different estadíoses (preoperative, transoperative and postoperative) through which a patient remitted to the specialty should overcome. It has a main objective: to provide the user with a tool that allows him to store and consult data related to the patients and to go feeding up a future data warehouse, which in a long term, will allow him to make decisions and to reach out for conclusions based on the cumulative knowledge. A free software-based technology has been chosen to implement the software (Java and PostgreSQL). This project allowed to conclude that the possibilities of using communications and information technologies (ICT) in a medical environment are wide and that the use of computer-based hospital management tools not only allows speeding up the work but also humanizing it, to win in dependability and confidentiality in data handling.

Agradecimientos

A todos los que a lo largo de 10 años entre computadoras han aportado granitos o camiones de arena para el camino hasta aquí, especialmente a:

- Mi familia y mi pareja por su apoyo y fortaleza de cada día, por el “dale que tu puedes” acompañado de un beso.
- Mis compañeros de trabajo desde la ESI hasta ahora, especialmente en esta última etapa Niriam, Lian y Lídice, por la paciencia, el tiempo y el conocimiento incondicionalmente puestos a mi disposición.
- El colectivo de profesores de la 9na Edición de Computación Aplicada que aportaron un caudal de conocimiento invalorable.
- Mi tutora Lydia, por la ayuda sonriente de siempre.
- El Guille, compañero de trabajo, travesía y estudio.
- Todos los amigos.
- Loli, el hada madrina de los sin techo en UCLV.

INTRODUCCIÓN

Las tecnologías de la información y las comunicaciones (TICs) han pasado a jugar un rol de vital importancia en diferentes sectores de la vida moderna en Cuba, sin embargo aún les resta un universo de retos y posibilidades no explotadas en lo que refiere a su empleo en el sector de la salud. Desde hace varias décadas las computadoras ayudan a los profesionales de la medicina en su lucha por la salud y contra la enfermedad, facilitan la gestión en grandes hospitales, la investigación científica, la docencia médica, en el diagnóstico y tratamiento y en el procesamiento de estadísticas médicas en casos de epidemias. Por otro lado la combinación con la microelectrónica, las telecomunicaciones y las técnicas para el procesamiento de datos permite que estas actividades médicas potencialicen sus logros.(García., 2006)

La información clínica y de salud

La información relativa al estado de salud de un ciudadano que está íntimamente ligada a su ciclo de vida, se genera cuando se produce un contacto con un profesional de la salud si queda debidamente documentado. Estos contactos con el sistema sanitario se producen con ocasión de exámenes de salud, revisiones médicas, programas de detección, reconocimientos médicos laborales e incluso la necropsia o investigaciones forenses.(Azcárate, 2006)

El concepto de información clínica, por lo tanto, se refiere a todo dato, cualquiera que sea su forma, clase o tipo, que permite adquirir o ampliar conocimientos sobre el estado de salud de una persona, o la forma de preservarla, cuidarla, mejorarla o recuperarla.

Informática Médica

La Informática Médica por su parte tiene que ver con la captura, almacenamiento, recuperación, comunicación y uso ético de la información y el conocimiento acerca de la salud (Padrón., 2007) . Su empleo para la solución de los problemas en todas las esferas de las ciencias biomédicas y en la gestión de las instituciones de salud crece a pasos agigantados tanto en países altamente industrializados como los que están en vía de desarrollo. Tiene un dominio de aplicación amplio que incluye tanto el manejo de los datos de los pacientes como a los procesos a través de los cuales se desarrollan el diagnóstico y el tratamiento médico, la gestión de la información médica, la enseñanza de las ciencias médicas, la investigación biomédica y la gerencia de salud.

Algunas aplicaciones específicas son (García., 2006):

- tratamiento automatizado de historias clínicas.

- sistemas automatizados para el control de las estadísticas sanitarias
- sistemas expertos para la ayuda al diagnóstico y tratamiento de diversas enfermedades.
- tutoriales y entrenadores para la enseñanza de las ciencias médicas.
- sistemas inteligentes para la ayuda a la gerencia de salud.

La informatización del Sistema Nacional de Salud Pública (SNS) en Cuba está dada por el conjunto de métodos, técnicas, procederes y actividades gerenciales dirigidas al manejo de la información en salud, la cual comprende la información sobre el estado de salud de la población, la información sobre el conocimiento de las ciencias de la salud y la información en general para la toma de decisiones, clínico-epidemiológicas, operativas y estratégicas.

Durante los últimos 20 años un grupo de instituciones cubanas han desarrollado sistemas encaminados a lograr determinados niveles de informatización de la salud. Estas soluciones carecían de integración y de una definición generalizable, aparte de que no existían los recursos tecnológicos necesarios para su ejecución en el SNS. A partir de 1997 se concibe una primera estrategia de informatización como respuesta del sector de la salud a los lineamientos estratégicos para la informatización de la sociedad cubana, con la finalidad de coordinar esfuerzos para el desarrollo de este proceso en el SNS.

Actualmente el Ministerio de Salud Pública (MINSAP) ha definido a la informatización como una de sus prioridades y ha convocando para ello a un grupo de instituciones propias de sector, del Ministerio de Informática y Comunicaciones y de otros organismos de la administración central del estado, para trazar de conjunto la estrategia a desarrollar. En algunos casos se ha tomado como punto de partida sistemas ya desarrollados en el país en el marco de aquella primera estrategia de desarrollo.

Para el público en general, cuando se le menciona el uso de tecnologías de avanzada en la salud se representa equipos de diagnóstico tales como el TAC, Escáneres, Unidades de Cuidados Intensivos, etc. Sin embargo cuando de información se trata nos encontramos con que por ejemplo la Historia Clínica de un paciente es aún en la mayoría de nuestros centros un legajo de papeles.

Tratamiento informatizado de historias clínicas.

Se ha reconocido ampliamente que la incorporación de las historias clínicas electrónicas (HCE) a los sistemas de salud redunda en ventajas desde el punto de vista asistencial, docente- investigativo y administrativo. Algunos investigadores llegan a afirmar que las

HCE se convertirán en el corazón de los sistemas de salud del futuro. En otros trabajos se llega a afirmar que la HCE puede redundar en más beneficios para la sociedad por sus ventajas, entre las que se puede mencionar la oportunidad de seguir un paciente a lo largo de todo el sistema de salud, independientemente del nivel de atención en el que haya sido tratado, aunque siempre hay que tener en cuenta los aspectos éticos y legales.(García., 2006)

Desarrollo de la Cirugía Endoscópica en Cuba:

La primera Laparoscopia realizada en Cuba fue en el año 1932, pero no es hasta 1945 que se introduce la Laparoscopia para el diagnóstico de las afecciones hepatobiliares.

En 1956 se extendieron las indicaciones de la Laparoscopia al abdomen agudo y se creó la escuela cubana de Endoscopia Digestiva.

La primera colecistectomía laparoscópica se realiza en la provincia de Sancti Spíritus por los doctores Alfredo F. Rodríguez y Jorge García Tamarit, en febrero de 1991 y en ese mismo año se hizo la primera colecistectomía videolaparoscópica en el Hospital Hermanos Ameijeiras por un grupo de especialistas dirigidos por el Dr. José Díaz Calderón.

En Marzo de 1993 se creó en el Hospital Calixto García un grupo multidisciplinario formado por gastroenterólogos, cirujanos, anestesiólogos y radiólogos para enfrentar de forma integral la cirugía endoscópica, la endoscopia terapéutica y la radiología intervencionista; grupo que se desarrolló dando lugar a la creación en 1994, del hoy Centro de Referencia Nacional para la Cirugía Endoscópica("Desarrollo de la Cirugía Endoscópica en Cuba," 2007).

El uso de la modalidad de Colecistectomía Laparoscópica para el tratamiento de las afecciones de la vesícula biliar redunda en beneficios tanto para el paciente como desde el punto de vista hospitalario debido a los beneficios derivados de realizarse mediante mínimo acceso reduciendo sensiblemente los riesgos del acceso a la cavidad abdominal así como la estadía hospitalaria. Específicamente en la provincia de Sancti Spíritus se aplica desde el año 1998 y en la actualidad existe 1 grupo de cirujanos realizando dichas intervenciones en una media de 80 pacientes mensualmente.

La recolección y tratamiento de la información referente a los pacientes remitidos a la consulta de Colecistectomía Laparoscópica en la provincia se realiza manualmente empleando un formato impreso que el cirujano va rellenando a lo largo del proceso desde la recepción del remitido en consulta hasta el alta hospitalaria con los datos específicos de cada paciente.

Este tipo de manejo implica una serie de debilidades desde el punto de vista del tratamiento de la información, entre las que se encuentran:

- Falta de confiabilidad, calidad y consistencia de la información.
- Deterioro del soporte documental con la consecuente perdida de los datos
- Plazos de tiempo excesivos dedicados a la localización de información
- Imposibilidad de acceder a datos estadísticos e históricos
- Gasto de papel e impresión implicados

En la actualidad se encuentran volcados a este formato los seguimientos de aproximadamente 2000 pacientes pertenecientes a la provincia de Sancti Spíritus.

En Cuba hasta el momento solamente se ha trabajado el campo de las herramientas de software orientadas a la rama en el Centro Nacional de Cirugía Endoscópica ubicado en Ciudad de la Habana donde en el año 1993 se desarrolló utilizando Visual Basic y Microsoft Access una aplicación denominado ColeSoft. Dicha aplicación desarrollada con software propietario no se llegó a generalizar aunque mediante su uso en dicho centro permitió introducir datos de alrededor de 8000 pacientes hasta el 2008.(Nilo, 2010).

Durante la investigación realizada no se encontraron referencias a herramientas de este tipo en otros países del área.

Toda esta información ha dado lugar al siguiente **problema científico:** ¿Cómo mejorar la gestión de la información referente a los pacientes intervenidos mediante Colecistectomía Laparoscópica en el Hospital Provincial Camilo Cienfuegos de Sancti Spíritus?

Dicho problema conlleva al **objetivo general** de la investigación: Diseñar e implementar una aplicación informática que gestione la información del seguimiento a paciente de la especialidad de Colecistectomía Laparoscópica y cumpla con los estándares de código abierto apoyándose en la utilización de marcos de trabajo.

Los **objetivos específicos** son:

1.- Realizar un estudio sobre la utilización de marcos de trabajo de Java para el desarrollo de aplicaciones de escritorio dedicadas a la gestión de información.

2.- Definir los procesos de negocio fundamentales para la gestión de información de seguimiento a pacientes de Colecistectomía Laparoscópica.

3.- Definir la arquitectura de la aplicación.

4.- Implementar la aplicación mediante el lenguaje de programación Java utilizando los marcos de trabajo seleccionados.

Se pretende que este trabajo sea referencia para futuras consultas en el sentido:

Práctico: está dado por la implementación de una herramienta informática que mejorará sensiblemente un flujo de información hospitalaria deficiente en estos momentos dentro del marco de la especialidad de Colecistectomía Laparoscópica.

Metodológico: por instruir desde el punto de vista teórico sobre las acciones a realizar para realizar con éxito la unión y configuración de los marcos de trabajo seleccionados, que permitan llevar a buen término el desarrollo de una aplicación de gestión de información basada en software libre.

El trabajo de tesis está estructurado en cuatro capítulos, distribuidos de la siguiente forma:

Capítulo 1: Aplicaciones Desktop con Java. Marcos de trabajo.

Capítulo 2: Análisis y diseño del proceso de gestión de información acerca de intervenciones de Colecistectomía laparoscópica.

Capítulo 3: Implementación del sistema.

Capítulo 4: Laparogest: Sistema de gestión de información acerca de intervenciones de Colecistectomía Laparoscópica.

En el primer capítulo se hace la revisión bibliográfica, especificando el uso de los marcos de trabajo utilizados en la aplicación, las razones de la selección así como sus ventajas y características generales.

El segundo capítulo describe el problema real, se procede al análisis y diseño del sistema desde el punto de vista computacional, con el uso de los diagramas propios de cada etapa del diseño de la ingeniería de software, así como la estructura de la base de datos que sirve de soporte a la gestión de la información de la aplicación.

En el tercer capítulo se detallan técnicamente las herramientas usadas y las interioridades de la programación con los marcos de trabajo en Java que posibilitaron realizar la implementación de este sistema informático.

En el último capítulo se presenta la interfaz de usuario del sistema.

Finalmente se ofrecen las conclusiones, recomendaciones y la bibliografía utilizada, así como un conjunto de anexos que permiten apoyar los temas desarrollados en el contenido del documento.

CAPITULO 1 APLICACIONES DESKTOP CON JAVA. MARCOS DE TRABAJO.

El entorno o plataforma de computación Java originaria de *Sun Microsystems*, capaz de ejecutar aplicaciones desarrolladas usando el Lenguaje de programación Java u otros lenguajes que compilen a código intermedio y un conjunto de herramientas de desarrollo. En este caso, la plataforma no es un hardware específico o un sistema operativo, sino más bien una máquina virtual encargada de la ejecución de aplicaciones, y un conjunto de bibliotecas estándares que ofrecen funcionalidad común(Zukowski, 2006).

La plataforma así llamada incluye:

- Edición Estándar (*Java Platform, Standard Edition*), o Java SE (antes J2SE)
- Edición Empresarial (*Java Platform, Enterprise Edition*), o Java EE (antes J2EE)
- Edición Micro (*Java Platform, Micro Edition*), o Java ME (antes J2ME)

Desde 2006, la versión actual de la Plataforma *Java Standard Edition* se conoce como Java SE 6 como versión externa, y 1.6 como versión interna. Sin embargo, se prefiere el término versión 6. (Pérez, 2009)

La Plataforma Java se compone de un amplio abanico de tecnologías, cada una de las cuales ofrece una parte del complejo de desarrollo o del entorno de ejecución en tiempo real. Por ejemplo, los usuarios finales suelen interactuar con la máquina virtual de Java y el conjunto estándar de bibliotecas. Además, las aplicaciones Java pueden usarse de forma variada, como por ejemplo ser incrustadas en una página Web. Para el desarrollo de aplicaciones, se utiliza un conjunto de herramientas conocidas como JDK (Java Development Kit, o herramientas de desarrollo para Java). (Froufe, 1997)

1.1 Java Runtime Environment

Un programa destinado a la Plataforma Java necesita dos componentes en el sistema donde se va a ejecutar: una máquina virtual de Java (*Java Virtual Machine, JVM*), y un conjunto de bibliotecas para proporcionar los servicios que pueda necesitar la aplicación. La JVM que proporciona Sun Microsystems, junto con su implementación de las bibliotecas estándares, se conocen como *Java Runtime Environment* (JRE) o Entorno en tiempo de ejecución para Java. El JRE es lo mínimo que debe contener un sistema para poder ejecutar una aplicación Java sobre el mismo.

En el concepto de máquina virtual se encierra el concepto común de un procesador “virtual” que ejecuta programas escritos en el lenguaje de programación Java. En

concreto, ejecuta el código resultante de la compilación del código fuente, conocido como bytecode. Este “procesador” es la máquina virtual de Java o JVM, que se encarga de traducir (interpretar o compilar al vuelo) el bytecode en instrucciones nativas de la plataforma destino. Esto permite que una misma aplicación Java pueda ser ejecutada en una gran variedad de sistemas con arquitecturas distintas, siempre que con una implementación adecuada de la JVM. Este hecho es lo que ha dado lugar a la famosa frase: “*write once, run anywhere”* (escriba una vez, ejecute en cualquier parte). La condición es que no se utilicen llamadas nativas o funciones especificas de una plataforma y aún así no se asegura completamente que se cumpla una verdadera independencia de plataforma. Desde la versión 1.2 de JRE, la implementación de la máquina virtual de Sun incluye un compilador JIT (Just In Time). De esta forma, en vez de la tradicional interpretación del código bytecode, que da lugar a una ejecución lenta de las aplicaciones, el JIT convierte el bytecode a código nativo de la plataforma destino. Esta segunda compilación del código penaliza en cuanto a tiempo, pero el código nativo resultante se ejecuta de forma más eficaz y rápida que si fuera interpretado. Otras técnicas de compilación dinámica del código durante el tiempo de ejecución permiten optimizar más aún el código, dejando atrás el estigma que caía sobre Java en cuanto a su lentitud y en sus últimas versiones la JVM se ha optimizado a tal punto que ya no se considera una plataforma lenta en cuanto a ejecución de aplicaciones.

Java no fue la primera plataforma basada en el concepto de una máquina virtual, aunque es la que de más amplia difusión ha gozado. El empleo de máquinas virtuales se había centrado principalmente en el uso de emuladores para ayudar al desarrollo de hardware en construcción o sistemas operativos, pero la JVM fue diseñada para ser implementada completamente en software, y al mismo tiempo hacer que fuera portable a todo tipo de hardware.(Campo, 2001)

1.2 Lenguaje de programación Java

La palabra Java, por sí misma, se refiere habitualmente al lenguaje de programación Java, que fue diseñado para usar con la Plataforma Java. Los lenguajes de programación se encuentran fuera del ámbito de lo que es una “plataforma”, aunque el lenguaje de programación Java es uno de los componentes fundamentales de la propia plataforma. El propio lenguaje y el entorno en tiempo de ejecución suelen considerarse una única entidad.

De acuerdo con el libro blanco de Java ("Original Java WhitePaper,"), 10s objetivos del diseño de Java eran ser "un lenguaje sencillo, orientado a objetos, distribuido,

interpretado, robusto, seguro, de arquitectura neutral, portátil, de gran rendimiento, multitarea y dinámico".

El cumplimiento de dichos objetivos por parte de sus desarrolladores en los laboratorios de Sun (actualmente Oracle) así como los aportes de la comunidad han conllevado a que en la actualidad Java sea un lenguaje que integra entre otras las siguientes características:

- Basado en C++ pero simplificado, mucho más fácil de usar, de más alto nivel. y
- Menos propenso a errores.
- Amplísima biblioteca estándar de clases predefinidas.
- Multiplataforma: Las aplicaciones Java pueden ser ejecutadas indistintamente en cualquier plataforma sin necesidad de recompilación.
- Amplio espectro: programación tradicional, distribuida, GUI, Web, dispositivos móviles, etc.
- Gestión avanzada de memoria mediante el uso de un recolector de basura.
- Gestión avanzada de errores, tanto en tiempo de compilación como de ejecución.
- Soporte sencillo de múltiples hebras de ejecución.
- Lenguaje abierto. Kits de desarrollo y documentación gratuitos en la red.

1.3 Bibliotecas de Java

En la mayoría de los sistemas operativos actuales, se ofrece una cantidad de código para simplificar la tarea de programación. Este código toma la forma, normalmente, de un conjunto de bibliotecas dinámicas que las aplicaciones pueden llamar cuando lo necesiten. Pero la plataforma Java está pensada para ser independiente del sistema operativo subyacente, por lo que las aplicaciones no pueden apoyarse en funciones dependientes de cada sistema en concreto. Lo que hace la plataforma Java, es ofrecer un conjunto de bibliotecas estándares, que contiene mucha de las funciones reutilizables disponibles en los sistemas operativos actuales.

Las bibliotecas de Java tienen tres propósitos dentro de la plataforma Java. Al igual que otras bibliotecas estándares, ofrecen al programador un conjunto bien definido de funciones para realizar tareas comunes, como manejar listas de elementos u operar de forma sofisticada sobre cadenas de caracteres. Además, las bibliotecas proporcionan una interfaz abstracta para tareas que son altamente dependientes del hardware de la plataforma destino y de su sistema operativo. Tareas tales como manejo de las funciones de red o acceso a ficheros, suelen depender fuertemente de la funcionalidad nativa de la plataforma destino. En el caso concreto anterior, las bibliotecas java.net y java.io

implementan el código nativo internamente, y ofrecen una interfaz estándar para que aplicaciones Java puedan ejecutar tales funciones. Finalmente, no todas las plataformas soportan todas las funciones que una aplicación Java espera. En estos casos, las bibliotecas bien pueden emular esas funciones usando lo que esté disponible, o bien ofrecer un mecanismo para comprobar si una funcionalidad concreta está presente.

1.4 Aplicaciones en capas

La estrategia tradicional de utilizar aplicaciones compactas causa gran cantidad de problemas de integración en sistemas de aplicaciones complejos como pueden ser los sistemas de gestión de una empresa o los sistemas de información integrados consistentes en más de una aplicación. Estas aplicaciones suelen encontrarse con importantes problemas de escalabilidad, disponibilidad, seguridad e integración. Para solventar estos problemas se ha generalizado la división de las aplicaciones en capas que normalmente serán tres: una capa que servirá para guardar los datos (modelo), una capa para centralizar la lógica de negocio (control) y por último una interfaz gráfica que facilite al usuario el uso del sistema (presentación). (Albin, 2003)

Si se establece una separación entre la capa de interfaz gráfica (cliente), replicada en cada uno de los entornos de usuario, y la capa del modelo, que quedaría centralizada en un servidor de base de datos se obtiene una potente arquitectura que otorga algunas ventajas:

- Centralización de los aspectos de seguridad y transaccionalidad, que serían responsabilidad del modelo.
- No replicación de la lógica de negocio en los clientes, lo que permite que las modificaciones y mejoras sean automáticamente aprovechadas por el conjunto de los usuarios, reduciendo los costos de mantenimiento.
- Mayor sencillez de los clientes.

La mayoría de las aplicaciones de escritorio comunes utilizan una **arquitectura basada en la de tres capas** extendida a sus particularidades.

1.5 Marcos de trabajo en Java

El término “marco de trabajo” o la mas utilizada palabra inglesa “framework” define, en términos generales, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular, que sirve como referencia para enfrentar y resolver nuevos problemas de índole similar.

En el desarrollo de software, un framework es una estructura conceptual y tecnológica 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. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.("Frameworks," 2007) Fuera de las aplicaciones en la informática, puede ser considerado como el conjunto de procesos y tecnologías usados para resolver un problema complejo. Es el esqueleto sobre el cual varios objetos son integrados para una solución dada.

Lo novedoso de las tecnologías usadas es uno de los puntos fuertes de este proyecto. Todas ellas son tecnologías Java de código abierto. El aprendizaje y familiarización ha ocupado la mayor parte del tiempo del proyecto. A continuación se describen algunos marcos de trabajo disponibles para la implementación de las diferentes capas.

1.5.1 Capa de datos

Los datos son parte integral de una aplicación, incluso las más puristas del modelo orientado a objeto. Todo el proceso viene a parar a la capa de datos donde tradicionalmente los desarrolladores Java han tenido que escribir consultas en lenguaje SQL las cuales van ganando en complejidad a medida que la aplicación crece llegando a convertirse en ocasiones en un serio problema de mantenibilidad de código.

Enfoque clásico basado en SQL

JDBC (Java Database Conectivity) es un API de Java que permite al programador ejecutar instrucciones en lenguaje estándar de acceso a Bases de Datos, **SQL** (*Structured Query Language*, lenguaje estructurado de consultas), que es un lenguaje de muy alto nivel que permite crear, examinar, manipular y gestionar Bases de Datos relacionales. Para que una aplicación pueda hacer operaciones en una Base de Datos, ha de tener una conexión con ella, que se establece a través de un *driver*, que convierte el lenguaje de alto nivel a sentencias de Base de Datos. Es decir, las tres acciones principales que realizará JDBC son las de establecer la conexión a una base de datos, ya sea remota o no; enviar sentencias SQL a esa base de datos y, en tercer lugar, procesar los resultados obtenidos de la base de datos.

Enfoque basado en ORM

El enfoque ideal seria que pudiera tratarse las operaciones de datos como objetos y aplicarle conceptos y técnicas provenientes de la POO tales como abstracción, herencia, encapsulación, polimorfismo logrando mediante esto esconder la complejidad de la capa

de datos subyacente. La comunidad Java ha generado diferentes enfoques orientados a objeto de la persistencia de datos

Este es le caso de los marcos de trabajo basados en el mapeo objeto-relacional, del inglés ORM (*Object-Relational Mapping*) que es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional. En la práctica esto crea una base de datos orientada a objetos virtual sobre la base de datos relacional. (Rodríguez, 2010)

Utilizar un framework ORM simplifica enormemente la programación de la lógica de persistencia. Se trata de una idea completamente madura que cada vez se vuelve más popular. En aplicaciones donde la lógica de negocios trabaja contra un modelo de dominio completamente orientado a objetos la generación de código se reduce entre un 30% y un 40%. Además el código generado suele ser mucho más sencillo y mantenible.

TopLink

Es un marco de trabajo ORM que almacena objetos Java en una base de datos relacional o convierte objetos Java a documentos XML. Está liberado con la licencia *Oracle Licence* y dentro de sus características fundamentales están:

Consultas que soportan *Query by Example* (QBE), EJB QL, SQL, y procedimientos almacenados,

Transacciones a nivel de objeto,

Caché avanzado que asegura la identidad de los objetos,

Mapeo de objetos a XML,

Soporte para fuentes de datos no relacionales

Editor visual.

iBATIS

Está basado en capas y es desarrollado por *Apache Software Foundation*. Se ocupa de la capa de persistencia y se sitúa entre la lógica de negocio y la capa de la base de datos.

Puede ser implementado en Java y .NET y también existe una versión para *Ruby on Rails*. *iBATIS* asocia objetos de modelo (*JavaBeans*) con sentencias SQL o procedimientos almacenados mediante ficheros descriptores XML, simplificando la utilización de bases de datos. La capa de persistencia se configura mediante un fichero XML de configuración, sql-map-config.xml. Además cada objeto de modelo, que representa al objeto en la aplicación, se relaciona con un fichero del tipo sqlMap.xml, que contiene sus sentencias SQL.

Hibernate

Hibernate es un potente mapeador objeto/relacional y servicio de consultas para Java. Es la solución ORM más popular en el mundo Java. Permite desarrollar clases persistentes a partir de clases comunes, incluyendo asociación, herencia, polimorfismo, composición y colecciones de objetos. El lenguaje de consultas de HQL (*Hibernate Query Language*), diseñado como una mínima extensión orientada a objetos de SQL, proporciona un puente elegante entre los mundos objetual y relacional. *Hibernate* también permite expresar consultas utilizando SQL nativo o consultas basadas en criterios. Soporta todos los sistemas gestores de bases de datos SQL y se integra de manera elegante y sin restricciones con los más populares servidores de aplicaciones J2EE y contenedores web, y por supuesto también puede utilizarse en aplicaciones *standalone*. Posee un alto rendimiento, tiene una caché de dos niveles y puede ser usado en un clúster. Permite inicialización perezosa (*lazy*) de objetos y colecciones y soporta la generación automática de llaves primarias(Bauer, 2005) (Bauer, 2006).

JPA

La Java Persistente API (JPA) es el API estándar de persistencia introducido como parte de la Java EE 5 Plataform con el objetivo de conformar una única API de persistencia para la Comunidad Java, que recoja lo mejor de las anteriores.

Tiene entre sus principales características

- Puede hacer uso de diferentes ORM's establecidos en el mercado tales como TopLink e Hibernate
- No precisa de contenedor. Funciona tanto en J2EE como J2SE
- Modelo de persistencia no intrusivo al estilo POJO
- Metadata con Anotaciones: Se eliminan los deployment descriptors, se reduce drásticamente el número de clases a crear (o generar)
- Reducir al máximo la complejidad
- Puede usarse independientemente del resto de los servicios (transacciones, seguridad,..)
- Configuraciones por defecto: reducen la cantidad de configuración a especificar
- No Intrusión: los objetos a persistir (Entity Beans) no necesitan implementar interfaces EJB
- Herencia y poliformismo
- Lenguaje de consulta (EJBQL) mejorado: inner and outer join,operaciones bulk, sql nativo.

Ventajas

- Simplicidad: una única clase para declarar la persistencia (con la ayuda de anotaciones)
- Facilidad de aprendizaje
- Transparencia: las clases a persistir son simples POJOs
- No hay restricciones con respecto a relaciones entre objetos (herencia, poliformismo)

Desventajas

- Las anotaciones suponen una descripción de la Base de Datos en el código
  - Solución: Siempre se puede utilizar un XML

1.5.2 Capa de Presentación

AWT (Abstract Windows Toolkit)

Desde sus inicios el entorno Java ya contaba con una biblioteca de componentes gráficos conocida como AWT. Esta biblioteca estaba concebida como una API estandarizada que permitía utilizar los componentes nativos de cada sistema operativo. Entonces una aplicación Java corriendo en Microsoft Windows usaría el botón estándar de Windows y una aplicación corriendo en UNIX usaría el botón estándar de Motif. En la práctica esta tecnología no funcionó:

- Al depender fuertemente de los componentes nativos del sistema operativo el programador AWT estaba confinado a un mínimo denominador común entre ellos. Es decir que sólo se disponen en AWT de las funcionalidades comunes en todos los sistemas operativos.
- El comportamiento de los controles varía mucho de sistema a sistema y se vuelve muy difícil construir aplicaciones portables.

Swing

El paquete de componentes de Swing fue originalmente creado debido a que los componentes básicos de AWT incluidos en la versión original de las librerías de Java eran insuficientes para la creciente complejidad de las aplicaciones del mundo real. Swing está basado en una arquitectura MVC modificada (Fowler, 2008). Sigue un simple modelo de programación por hilos, y posee las siguientes características principales:

- Independencia de plataforma.

- Extensibilidad: es una arquitectura altamente particionada: los usuarios pueden proveer sus propias implementaciones modificadas para sobrescribir las implementaciones por defecto. Se puede extender clases existentes proveyendo alternativas de implementación para elementos esenciales.
- Personalizable: dado el modelo de representación programático del framework de Swing, el control permite representar diferentes estilos de apariencia "look and feel" (desde apariencia MacOS hasta apariencia Windows XP pasando por apariencia GTK+, IBM UNIX o HP UX entre otros).

La arquitectura Swing presenta una serie de ventajas respecto a su antecedente AWT:

- El diseño en Java puro posee menos limitaciones de plataforma.
- El desarrollo de componentes Swing es más activo.
- Amplia variedad de componentes: En general las clases que comiencen por "J" son componentes que se pueden añadir a la aplicación. Por ejemplo: JButton.
- Aspecto modificable (look and feel): Se puede personalizar el aspecto de las interfaces o utilizar varios aspectos que existen por defecto (Metal Max, Basic Motif, Windows Win32).
- Arquitectura Modelo-Vista-Controlador: Esta arquitectura da lugar a todo un enfoque de desarrollo muy arraigado en los entornos gráficos de usuario realizados con técnicas orientadas a objetos. Cada componente tiene asociado una clase de modelo de datos y una interfaz que utiliza. Se puede crear un modelo de datos personalizado para cada componente, con sólo heredar de la clase Model.
- Gestión mejorada de la entrada del usuario: Se pueden gestionar combinaciones de teclas en un objeto KeyStroke y registrarlo como componente. El evento se activará cuando se pulse dicha combinación si está siendo utilizado el componente, la ventana en que se encuentra o algún hijo del componente.
- Objetos de acción (action objects): Estos objetos cuando están activados (enabled) controlan las acciones de varios objetos componentes de la interfaz. Son hijos de ActionListener.
- Contenedores anidados: Cualquier componente puede estar anidado en otro. Por ejemplo, un gráfico se puede anidar en una lista.

- Escritorios virtuales: Se pueden crear escritorios virtuales o "interfaz de múltiples documentos" mediante las clases JDesktopPane y JInternalFrame.
- Bordes complejos: Los componentes pueden presentar nuevos tipos de bordes. Además el usuario puede crear tipos de bordes personalizados.
- Diálogos personalizados: Se pueden crear multitud de formas de mensajes y opciones de diálogo con el usuario, mediante la clase JOptionPane.
- Clases para diálogos habituales: Se puede utilizar JFileChooser para elegir un fichero, y JColorChooser para elegir un color.
- Componentes para tablas y árboles de datos: Mediante las clases JTable y JTree.
- Potentes manipuladores de texto: Además de campos y áreas de texto, se presentan campos de sintaxis oculta JPassword, y texto con múltiples fuentes JTextPane. Además hay paquetes para utilizar ficheros en formato HTML o RTF.
- Capacidad para "deshacer": En gran variedad de situaciones se pueden deshacer las modificaciones que se realizaron.
- Soporte a la accesibilidad: Se facilita la generación de interfaces que ayuden a la accesibilidad de discapacitados, por ejemplo en Braille.

Standard Widget Toolkit (SWT)

Standard Widget Toolkit (SWT) es un paquete de componentes gráficos para ser usados en la plataforma Java. Fue originalmente desarrollado por IBM y actualmente lo hace la Eclipse Foundation en paralelo con el IDE Eclipse. Provee una alternativa para paquetes como AWT y Swing desarrollados por Sun Microsystems como parte de la Java Plataform, Standard Edition. SWT esta escrito en Java.

Para mostrar elementos de interfaz grafica SWT accede a las librerías nativas del sistema operativo usando JNI (Java Native Interface) de una manera similar a la usada por los programas escritos usando APIs especificas del SO. Las aplicaciones que usan SWT son portables, pero la implementación del toolkit, a pesar del hecho de que esta escrito en Java, es única para cada plataforma.

Beans Binding

Beans Binding o JSR 295 es una API orientada al uso con las librerías SWING en aplicaciones de escritorio y desarrollada con el propósito específico de mantener sincronizadas dos propiedades de Java Beans. Permite sincronizar de manera declarativa mediante el uso de Expression Language componentes de la capa de presentación con objetos del modelo de datos de la aplicación simplificando el proceso de escribir listeners para cada uno de ellos. Admite validación de datos y conversión entre tipo en el proceso de manejo de datos.

1.5.3 Generadores de reportes

BIRT Report

BIRT (Business Intelligence and Reporting Tools) es un proyecto de software de código abierto que provee las funcionalidades de informes e inteligencia de negocios para aplicaciones Web, especialmente aquellas basadas en Java y J2EE. BIRT es uno de los proyectos principales de la Eclipse Foundation. Las metas del proyecto son las de abarcar una amplia gama de necesidades de informes que incluyan desde los típicos informes de una aplicación empresarial de gestión de datos hasta un procesamiento analítico multidimensional en línea (OLAP). El proyecto cuenta con una amplia comunidad de usuarios y desarrolladores.

BIRT tiene dos componentes fundamentales: un diseñador visual de informes incluido en el IDE Eclipse y un componente en tiempo de ejecución para generar informes que pueden ser ejecutados en cualquier ambiente Java.

Jasper Reports

Jasper Report es una librería Open Source de Java diseñada proveer capacidad de generación de reportes a aplicaciones Java. Está escrito completamente en Java y puede ser usado en gran variedad de aplicaciones, tanto web como de escritorio e incluso de línea de comandos, para generar contenido dinámico. Su propósito principal es ayudar a crear documentos de tipo páginas, preparados para imprimir en una forma simple y flexible. Puede exportar el contenido a la impresora o a ficheros PDF, HTML, XLS, CSV y XML.

JasperReports se usa comúnmente con iReport, una interfaz visual de código abierto para la edición de informes y se presenta en forma de plugin que permite completa integración con el IDE Netbeans(Heffelfinger, 2006).

¿Cuál es la mejor combinación? Para tomar una decisión de tanta trascendencia en la vida de un proyecto de software se ha tomado en cuenta tanto factores obtenidos durante la especificación de requisitos como en el estudio realizado, entre ellos podemos nombrar:

De la especificación de requisitos:

- Necesidades del proyecto
- Tamaño del proyecto
- Necesidades de la organización
- Experiencia de los desarrolladores

De los frameworks estudiados:

- Madurez de la tecnología
- Documentación y Soporte
- Curva de aprendizaje
- Facilidad de Debug
- Rendimiento
- Facilidad de Uso
- Integración con otras herramientas
- Testeabilidad
- Integración
- Intrusión
- Reusabilidad
- Soporte para Transacciones
- Escalabilidad
- Facilidad de Refactorización
- Seguridad
- Persistencia transitiva (estilo de cascada)
- Herramientas de apoyo

Al finalizar la consulta de información y basado en la evaluación de los factores a los marcos de trabajo estudiados se decide implementar la siguiente arquitectura de marcos de trabajo:
Swing para la capa de presentación, **JPA** para la capa de modelo del negocio y persistencia de datos, **Beans Binding** para la capa de enlace y **Jasper Report** como generador de reportes .

CAPÍTULO 2: MODELADO Y ANÁLÍSIS DEL PROCESO DE GESTIÓN DE INFORMACIÓN (SEGUIMIENTO) EN INTERVENCIONES DE COLECISTECTOMÍA LAPAROSCÓPICA.

2.1 Algunos términos del negocio

Para la cabal explicación de los procesos a describir será necesario esclarecer el uso de algunos términos que forman parte del negocio, algunos de las palabras aquí tratadas tienen diferentes acepciones pero solo serán mencionadas las que son significativas y están directamente relacionadas con el problema de investigación que nos ocupa.

- Paciente: Persona que padece física y corporalmente, y especialmente quien se halla bajo atención médica. || Persona que es o va a ser reconocida médicamente.
- Vesícula biliar: Órgano muscular que almacena la bilis, presente en la mayoría de los vertebrados. En el ser humano es un saco membranoso con forma de pera situado bajo la superficie del lóbulo derecho del hígado, justo detrás de las costillas inferiores.
- Colecistectomía: Resección quirúrgica de la vesícula biliar. || Intervención quirúrgica que tiene como objetivo extirpar la vesícula biliar de un paciente.
- Laparoscopía: Técnica diagnóstica y terapéutica basada en sistemas de visión y manipulación especiales introducidos en la cavidad abdominal a través de incisiones puntiformes.
- Laparotomía (apertura quirúrgica de la pared abdominal).
- Colecistectomía Laparoscópica: Intervención de colecistectomía realizada mediante el uso de la técnica de mínimo acceso, posibilita reducir sensiblemente los riesgos así como el tiempo de hospitalización necesario.
- Seguimiento: Proceso de recolección de datos que permite al especialista vinculado mantener control detallado de la evolución de un paciente.
- Preoperatorio: Primera etapa del proceso de seguimiento, comprende desde la remisión de dicho paciente a la consulta de colecistectomía laparoscópica hasta la ejecución del acto quirúrgico.
- Transoperatorio: Segunda etapa del proceso de seguimiento, comprende toda la información relacionada con el acto quirúrgico en si.
- Postoperatorio: Tercera etapa del proceso de seguimiento, comprende la información generada desde la conclusión del acto quirúrgico hasta el alta hospitalaria del paciente.

2.2 Proceso AS-IS

A la consulta de Colecistectomía Laparoscópica llegan pacientes remitidos de diferentes especialidades pero el peso mayor lo lleva Gastroenterología por la relación entre los trastornos digestivos y las patologías relacionadas con la vesícula biliar. Al momento de recibirse un remitido el especialista (Cirujano) que lleva el caso toma los datos personales del paciente, consulta su historia clínica y determina la pertinencia de su remisión.

Una vez aceptado el paciente inicia el proceso de seguimiento (**etapa preoperatoria**) mediante la recolección de los datos clínicos básicos y la orientación de una serie de exámenes complementarios, en el transcurso dicha etapa se almacenan en un impreso datos referentes a:

- Cuadro Clínico
- Tiempo evolución
- Peso
- Talla
- Antecedentes patológicos
- Factores de Riesgo
- Intervenciones quirúrgicas anteriores
- Resultado del examen de FAS
- Resultado del examen de TGP
- Resultado del examen de EritroSedimentación
- Resultado del examen de Bilirrubina Total
- Resultado del examen de Bilirrubina Directa

Basado en los resultados del seguimiento en esta etapa el especialista puede llegar a un diagnóstico presuntivo del caso, basado en este precisar la pertinencia y consiguiente fecha de ejecución del acto quirúrgico o posponer o descartar este en caso de no considerarlo necesario.

En caso de precisar la intervención se lleva el seguimiento a la siguiente etapa (**etapa transoperatoria**) la cual se centra en la ejecución del acto quirúrgico, luego de su culminación el especialista vuelca al impreso información referente a este:

- Cirujano que realiza
- Fecha de ejecución
- Tiempo quirúrgico
- Tipo de operación
- Tipo de Colecistectomía realizada

- Tipo de Laparascopía realizada
- Tipo de extracción de vesícula
- Uso de drenaje
- Uso de antibiótico
- Otras intervenciones realizadas
- Accidentes propios de la técnica
- Hallazgos quirúrgicos

La culminación del acto permite llegar a un diagnóstico quirúrgico que implica un mayor nivel de exactitud en la determinación de la patología que aqueja al paciente.
Luego de la salida del paciente del quirófano se procede a iniciar una nueva etapa del seguimiento (**etapa postoperatoria**) en la cual se pasan al impreso los datos correspondientes al lapso que transcurre desde dicho momento hasta la emisión del alta de la consulta, en el conjunto de dichos datos se encuentran

- Cuantía de la estadía hospitalaria
- Complicaciones postoperatorias
- Diagnostico Anamopatológico

En dicha etapa se reciben desde el departamento de Anatomía Patológica los resultados del diagnóstico anamopatológico.

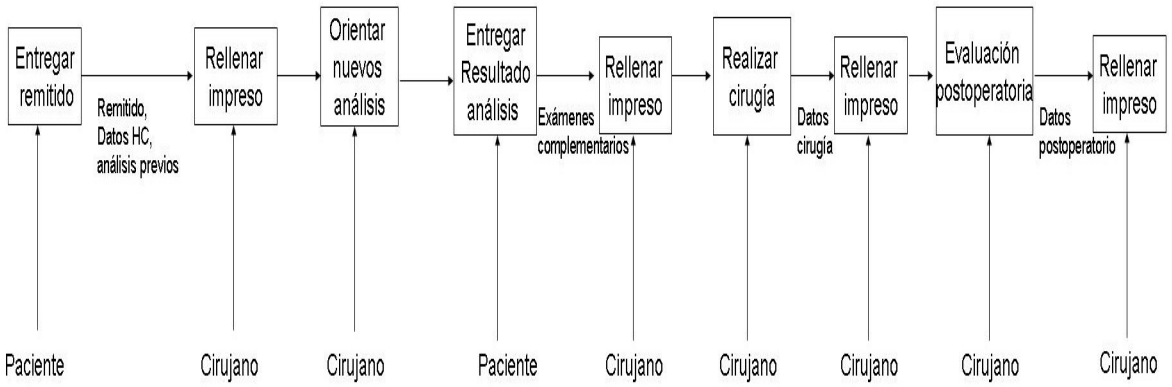

Fig. 1 Modelado procesos del negocio AS IS notación IDEF

El modelado del negocio permite obtener una serie de artefactos correspondientes a esta etapa, debido a su trascendencia aquí se muestran dos de ellos

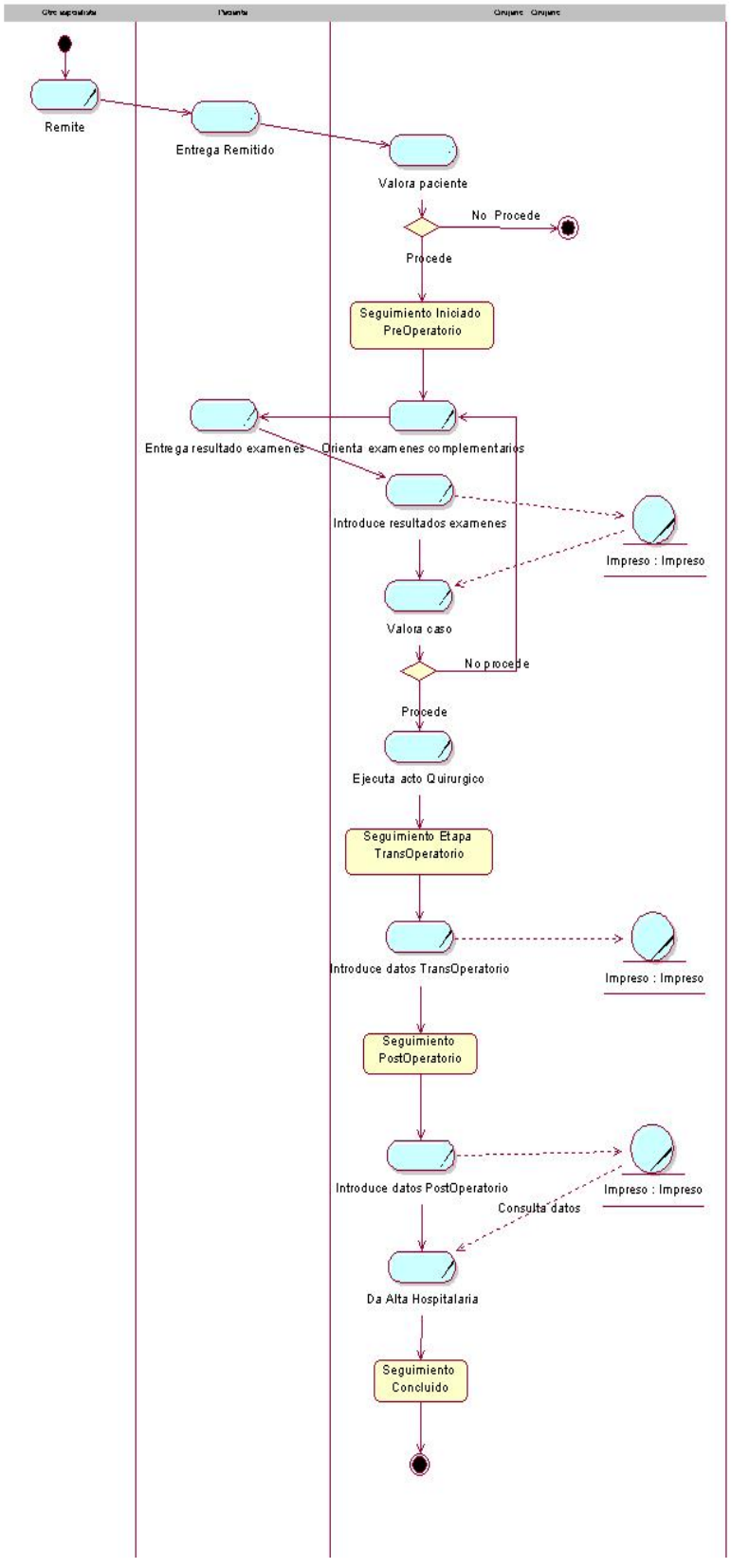

Fig. 2 Diagrama de actividades del negocio

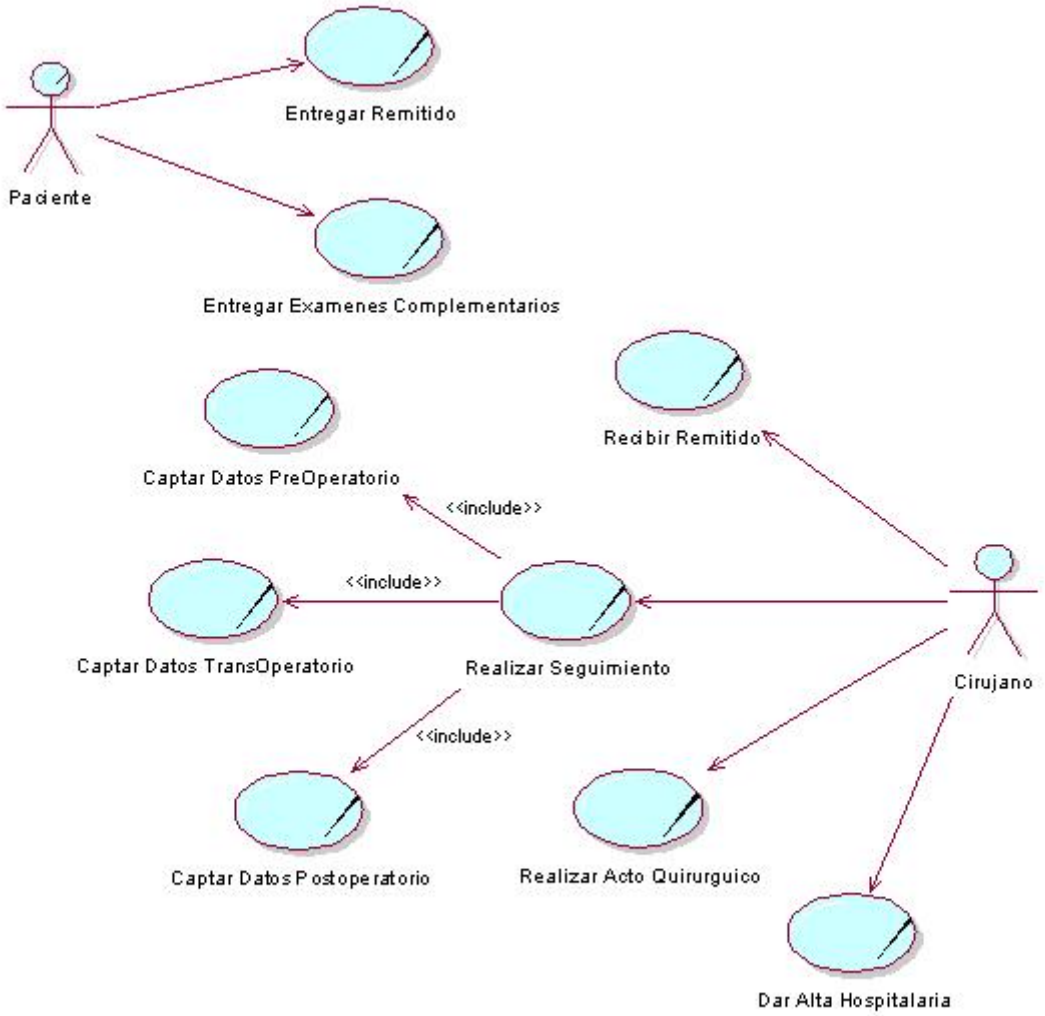

Fig. 3 Modelo de casos de uso del negocio

Conclusiones de la etapa de modelado del negocio:

El estado actual de los procesos del negocio se traduce en un seguimiento completamente manual, por ende lento y trabajoso con gasto de papel y tiempo invertido, con escasas o nulas posibilidades de confrontación de información, debido a dichas debilidades del proceso del negocio se desperdicia un gran potencial de datos susceptible a ser usado en la toma de decisiones o en procesos de aprendizaje.

2.3 Requisitos funcionales detectados

A continuación se describen los requisitos funcionales detectados en el estudio del negocio, el resto de los requisitos se describen en el anexo 1

RF 1 Gestión de usuario y seguridad.

RF 1.1 Autenticar Usuarios.

Este requisito es el responsable de autenticar los usuarios que entren al sistema, dicha autenticación se realizará mediante una combinación de nombre-contraseña garantizando la seguridad y confidencialidad de este.

RF 1.2 Crear Usuarios.

Este requisito es el responsable de crear un nuevo usuario del sistema insertando todos los datos del mismo.

- Nombre de usuario
- Contraseña
- Descripción

RF 1.3 Modificar Usuarios.

Este requisito se encarga de actualizar algún dato que se necesite de los usuarios existentes.

RF 1.4 Eliminar Usuarios.

Este requisito elimina a un usuario existente y todos sus datos del sistema.

RF 1.5 Cambiar Contraseña.

Este requisito permite que un usuario registrado pueda desde su sesión cambiar su contraseña de acceso.

RF 2 Gestionar datos de Nomencladores.

Este requisito se encarga de garantizar la gestión de los Nomencladores del sistema y se desglosa en 12 requisitos independientes.

RF 2.1 Gestionar datos Nomenclador de Accidentes Propios de la Técnica.

Este requisito es el responsable de Gestionar todos los datos referidos a los accidentes propios de la técnica de Laparoscopía.

En el mismo se debe incluir:

- Descripción.

RF 2.2 Gestionar datos Nomenclador de Causas de Conversión.

Este requisito es el responsable de registrar todos los datos referidos a las causas de conversión entre modalidades de Colecistectomía. En el mismo se debe incluir

- Descripción.

RF 2.3 Gestionar datos Nomenclador de Causas de Intervención.

Este requisito es el responsable de registrar todos los datos referidos a las causas de intervención. En el mismo se debe incluir

- Descripción.

RF 2.4 Gestionar datos Nomenclador de Causas de Muerte.

Este requisito es el responsable de registrar todos los datos referidos a las causas de muerte implicadas en intervenciones de este tipo. En el mismo se debe incluir

- Descripción.

RF 2.5 Gestionar datos Nomenclador de Antecedentes Patológicos.

Este requisito es el responsable de registrar todos los datos referidos los antecedentes de tipo patológico. En el mismo se debe incluir

- Descripción.
- Categoría.

RF 2.6 Gestionar datos Nomenclador de Complicaciones.

Este requisito es el responsable de registrar todos los datos referidos a las posibles complicaciones a ocurrir. En el mismo se debe incluir

- Descripción

RF 2.7 Gestionar datos Nomenclador de Diagnósticos.

Este requisito es el responsable de registrar todos los datos referidos a los posibles Diagnósticos. En el mismo se debe incluir

- Descripción

RF 2.8 Gestionar datos Nomenclador de Factores de Riesgo Litiasis.

Este requisito es el responsable de registrar todos los datos referidos a los factores de riesgo que predisponen a un paciente a un padecimiento de esta rama. En el mismo se debe incluir

- Descripción

RF 2.9 Gestionar datos Nomenclador de Hallazgos Quirúrgicos.

Este requisito es el responsable de registrar todos los datos referidos a los posibles hallazgos durante el acto quirúrgico. En el mismo se debe incluir

- Descripción.

RF 2.10 Gestionar datos Nomenclador de Otras Operaciones.

Este requisito es el responsable de registrar todos los datos referidos otras intervenciones quirúrgicas de interés. En el mismo se debe incluir

- Descripción

RF 2.11 Gestionar datos Nomenclador de Procederes Diagnóstico Transoperatorio.

Este requisito es el responsable de registrar todos los datos referidos a los procederes de diagnóstico transoperatorio factibles a ser usados durante una intervención de Colecistectomía. En el mismo se debe incluir

- Descripción.

RF 2.12 Gestionar datos de Cirujanos.

Este requisito es el responsable de registrar todos los datos referidos a los Cirujanos que pueden efectuar el acto quirúrgico. En el mismo se debe incluir

- CI
- Nombre.
- Apellidos.
- Especialización.

RF 3 Gestionar Información de Pacientes.

Este requisito es el encargado de gestionar la información básica acerca de los pacientes remitidos (antes de iniciado el seguimiento). Acerca de estos se debe de almacenar:

- Nro de CI
- Sexo
- Color de piel
- Nombre(s)
- Primer Apellido
- Segundo Apellido
- Foto
- Dirección
- Municipio
- Provincia

RF 4 Realizar seguimiento a paciente.

Este requisito realiza el seguimiento de los estadíos por los que pasa el paciente, esta dividido en 3 requisitos que corresponden a las etapas en las se divide el seguimiento.

RF 4.1 Gestionar información etapa preoperatoria

En este requisito se gestiona la información correspondiente a dicha etapa en la cual el cirujano recoge datos de historia clínica y orienta una serie de exámenes complementarios, se debe introducir información referente a:

- Cuadro Clínico

- Tiempo de evolución
- Peso
- Talla
- Ultrasonidos realizados
- Antecedentes patológicos
- Factores riesgo de litiasis
- Intervenciones previas
- Resultado examen FAS
- Resultado examen TGP
- Resultado examen Eritrosedimentación
- Resultado examen Bilirrubina Directa
- Resultado examen Bilirrubina Total
- Diagnóstico presuntivo

RF 4.2 Gestionar información etapa transoperatoria

En este requisito se gestiona la información correspondiente al acto quirúrgico, se debe introducir información referente a:

- Cirujano que efectúa
- Fecha
- Tiempo quirúrgico
- Tipo de operación
- Tipo de extracción de vesícula
- Tipo de Colecistectomía realizada
- Tipo de laparoscopía realizada
- Diagnóstico quirúrgico
- Uso drenaje
- Uso antibiótico
- Otras operaciones realizadas durante el acto
- Accidentes propios de la técnica
- Hallazgos quirúrgicos

RF 4.3 Gestionar información etapa postoperatoria

En este requisito se gestiona la información correspondiente a la etapa postoperatoria, o sea posterior al acto quirúrgico, se debe introducir información referente a:

- Tiempo estadía hospitalaria
- Complicaciones postoperatorias
- Diagnóstico anamopatológico
- Fecha de alta hospitalaria
- Fallecimiento (si procede)
- Causa de fallecimiento (si procede)

RF 5 Consultar información almacenada

Este requisito es el responsable de permitir consultar la información almacenada filtrándola mediante criterios. Dichos criterios serán construidos por el usuario mediante el uso de uno o la combinación de varios de los listados a continuación.

- Rango de edades (en el momento de la intervención)
- Sexo
- Raza
- Municipio de procedencia
- Antecedentes patológicos personales
- Factores riesgo para padecer litiasis vesicular
- Intervenciones quirúrgicas previas
- Cuadro clínico
- Resultados exámenes complementarios
- Tipo de colecistopatías
- Cirujano que opera
- Distribución según tiempo quirúrgico
- Correlación entre el diagnostico preoperatorio y el postoperatorio
- Hallazgos quirúrgicos
- Tipo de operación
- Colecistectomía realizada
- Cuantía estadía hospitalaria
- Causas de conversión
- Procederes diagnóstico transoperatorio usados
- Otras operaciones realizadas
- Tipo de laparoscopia
- Extracción de la vesícula
- Uso de drenaje

- Uso de antibiótico peri operatorio
- Ocurrencia accidentes propios de la técnica
- Complicaciones quirúrgicas postoperatorias
- Fallecidos
- Causas Fallecimiento

RF 6 Generar reportes

Este requisito es el responsable de generar salidas visualizables e imprimibles de la información almacenada.

RF 7 Realizar salva de seguridad de la base de datos.

Este requisito es el responsable de permitir realizar salvas periódicas de la información almacenada así como la restauración de dichas salvas ante la ocurrencia de eventos que impliquen la reinstalación del sistema o el traslado de los datos a otro servidor de bases de datos.

RF 8 Gestionar información de conectividad a servidor de bases de datos

Este requisito debe permitir modificar la configuración de conectividad de nuestra aplicación, Ej. Modificar el nombre del servidor de bases de datos a conectarse, cuenta de usuario o contraseña de dicho servidor.

2.4 Actores del Sistema

Los actores no forman parte del sistema. Un actor es una entidad externa al sistema que de alguna manera participa en una funcionalidad de este. Generalmente estimula al sistema con eventos de entrada, o bien recibe algo de él (Castillero, 2003).

Debido a la escasa complejidad del modelo de seguridad presentado se determinan dos roles de usuario del sistema , el Administrador del sistema el cual va a interactuar con la totalidad del sistema más la gestión de usuarios y el actor denominado Cirujano el cual representa el rol de los usuarios que van a interactuar con las funcionalidades del sistema inherentes al proceso de seguimiento ,además de estos están presentes dos actores externos representados por los conversores de DICOM EzDicom y Dicompresor.

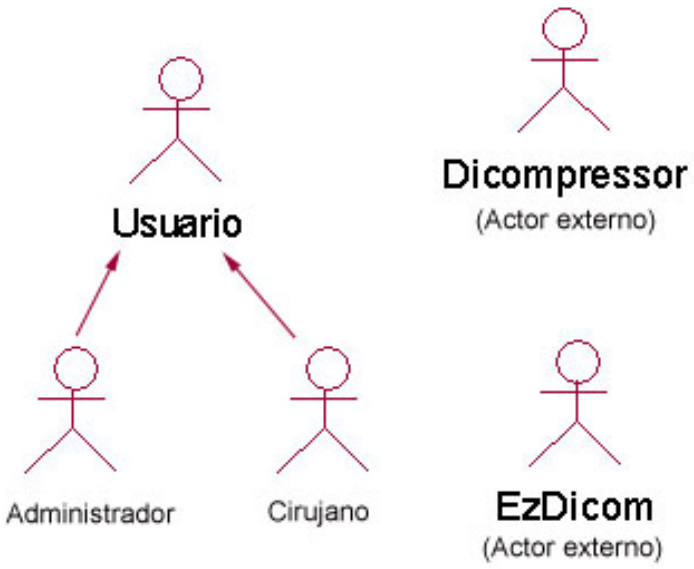

Fig. 4 Diagrama de actores del sistema

2.5 Casos de uso del sistema

Un caso de uso constituye una técnica utilizada para describir el comportamiento del sistema, a través de un documento narrativo que define la secuencia de acciones que obtienen resultados de valor para un actor que utiliza un sistema para completar un proceso, sin importar los detalles de la implementación. Ningún sistema se encuentra aislado. Cualquier sistema interesante interactúa con actores humanos, mecánicos u otros sistemas, que lo utilizan con algún objetivo y que esperan que el sistema funcione de forma predecible. Un caso de uso especifica el comportamiento de un sistema o de una parte del mismo, y es una descripción de un conjunto de secuencias de acciones, incluyendo variantes, que ejecuta un sistema para producir un resultado observable de valor para un actor. Los casos de uso se emplean para capturar el comportamiento deseado del sistema en desarrollo, sin tener que especificar cómo se implementa ese comportamiento. Los casos de uso proporcionan un medio para que los desarrolladores, los usuarios finales del sistema y los expertos del dominio lleguen a una comprensión común del sistema.(Rodríguez, 2010)

Derivados de los requisitos funcionales se obtienen casos de uso de primer nivel:

CU1 Gestionar usuarios y seguridad

CU2 Gestionar Nomencladores

CU3 Gestionar Información de pacientes

CU4 Realizar seguimiento de paciente

CU4.1 Gestionar información seguimiento etapa preoperatorio

CU4.2 Gestionar información seguimiento etapa transoperatorio

CU4.3 Gestionar información seguimiento etapa postoperatorio

CU5 Consultar información almacenada

CU6 Generar reportes

CU7 Realizar salva de seguridad de la base de datos

CU8 Gestionar información de conectividad al servidor de bases de datos

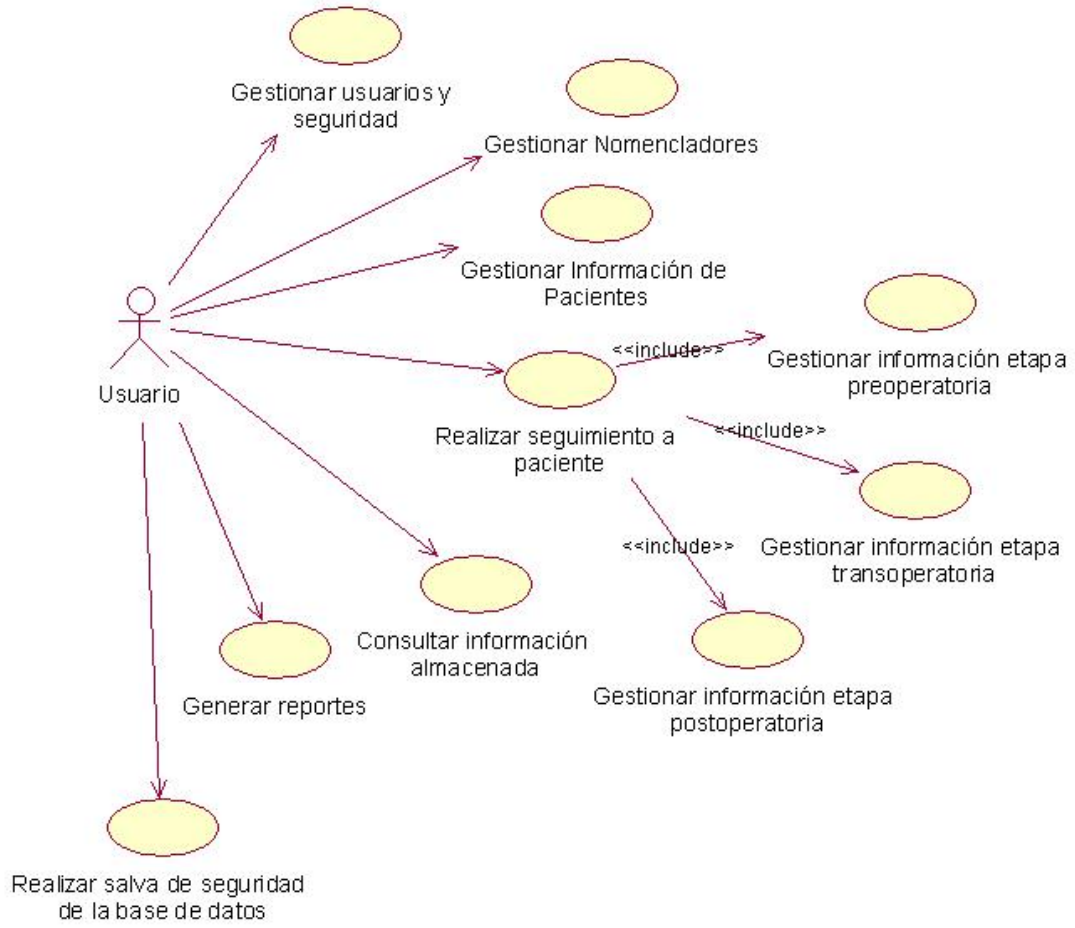

Fig 5 Casos de uso del sistema

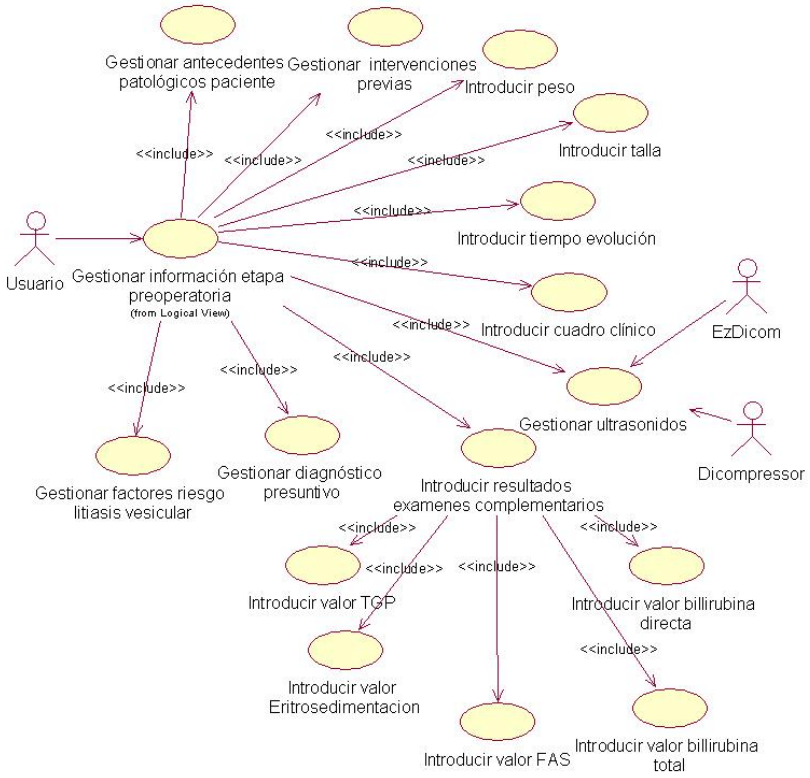

Fig. 6 Casos de uso del sistema Gestionar información etapa preoperatoria

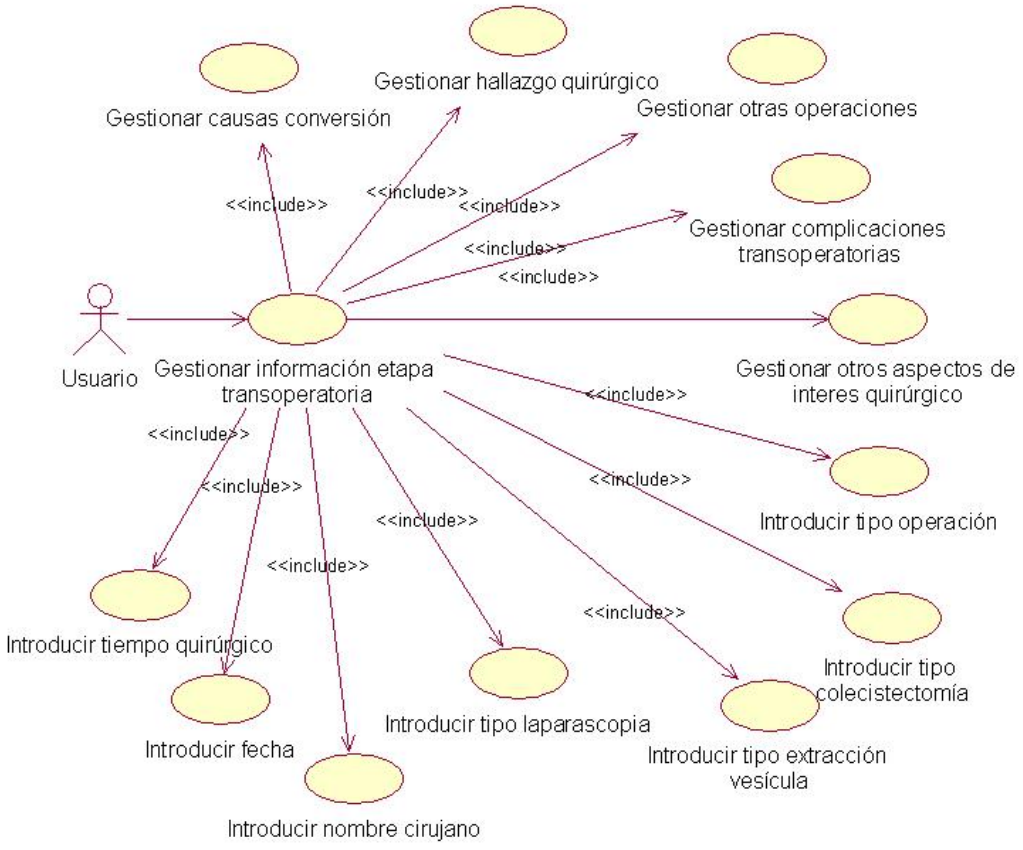

Fig. 7 Casos de uso del sistema Gestionar información etapa transoperatoria

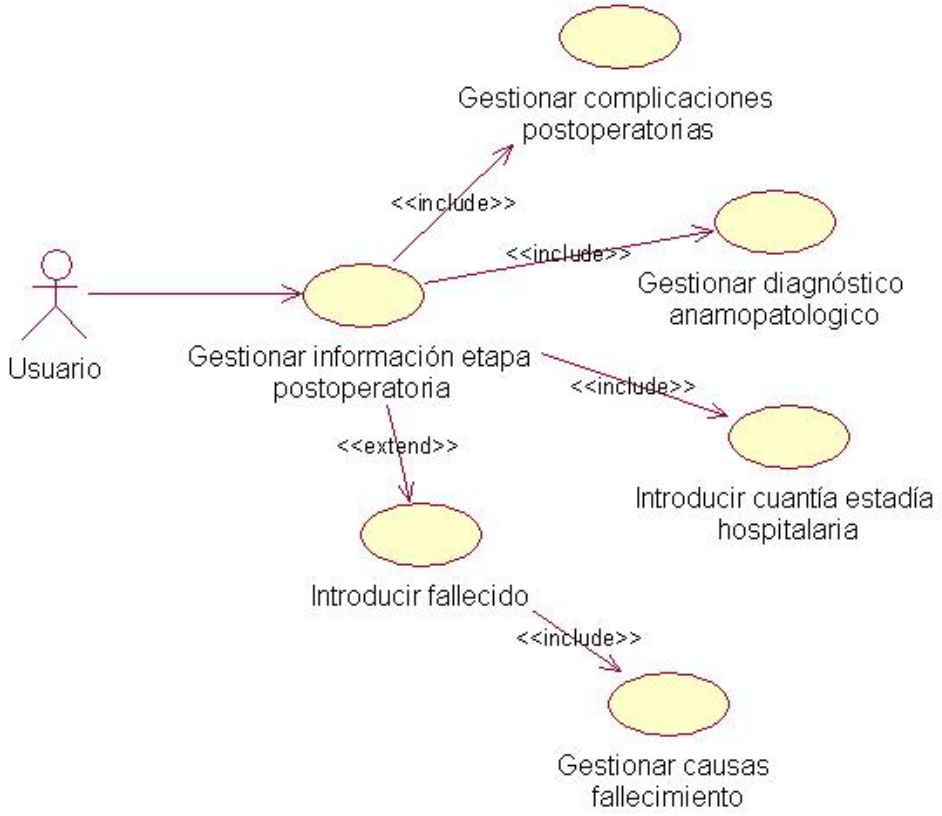

Fig. 8 Casos de uso del sistema Gestionar información etapa postoperatoria

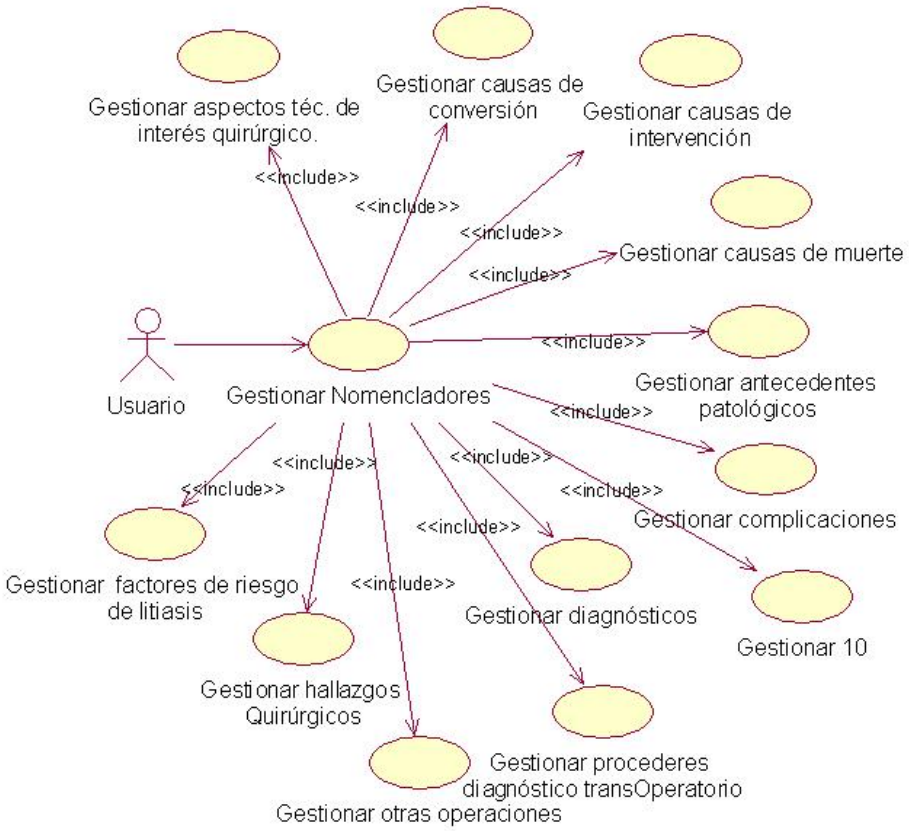

Fig. 9 Casos de uso del sistema Gestionar información etapa transoperatoria

CAPÍTULO 3. DISEÑO E IMPLEMENTACIÓN DE LAPAROGEST

En la comunidad Java y en el mundo del desarrollo de software en sentido general se ha establecido una fuerte polémica en cuanto a las ventajas y desventajas de las dos plataformas de despliegue de aplicaciones informáticas por excelencia : Desktop vs. Web. El indetenible desarrollo de Internet y el despliegue de la Cloud Computing para infinidad de usos tanto a nivel de usuario como empresariales le ha dado un fuerte espaldarazo a las aplicaciones soportadas sobre “thin client” como se ha dado a llamar a las aplicaciones alojadas en un servidor de aplicaciones Web (IIS, Apache) y desplegadas sobre cualquiera de los navegadores comerciales disponibles (Mozilla Firefox, Opera, Internet Explorer, Google Chrome) entre las ventajas de esta plataforma podemos enumerar:

- Facilidad de soporte
- Facilidad de actualización
- Reducidos requisitos de hardware por parte del cliente
- Independencia de plataforma
- Posibilidad de ejecutar desde cualquier estación conectada

No obstante las innegables ventajas de dicha plataforma las fortalezas de la plataforma denominada “fat client”, mas comúnmente llamada Desktop o Escritorio, sigue haciendo de esta una elección viable a la hora de determinar arquitecturas de software, entre dichas ventajas podemos enumerar:

- Interacción directa con recursos del sistema
- Tiempo de respuesta más rápido
- Mejor interacción con recursos de bajo nivel (hardware, etc.)
- No dependen de recursos de conectividad externos
- Mayor riqueza en componentes de interfaz gráfica disponibles

En correspondencia con los requisitos obtenidos se decidió implementar una aplicación Java de escritorio conectada a un servidor PostreSQL que pudiera ser local o remoto

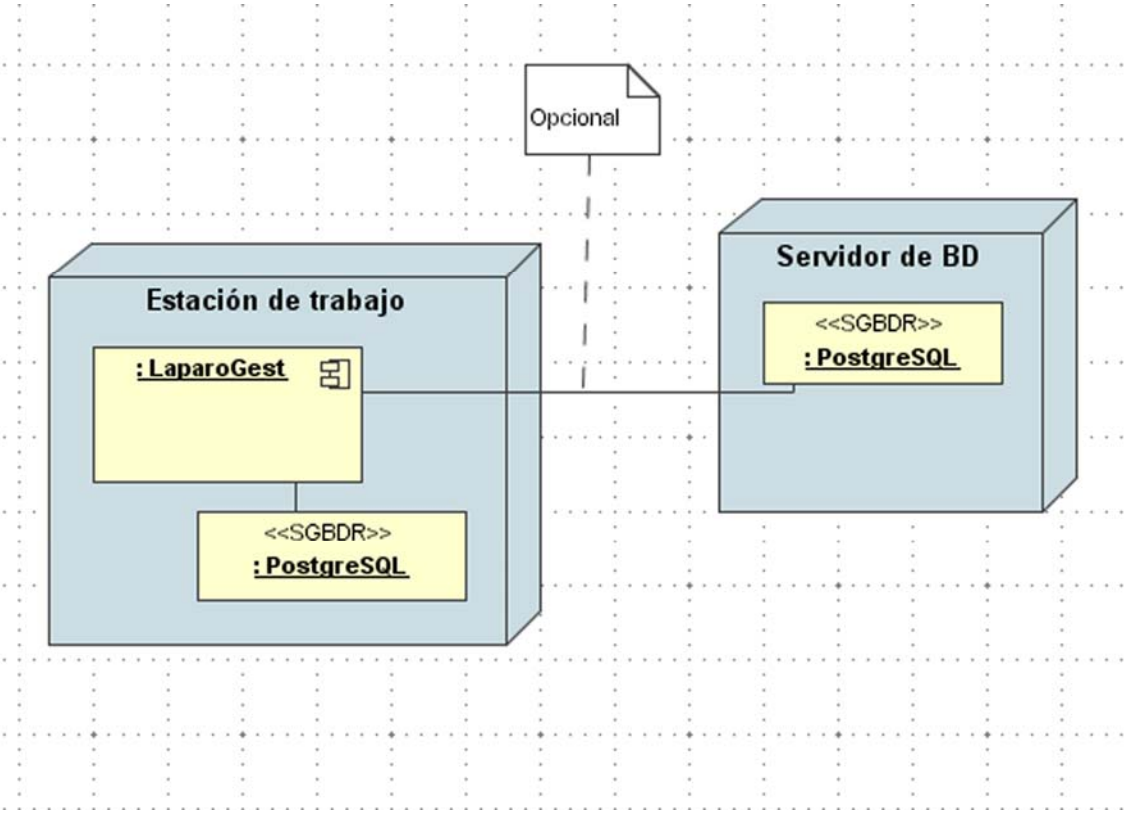

Fig. 10 Diagrama despliegue

3.1 Herramientas utilizadas

Para la implementación de la propuesta de software llamada a dar solución al problema enunciado se escogieron una serie de herramientas establecidas de probada efectividad, a continuación se enumeran las más significativas:

- Java Runtime Enviroment 6
- Java SE Software Development Kit 6
- Rational Rose Enterprise Edition
- EMS SQL Manager for PostgreSQL
- Adobe Photoshop CS3 para el diseño de la iconografía
- NetBeans IDE desde la versión 5.0 hasta la 6.8 con los siguientes módulos:
    - Plugin de UML para Netbeans
    - Diseñador de IGU Matisse
    - Plugin de IReport Designer para la elaboración de los reportes
    - Plugin de Subversion para manejo de repositorio y control de versiones

3.3 Diseño de la Base de datos

En el trabajo con frameworks ORM como es el caso de JPA se pueden utilizar dos variantes a la hora de definir la capa de datos de una aplicación:

Ingeniería directa: consiste en indicarle a Hibernate que genere tablas sobre un SGBD a partir de las clases del modelo.

Ventajas: no requiere de scripts de instalación de la BD, es puramente orientada a objetos y genera una capa de datos portable.

Desventajas: bases de datos menos eficientes desde el punto de vista relacional, se desaprovechan fortalezas específicas de gestores de BD's.

Ingeniería inversa: consiste en generar clases del modelo que representen entidades ya definidas en un modelo relacional.

Ventajas: BD's eficientes y normalizadas que permiten aprovechar fortalezas del SGBD utilizado, control total sobre el modelo.

Desventajas: requiere de scripts de instalación de la BD generados y controlados por el desarrollador, menor portabilidad, los cambios hay que replicarlos manualmente de la capa de datos al modelo.

Debido a las ventajas antes mencionadas y la experiencia de trabajo del equipo en el modelado relacional se decide emplear la variante de Ingeniería Inversa sobre una BD PostgreSQL ya definida.

Para el diseño de la BD, se utiliza el modelo entidad – relación extendido, el cual permite modelar objeto del mundo real en forma de entidades, atributos y sus interrelaciones, capturando de esta manera en un modelo de datos computacional los requisitos obtenidos durante la etapa de análisis. (Ver Anexo 2)

Como resultado de este proceso se implementaron 32 tablas sobre el SGBDR seleccionado.

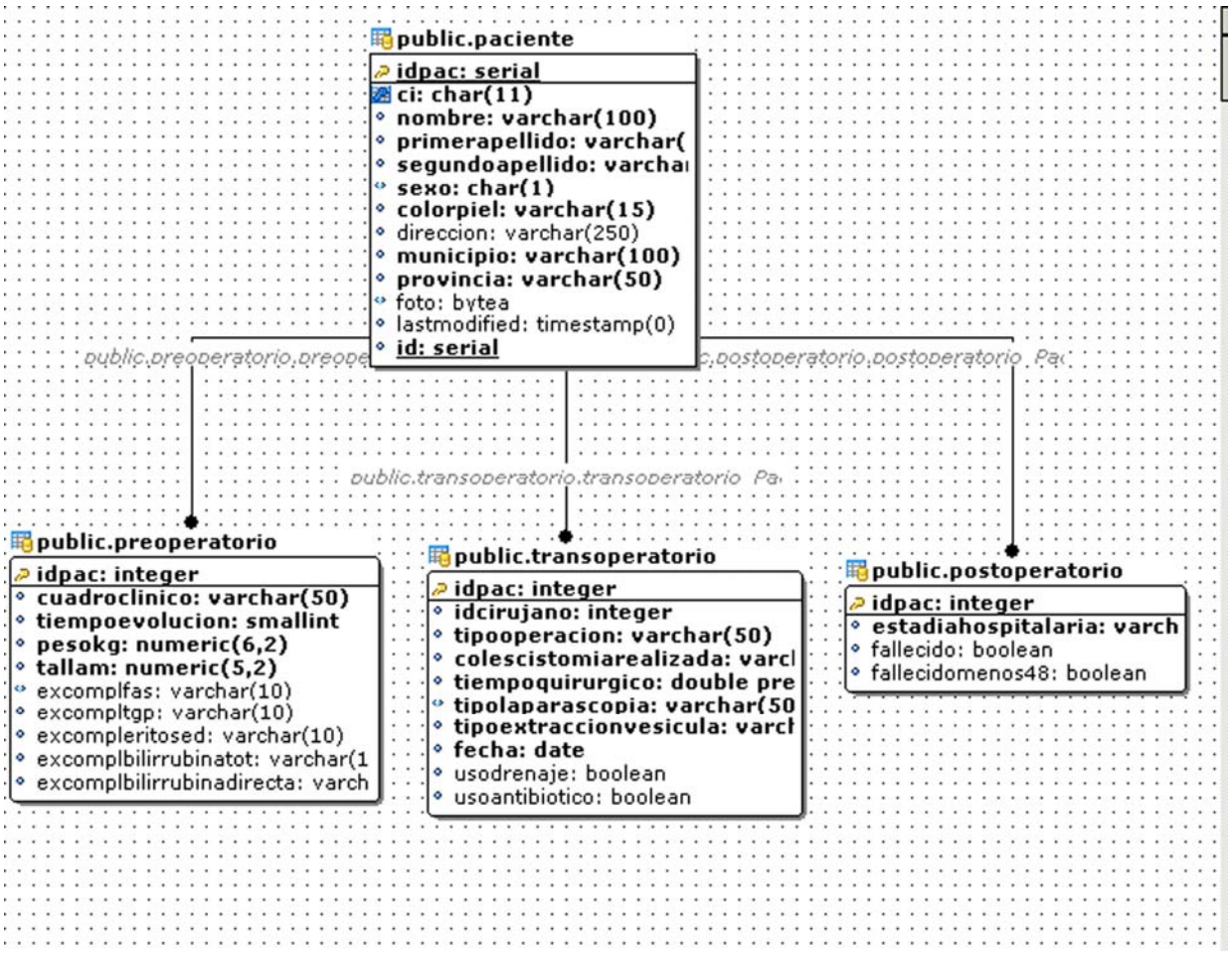

Fig. 11 Diagrama BDs

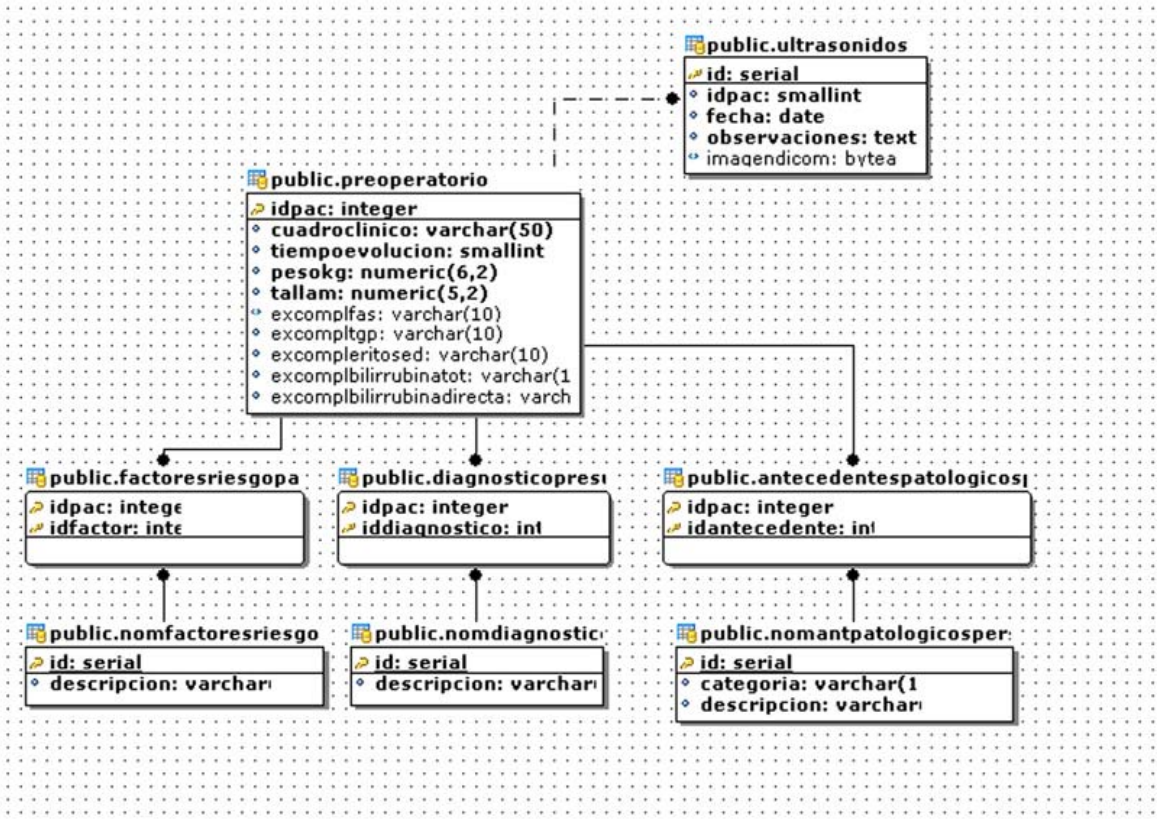

Fig. 11.1 Diagrama BDs etapa preoperatoria

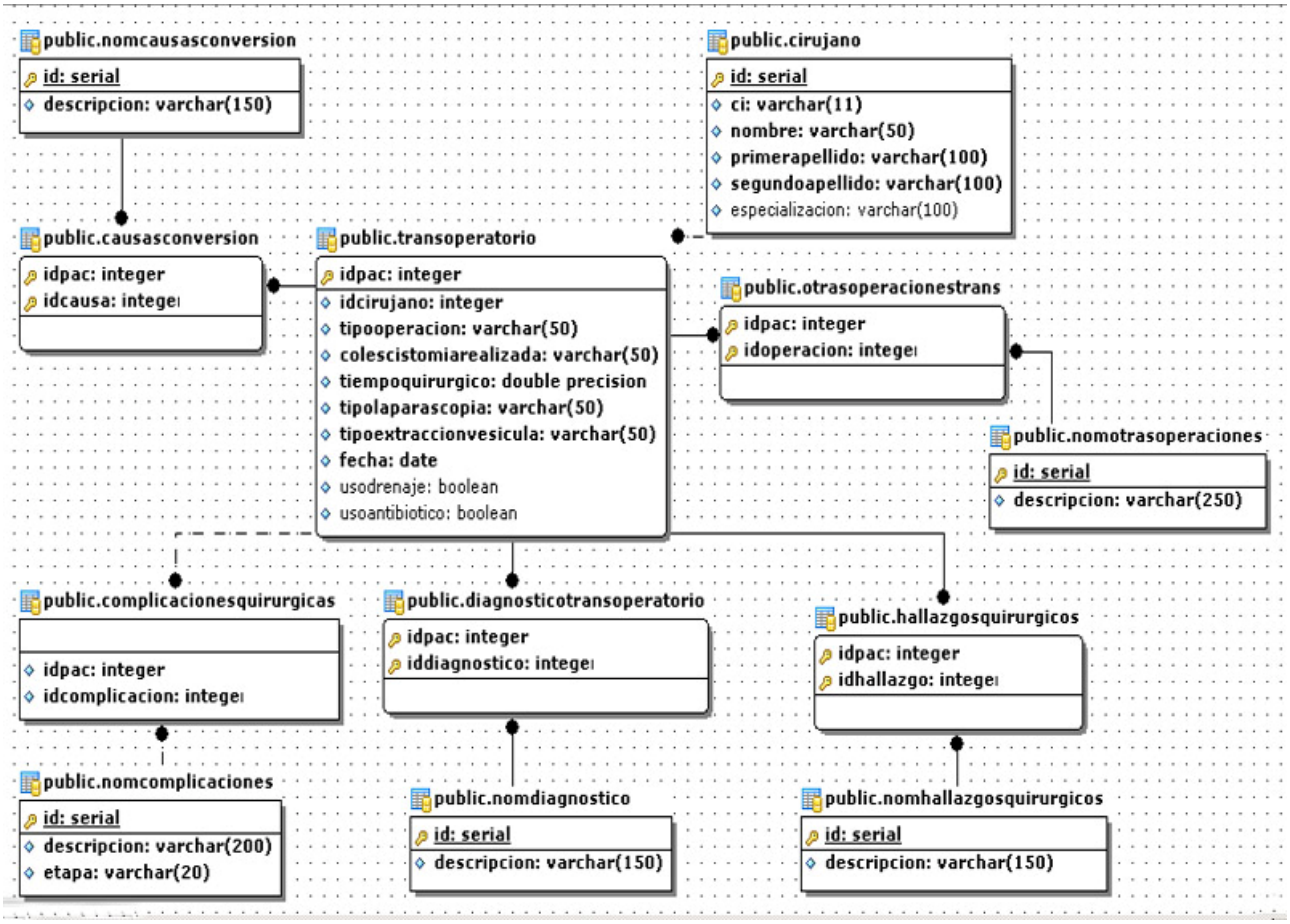

Fig. 11.2 Diagrama BDs etapa transoperatoria

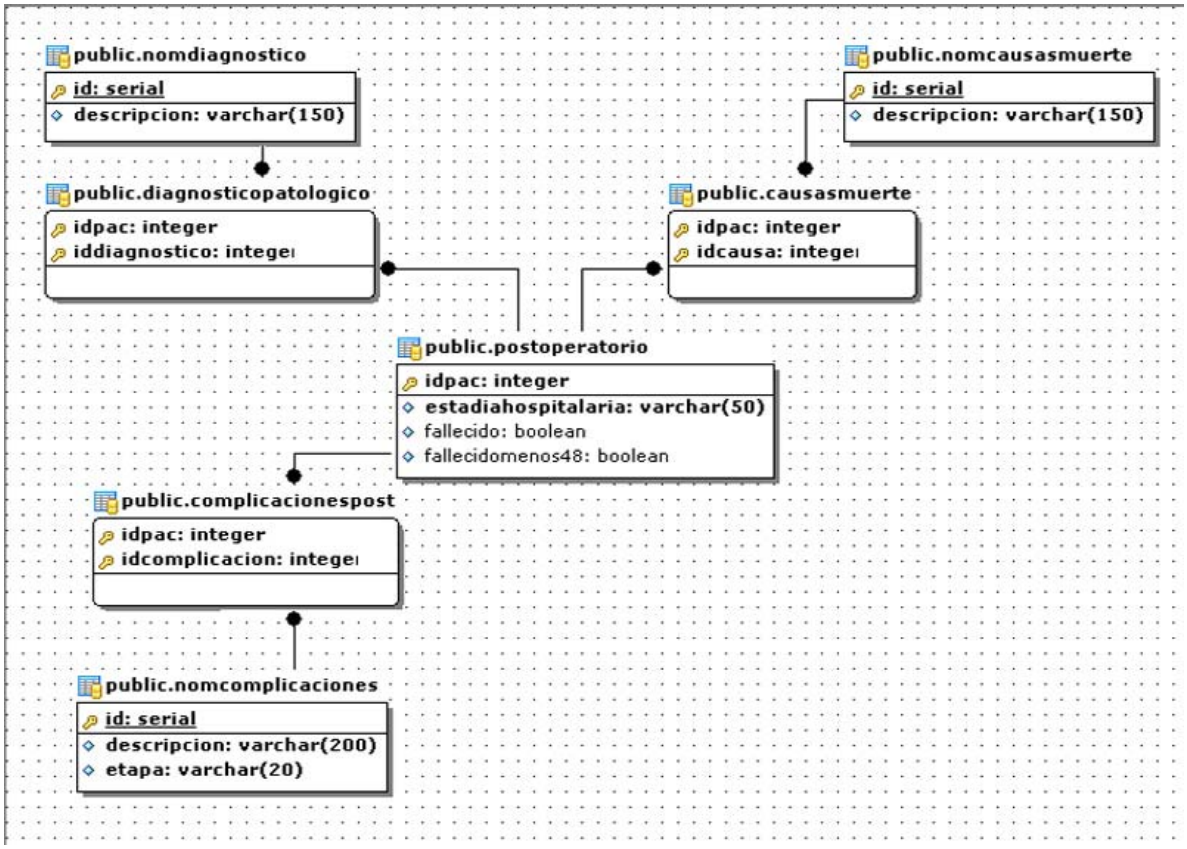

Fig. 11.3 Diagrama BDs etapa postoperatoria

3.4 Arquitectura de capas de la aplicación

Nombre de la capa	Responsabilidad	Implementación tecnológica
Presentación	Interfaz gráfica de usuario	Swing
Enlace	Relación dominio-IGU	Beans Binding
Dominio	Modelo del dominio, lógica del dominio de negocio, mapeo objeto-relacional	POJO (con Annotations)
Persistencia	Persistencia de objetos del dominio	JPA con Hibernate

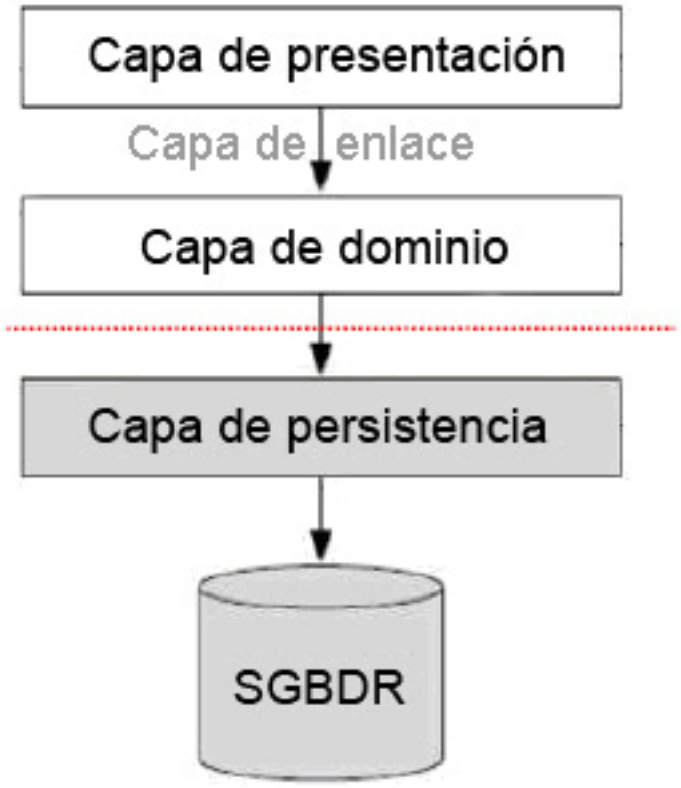

Fig.12 Diagrama estructura capas

En las aplicaciones Java de escritorio que utilizan el API Swing se establece una arquitectura derivada de la conocida como Modelo-Vista-Controlador llamada Modelo-Delegado (Deitel, 2001) en la cual los delegados de los componentes de interfaz grafica de usuario y la clase EntityManager del JPA absorben la capa controladora de la arquitectura clásica, el uso de la API Beans Binding permite una capa de enlace entre el modelo y las propiedades de dichos componentes que facilita y la vez agiliza en grado sumo dicho proceso.

3.5 Estructura del proyecto

De acuerdo a la arquitectura seleccionada se procedió a definir la estructura del proyecto en una serie de paquetes.

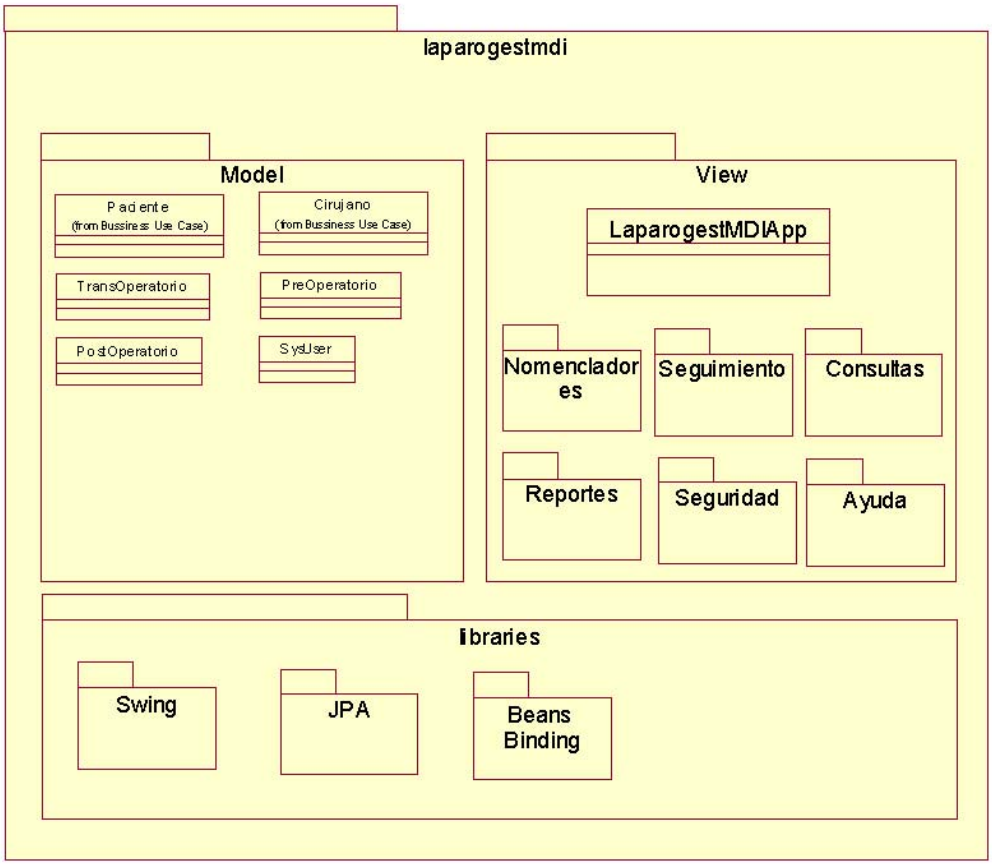

Fig. 13 Diagrama paquetes

La estructura de paquetes se compone de cuatro paquetes de nivel superior definidos por el IDE para un proyecto de este tipo y un conjunto de paquetes definidos por el desarrollador a partir de estos:

Source Packages: Contiene los fuentes del proyecto

- META-INF: almacena la unidad de persistencia en el archivo persistente.xml
- laparogestmdi: carpeta raíz del proyecto
    - extensions: contiene clases de utilidad y extensiones del proyecto
    - bdbackup: contiene salvas periódicas de la bd en etapa de codificación
    - model: contiene las clases del modelo
    - view: Contiene las clases de la capa de presentación o IGU
- Libraries: Almacena las librerías a utilizar en el proyecto

3.6 Clases de modelo del negocio

Se identifican 17 entidades del negocio y las relaciones que se establecen entre ellas.

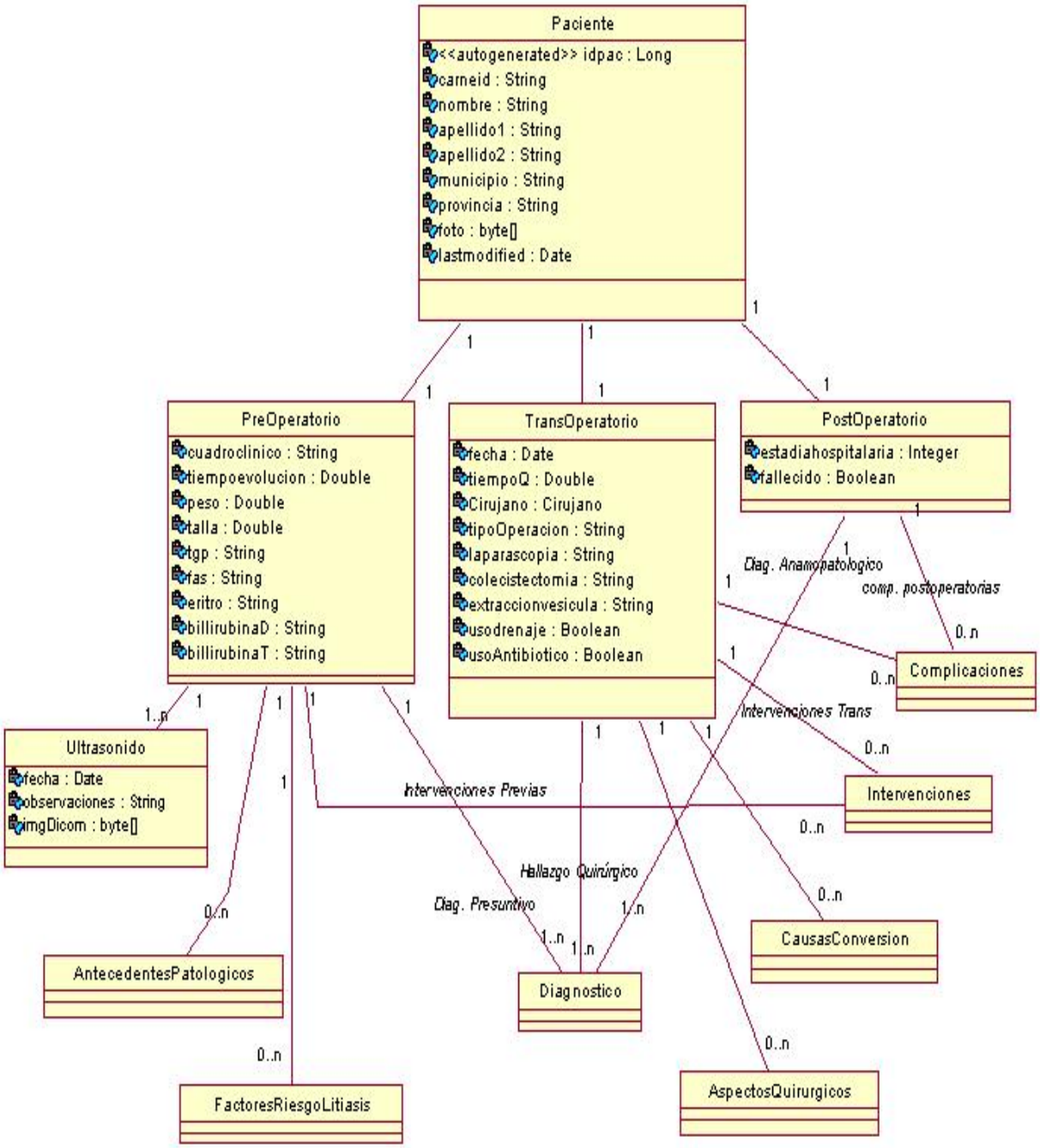

Fig. 14 Diagrama de clases del modelo

Una entidad es un objeto de dominio de persistencia. Normalmente, una entidad representa una tabla en el modelo de datos relacional y cada instancia de esta entidad corresponde a un registro en esa tabla. El estado de persistencia de una entidad se

representa a través de campos persistentes o propiedades persistentes. Estos campos o propiedades usan anotaciones para el mapeo de estos objetos en el modelo de base de datos. El estado persistente de una entidad puede ser accesible a través de variables de instancia a la entidad o bien a través de las propiedades de estilo de JavaBean.

En el paquete model de la estructura del proyecto se almacenan las clases persistentes llamadas a representar las entidades obtenidas en el modelado del dominio (ver Fig. 14). Siguiendo la arquitectura escogida se generan clases POJO (Plain Old Java Objects) en los cuales se representa la estructura de la capa de datos mediante el uso de Annotations asignados a los atributos u operaciones de la clase.

Las entidades podrán utilizar campos persistentes o propiedades. Si las anotaciones de mapeo se aplican a las instancias de las entidades, la entidad utiliza campos persistentes, En cambio, si se aplican a los métodos getters de la entidad, se utilizarán propiedades persistentes. Si la entidad utiliza campos persistentes, los accesos se realizan en tiempo de ejecución. Aquellos campos que no tienen anotaciones del tipo javax.persistence.Transient o no han sido marcados como Java transitorio serán persistentes para el almacenamiento de datos. Las anotaciones de mapeo objeto/relación deben aplicarse a los atributos de la instancia. Si la entidad utiliza propiedades persistentes, la entidad debe seguir el método de los convenios de componentes JavaBeans. Las propiedades de JavaBean usan métodos getters y setters en cuyo nombre va incluido el atributo de la clase al cual hacen referencia.

Uso de Annotations

El mapeo objeto/relacional, es decir, la relación entre entidades Java y tablas de la base de datos, se realiza mediante anotaciones en las propias clases de entidad, por lo que no se requieren ficheros descriptores XML. También pueden definirse transacciones como anotaciones JPA. (Chisty, 2005)

Las Anotaciones se adicionan en la versión 5 del JDK, están definidas en la literatura como “meta-tags que se pueden adicionar al código y aplicarlas a declaraciones de paquetes, declaraciones de tipo, constructores, métodos, atributos, parámetros y variables” (Chisty, 2005)También se pueden definir como un método para asociar meta-tags a elementos de código para de esa forma permitir al compilador o la maquina virtual extraer comportamientos deseados de dichos elementos y generar código en caso de ser necesario para implementar este comportamiento. Las anotaciones no afectan directamente la semántica del programa pero afectan la manera el código es tratado por herramientas y librerías, lo cual puede conllevar a cambios en la semántica del programa

en tiempo de corrida, las anotaciones pueden ser leídas desde código fuente, archivos o mediante Reflextion en tiempo de corrida("Annotations," 2004).

En el uso de Hibernate desde JPA se sustituyen los descriptores de las clases en formato .XML por anotaciones (*javax.persistence.**) introducidas en el código fuente logrando con esto una mayor legibilidad y facilidades de mantenimiento, no obstante se mantiene la posibilidad de usar el formato xml en caso de necesidad o para facilitar la reusabilidad de código proveniente de otras aplicaciones.

Un ejemplo que describe concretamente el uso de dicha filosofía de diseño lo representa la clase Paciente (*laparogestmdi.model.Paciente*) (véase anexo X), la cual constituye la raíz de la estructura jerárquica a partir de la cual se establece el modelado del proceso de seguimiento. En los fragmentos de código a continuación se describen algunos de los principales usos de dicha librería en la arquitectura definida.

```
@Entity
@Table(name="paciente")
@NamedQueries({
    @NamedQuery(name = "Paciente.findAll", query = "FROM Paciente p"),
    @NamedQuery(name = "Paciente.findByIdPac", query = "SELECT p FROM Paciente p
WHERE p.idpac = :id"),
   @NamedQuery(name = "Paciente.findByCI", query = "FROM Paciente p WHERE
p.carneid = :ci")
})
```

En este fragmento de código se expresa mediante el uso de la anotación @Entity que la clase va a ser un Java Beans, mediante @Table a que relación del modelo corresponde y mediante el uso de @NamedQuery se definen consultas en lenguaje HPQL sobre la entidad que pueden ser invocadas usando clases de JPA.

```
@Id
  @Generated Value(strategy = GenerationType.IDENTITY)
  @Column(name="idpac", unique = true, nullable = false)
  public Long getidpac()   {     return idpac;   }
```

En este fragmento de código se especifica mediante la anotación *@Id* que dicho atributo constituye el identificador de la entidad (llave primaria en el modelo relacional), mediante *@GeneratedValue* el tipo de generación a emplear y mediante *@Column* con sus atributos *name ,unique* y *nullable* la relación con el campo de la tabla ,la unicidad y la restricción de introducir valores nulos respectivamente , dicha atributo representa un atributo numérico autogenerado expresado en el modelo mediante un tipo Serial de PostgreSQL

```
@Column (name="`ci`") @NotNull @Length (min = 1, max = 11)
public String getcarneid()  {  return carneid;  }
```

En este caso se introducen mediante el uso de la anotación @Length restricciones del dominio a los valores del atributo, en este caso de longitud para un carné de identidad.

```
@Version @NotNull
 public Timestamp getLastModified() { return lastmodified;}
```

El empleo de la anotación @Version especifica un atributo de manejo de tiempo que se actualizará en cada modificación de la instancia.

```
@Lob  @Basic(fetch=FetchType.LAZY)
   public byte[] getFoto() { return (foto);}
```

El uso de @Lob permite especificar que se estar refiriendo aun objeto de tipo binario, en este caso una imagen almacenada en un campo de la base de datos, el uso de FetchType.LAZY como valor de la propiedad fetch indica que dicha imagen solo se cargara desde el modelo en caso de invocación directa del atributo.

```
 @OneToOne(fetch=FetchType.LAZY,cascade=CascadeType.ALL)
 @PrimaryKeyJoinColumn
 public PostOperatorio getPostoperatorio() { return postoperatorio;}
```

La anotación @OneToOne permite expresar una relación uno a uno en el modelo, en este caso expresada por la restricción del dominio que indica que por la naturaleza de la intervención un paciente puede tener uno y solo un postoperatorio, @PrimaryKeyColumn

expresa que dicha relación se mantiene mediante el uso de la llave primaria en la clase referida, cascade=CascadeType.ALL indica que se replican las operaciones de modificación u borrado del padre en cascada.

```
protected List<NomComplicaciones> complicacionespost;
@OneToMany
@JoinTable(name="complicacionespost",joinColumns=@JoinColumn(name="idpac"),
                                inverseJoinColumns=@JoinColumn(name="idcomplicacion"))
public List<NomComplicaciones> getComplicacionespost() {
      return complicacionespost;
}
```

En el fragmento anterior se ejemplifica una relación de una clase Postoperatorio (*laparogestmdi.model.Postoperatorio*) con los valores nomenclados para Complicaciones, expresada de la manera "Un paciente puede en su etapa postoperatoria tener de 0 a n complicaciones" .Dicha relación esta definida en el modelo relacional con el uso de a tabla intermedia complicacionespost para implementar una relación muchos a muchos entre la tabla postoperatorio y la tabla NomComplicaciones. En nuestro modelo de clases se representa como una lista de entidades NomComplicaciones adicionada como atributo del postoperatorio y mapeada mediante una anotación @OneToMany para especificar que solo es necesario representar dicha cardinalidad para navegar en este sentido y una anotacion @JoinTable que mediante su propiedad name permite especificar que tabla del modelo relacional se va utilizar como enlace , joinColumns e InverseJoinColumns permiten especificar las llaves primarias a utilizar en dicha relación. El uso de elementos del modelo relacional en este tipo de anotaciones ha sido criticado como una inclusión de los datos en un modelo de clases pero la experiencia demuestra que le imprime una potencia y una facilidad de codificación que hacen recomendable su uso.

Uso de Beans Binding en clases del modelo

El uso de la librería Beans Binding para mantener sincronizadas dos propiedades de Java Beans requiere que cada de unos de estos componentes mantenga una lista de escuchas (listeners) a los cuales notificar en caso de cambios en sus propiedades, para esto se emplean instancias de las clases PropertyChangeListener y PropertyChangeSupport tanto en las clases del modelo como en las clases de la vista. El fragmento de código a continuación ejemplifica el uso de dichas clases en una entidad del modelo.

```
 private PropertyChangeSupport changeSupport= new PropertyChangeSupport(this);

 public void addPropertyChangeListener(PropertyChangeListener listener){
     changeSupport.addPropertyChangeListener(listener);
 }
 public void removePropertyChangeListener(PropertyChangeListener listener){
    changeSupport.removePropertyChangeListener(listener);
}
 public void setcarneid(String carneId)  {
   String OldValue=this.carneid;
    this.carneid = carneId;
    changeSupport.firePropertyChange("carneid",OldValue,carneId );
}
```

En este caso se utiliza el método firePropertyChange de la instancia de PropertyChangeSupport incluida en la clase para notificar a los objetos registrados como oyentes de una modificación realizada a la propiedad carneid de una instancia de Paciente. Refleja el caso típico en el cual se notifica a controles de la capa de presentación de una modificación a una propiedad del modelo que deben reflejar.

3.7 Clases de presentación

En el paquete view de la estructura del proyecto se almacenan las clases de presentación de datos definidas mediante la API SWING. En una arquitectura MDI se emplea una pantalla principal que funciona como contenedor del resto de las pantallas de trabajo aunque para necesidades específicas se permite desplegar ventanas de diálogo. En esta aplicación se implementa esta arquitectura mediante el uso de la clase denominada LaparoGestMDIApp, descendiente de *javax.swing.JFrame* como pantalla principal, un conjunto de clases descendientes de *javax.swing.JInternalFrame* para el caso de las pantallas que se van a desplegar en el área de trabajo y descendientes de *javax.swing.JDialog* para el caso de las que se van a desplegar en forma de diálogo.

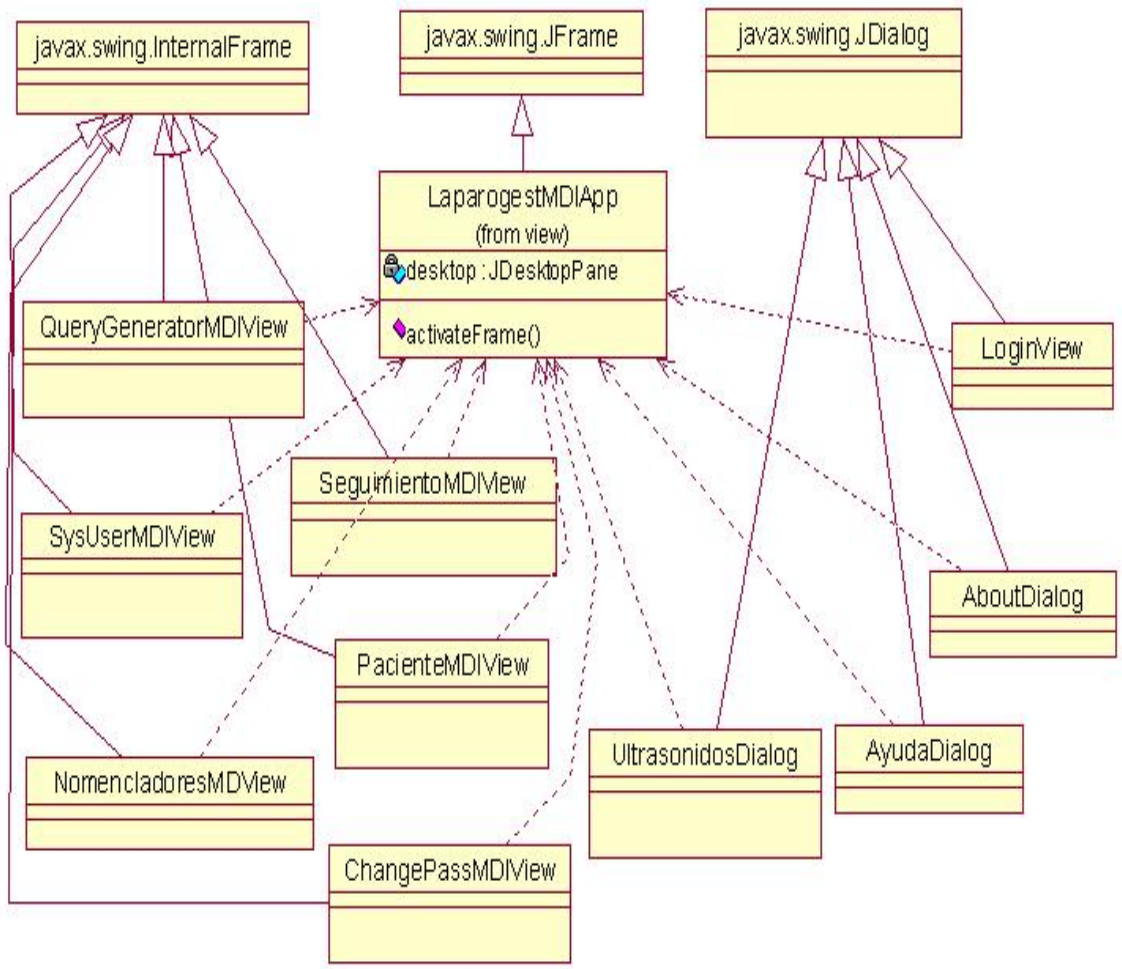

Fig. 15 Diagrama clases capa de presentación

Las clases de Swing presentan soporte para Beans Binding, dicho soporte puede activarse mediante código fuente o utilizando el submenú Bind presente en el menú contextual de los componentes.

Uso de Beans Binding en clases de presentación

La sincronización mediante código fuente entre un componente de la capa de presentación, en este caso una instancia de javax.swing.JTextField con la propiedad de una clase del modelo (la propiedad carneid de la clase Paciente) se ejemplifica en el siguiente fragmento:

```
org.jdesktop.beansbinding.Binding binding =
org.jdesktop.beansbinding.Bindings.createAutoBinding(
    org.jdesktop.beansbinding.AutoBinding.UpdateStrategy.READ,
    this,
```

```
    org.jdesktop.beansbinding.ELProperty.create("${paciente.carneid}"),
    HCTextField,
    org.jdesktop.beansbinding.BeanProperty.create("text")
);
bindingGroup.addBinding(binding);
```

La implementación de la misma funcionalidad se puede lograr usando el submenú Bind del menú contextual del control en la vista Diseño de la pantalla en NetBeans ver 5.0 o superior.

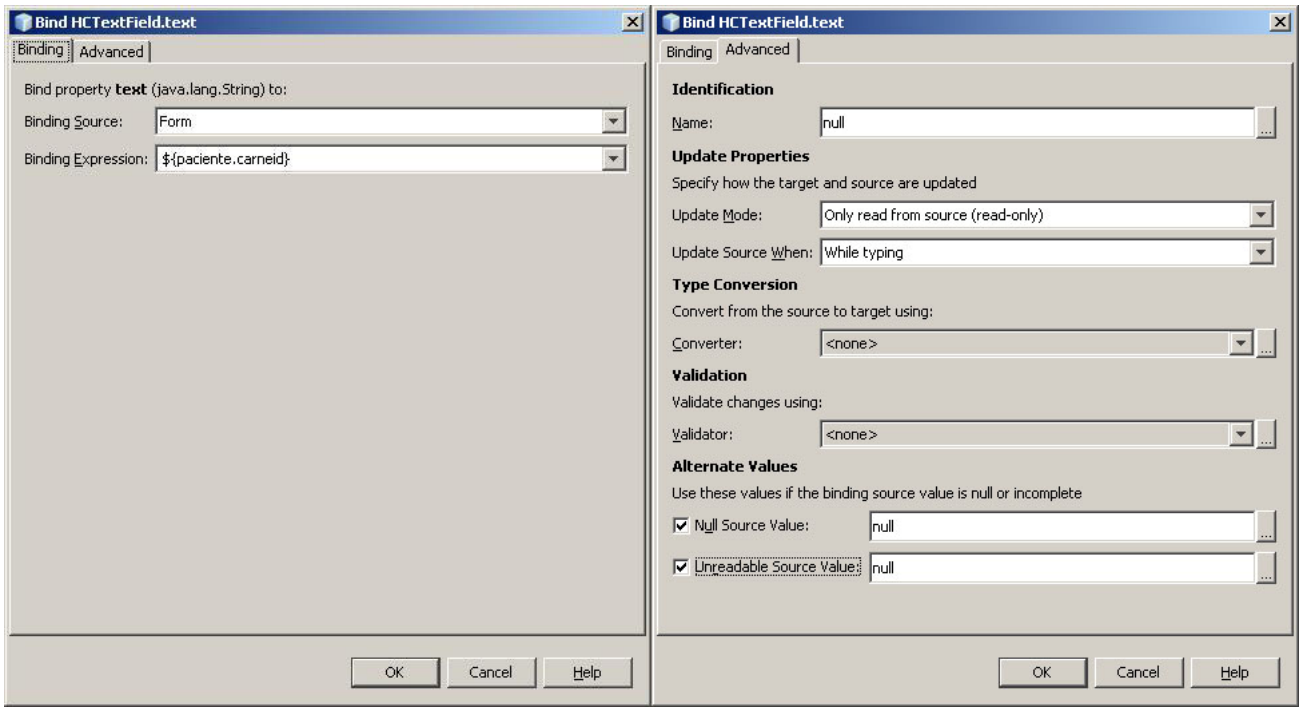

Fig. 16 Uso Beans Binding editor visual

3.8 Uso de la clase Entiy Manager

Para gestionar la interacción de las entidades con la JPA, una aplicación utiliza la interfaz EntityManager. Esta interfaz proporciona métodos que realizan las funciones comunes de una base de datos, como consulta y actualización de la base de datos, crea y elimina instancias de entidades persistentes, busca entidades a partir de su clave primaria y permite ejecutar consultas.

Las entidades son gestionadas por el Entity Manager. Éste se representa por instancias javax.persistence.EntityManager. A cada instancia de EntityManager se le asocia con un contexto de persistencia. Un contexto de persistencia define el ámbito particular, bajo el cual se crean o se eliminan las instancias de la entidad.

El contexto de persistencia

Un contexto de persistencia es un conjunto de instancias que existen en un almacén de datos. El interfaz EntityManager define los métodos que se utilizan para interactuar con el contexto de persistencia.

Unidades de persistencia

La unidad de persistencia define un conjunto de todas las entidades (clases) que son gestionadas por la instancia del EntityManager en una aplicación. Este conjunto de clases de entidad representa los datos contenidos en un único almacén de datos.

Las unidades de persistencia se definen en el fichero de configuración persistence.xml. El fichero JAR cuyo directorio META-INF contiene persistence.xml se llama raíz de la unidad de persistencia. El ámbito de la unidad de persistencia se determina por la raíz de la unidad de persistencia. Cada unidad de persistencia debe ser identificada con un nombre único en el ámbito de la unidad de persistencia. Las unidades de persistencia pueden empaquetarse como parte de un WAR o un fichero EJB JAR, o pueden empaquetarse como un archivo JAR que pueda ser incluido en un fichero WAR o EAR.

El fichero persistence.xml

El fichero persistence.xml define una o más unidades de persistencia:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" >
  <persistence-unit name="LaparoGestMDIPU" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <class>laparogestmdi.model.Cirujano</class>
    <class>laparogestmdi.model.SysUser</class>
```

```
    <properties>
      <property name="hibernate.dialect"
value="laparogestmdi.Extensions.PostgreSQLDialectExt"/>
      <property name="hibernate.ejb.naming_strategy"
value="laparogestmdi.Extensions.PostgreSQLNamingStrategy"/>
      <property name="hibernate.connection.username" value="postgres"/>
      <property name="hibernate.connection.driver_class" value="org.postgresql.Driver"/>
      <property name="hibernate.connection.password" value="admin"/>
      <property name="hibernate.connection.url" value="jdbc:postgresql://localhost/LaparoGest"/>
      <property name="hibernate.connection.pool_size" value="2"/>
      <!-- Opciones de SQL debug  -->
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
      <property name="hibernate.use_sql_comments" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```

Este archivo define una unidad de persistencia llamada LaparoGestMDIPU, la cual utiliza org.postgresql.Driver como driver de conexión con el SGBD. El fichero JAR y los elementos de la clase especifican las clases de persistencia: clases de entidad, clases embebidas y superclases mapeadas.

El elemento jar-file especifica los ficheros JAR en los que se encuentran las clases persistentes, mientras que los elementos class indica el nombre de esas clases.

CAPÍTULO 4 LAPAROGEST: SISTEMA PARA LA GESTIÓN DE LA INFORMACIÓN ACERCA DE INTERVENCIONES DE COLECISTECTOMÍA LAPAROSCOPICA

LaparoGest (Laparo proveniente del nombre de la técnica utilizada en las intervenciones quirúrgicas y Gest por gestión de información) se presenta al usuario como una aplicación multiplataforma de escritorio de tipo MDI (Multiple Document Interface) con un diseño sobrio y amigable de cara al usuario. En este capítulo se procede a describir algunos puntos de su funcionamiento e interfaz de usuario.

Para la ejecución de la aplicación se requiere

- Java Runtime Enviroment (JRE)
- Java Development Kit (JDK)
- PostgreSql 8.4

4.1 Autenticación

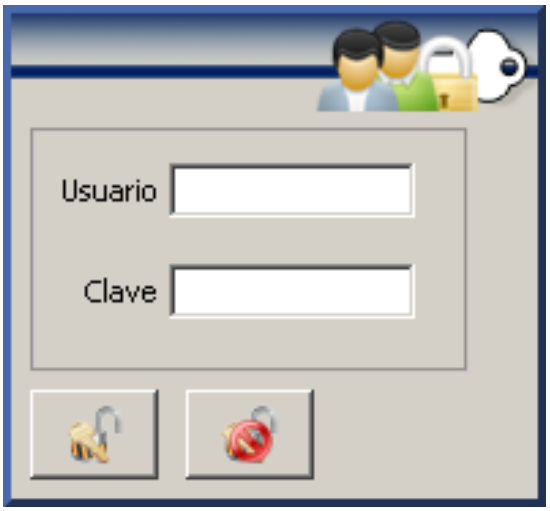

Fig. 17 Formulario Autenticación

La ventana de autenticación se muestra de forma modal sobre la pantalla principal al iniciar el sistema con el objetivo de prevenir accesos no autorizados. Mediante las cajas de texto allí situados se introduce el identificador de usuario y la clave para realizar el proceso de autenticación basado en el conjunto de usuarios autorizados del sistema. En caso de cancelar el proceso de autenticación se muestra un mensaje de confirmación y se procede a salir completamente de la aplicación.

4.2 Pantalla Principal

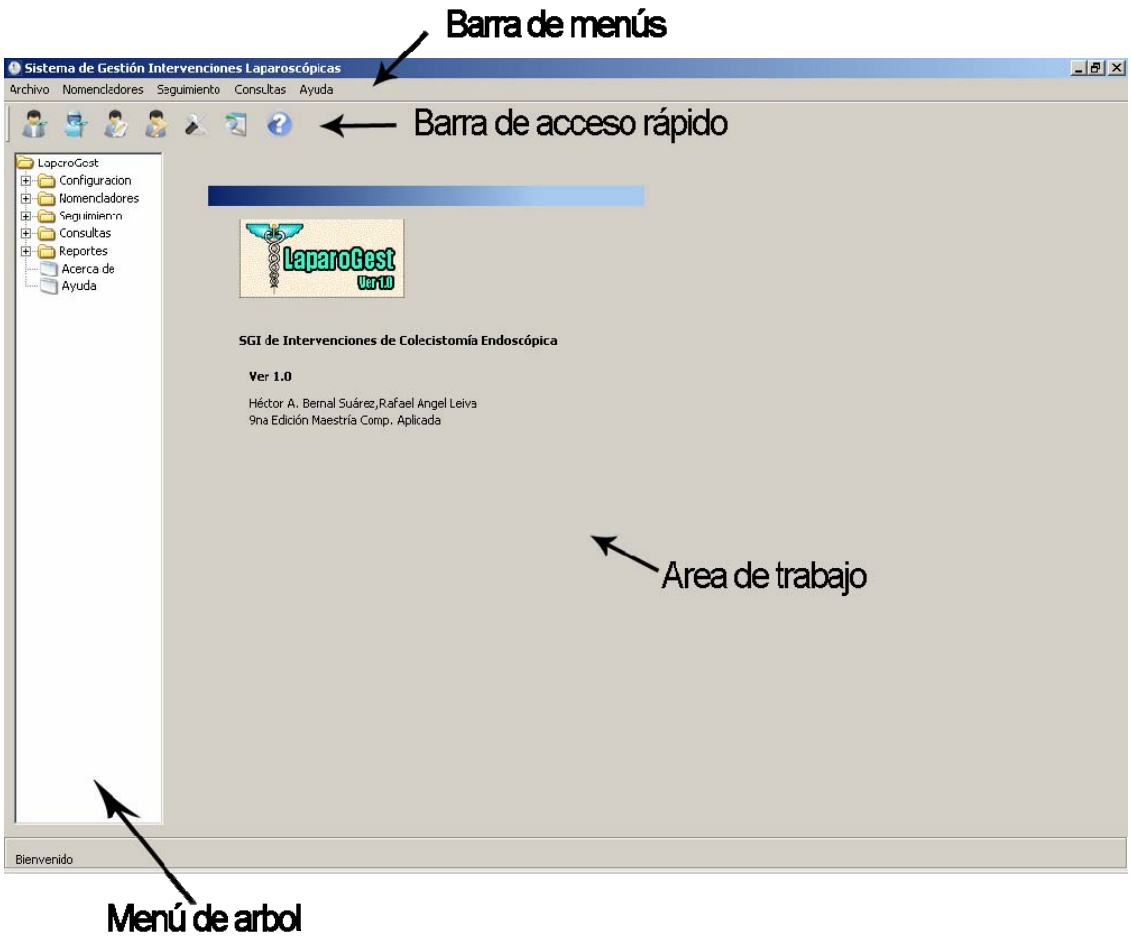

Fig. 18 Pantalla Principal

La pantalla principal se activa al concluir el proceso de autenticación de usuarios. El objetivo principal de esta pantalla es servir de punto de comienzo de la aplicación y contenedor de la mayoría de las formas, esta dividida en 4 secciones principales:

- Barra de menús: permite el acceso mediante menús textuales a la totalidad de componentes del sistema.
- Barra de acceso rápido: permite el acceso mediante iconos a los elementos de uso mas frecuente
- Menú de árbol: permite el acceso mediante una estructura jerárquica arbórea a las diferentes pantallas del sistema, a su vez permite invocar pantallas previamente abiertas y solapadas por la forma actual.
- Área de trabajo: es el contenedor de las formas de trabajo, permite visualizar y gestionar las pantallas en un formato MDI.

.

4.3 Nomencladores

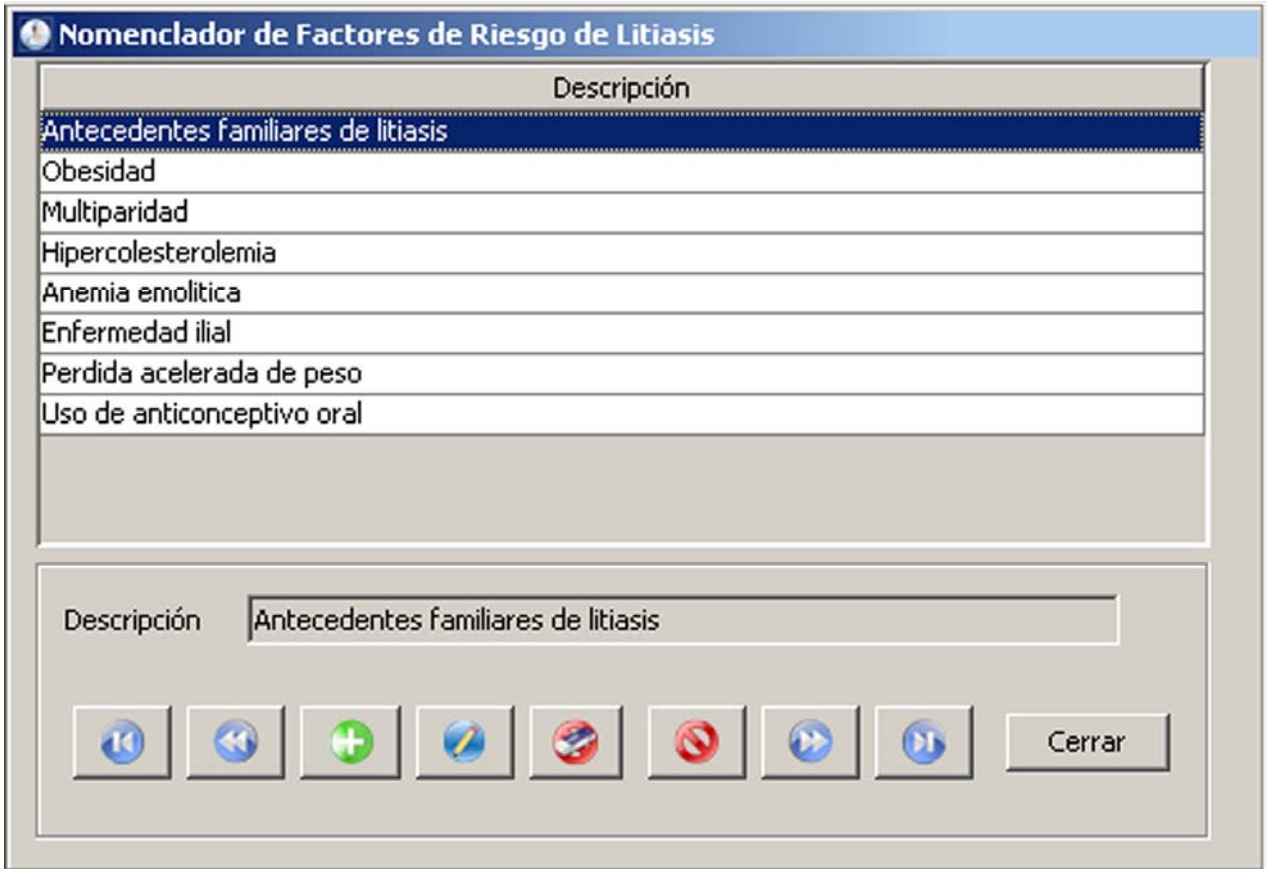

Fig. 19 Vista Nomencladores

Las pantallas de gestión de nomencladores responden al objetivo de realizar operaciones de CRUD (create, read, update, delete) sobre los nomencladores de valores del negocio. La totalidad de las pantallas clasificadas como nomencladores responden a un diseño de interfaz homogéneo basado en iconografía y uso de tooltips que se mantiene para el resto de la aplicación.

Fig. 20 Interfaz e iconografía

Primer Registro: Sitúa el cursor en el primer registro introducido

Registro Anterior: Sitúa el cursor en el registro anterior

Adicionar Registro: Activa la funcionalidad de agregar registro nuevo

Modificar Registro: Permite modificar registro seleccionado

Eliminar Registro: Elimina registro seleccionado (emite un mensaje de confirmación)

Cancelar Operación: Cancela operación en curso (valido para adición y modificación no confirmadas)

Próximo Registro: Sitúa el cursor en el próximo registro

Ultimo Registro: Sitúa el cursor en el último registro introducido

Por medio del menú Nomencladores accesible desde la pantalla principal se puede acceder a 12 pantallas de nomencladores que se desplegarán en el área de trabajo.

- **Aspectos Técnicos de Interés Quirúrgico:** Gestión de nomenclador de Aspectos Técnicos de Interés Quirúrgico a usar durante la etapa de seguimiento Transoperatorio.
- **Causas de Conversión**: Gestión de nomenclador de Causas de Conversión a usar durante la etapa de seguimiento Transoperatorio.
- **Causas de Intervención:** Gestión de nomenclador de Causas de Intervención a usar durante la etapa de seguimiento Transoperatorio.
- **Causas de Muerte:** Gestión de nomenclador de Causas de Muerte a usar durante la etapa de seguimiento Postoperatorio.
- **Antecedentes Patológicos:** Gestión de nomenclador de Antecedentes Patológicos a usar durante la etapa de seguimiento Preoperatorio.
- **Complicaciones:** Gestión de nomenclador de Complicaciones a usar durante la etapa de seguimiento Transoperatorio o Postoperatorio.
- **Diagnósticos:** Gestión de nomenclador de Complicaciones a usar durante las etapas de seguimiento Preoperatorio, Transoperatorio o Postoperatorio.
- **Factores de riesgo de Litiasis:** Gestión de nomenclador de Complicaciones a usar durante la etapa de seguimiento Preoperatorio.
- **Hallazgos Quirúrgicos:** Gestión de nomenclador de Hallazgos Quirúrgicos a usar durante la etapa de seguimiento Transoperatorio.
- **Otras Operaciones:** Gestión de nomenclador de Otras Operaciones a usar durante las etapas de seguimiento Preoperatorio, Transoperatorio o Postoperatorio.
- **Procederes Diagnóstico Transoperatorio:** Gestión de nomenclador de Procederes Diagnóstico Transoperatorio a usar durante la etapa de seguimiento Transoperatorio.

4.4 Pantalla Información de Pacientes

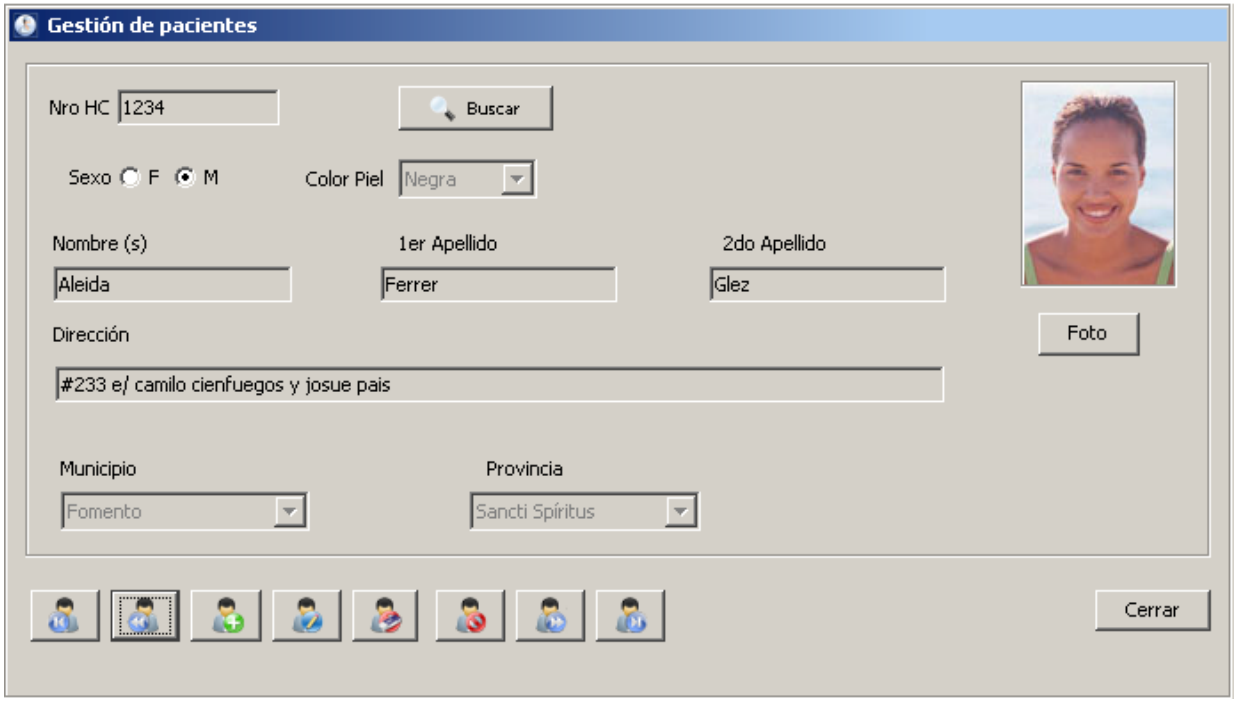

Fig. 21 Vista gestión de pacientes

Mediante el uso del menú Seguimiento o el icono de la barra de acceso rápido se puede acceder la pantalla de Gestión de pacientes que se desplegara en el área de trabajo de la aplicación. La pantalla de gestión de pacientes se emplea para captar los datos básicos de un paciente remitido a consulta así como su posterior modificación o eliminación. Constituye de esta manera el primer contacto de la información de paciente con el sistema.

4.5 Pantalla Seguimiento

La pantalla de Seguimiento permite realizar el proceso de negocio más importante de la aplicación que consiste en el seguimiento del paciente a través de todo el proceso de atención en la especialidad de Colecistectomía Laparoscópica. En la parte superior de la pantalla se selecciona el paciente deseado al teclear su número de Historia Clínica en la caja de texto etiquetada como HC, en caso de no existir dicho paciente se muestra un mensaje de error y la opción de proceder a introducirlo mediante la pantalla de Gestión de Pacientes, en caso de existir se muestran sus datos personales en la parte superior y los datos correspondientes a las diferentes etapas del seguimiento en las pestañas localizadas inmediatamente debajo.

4.5.1 Pestaña Preoperatorio

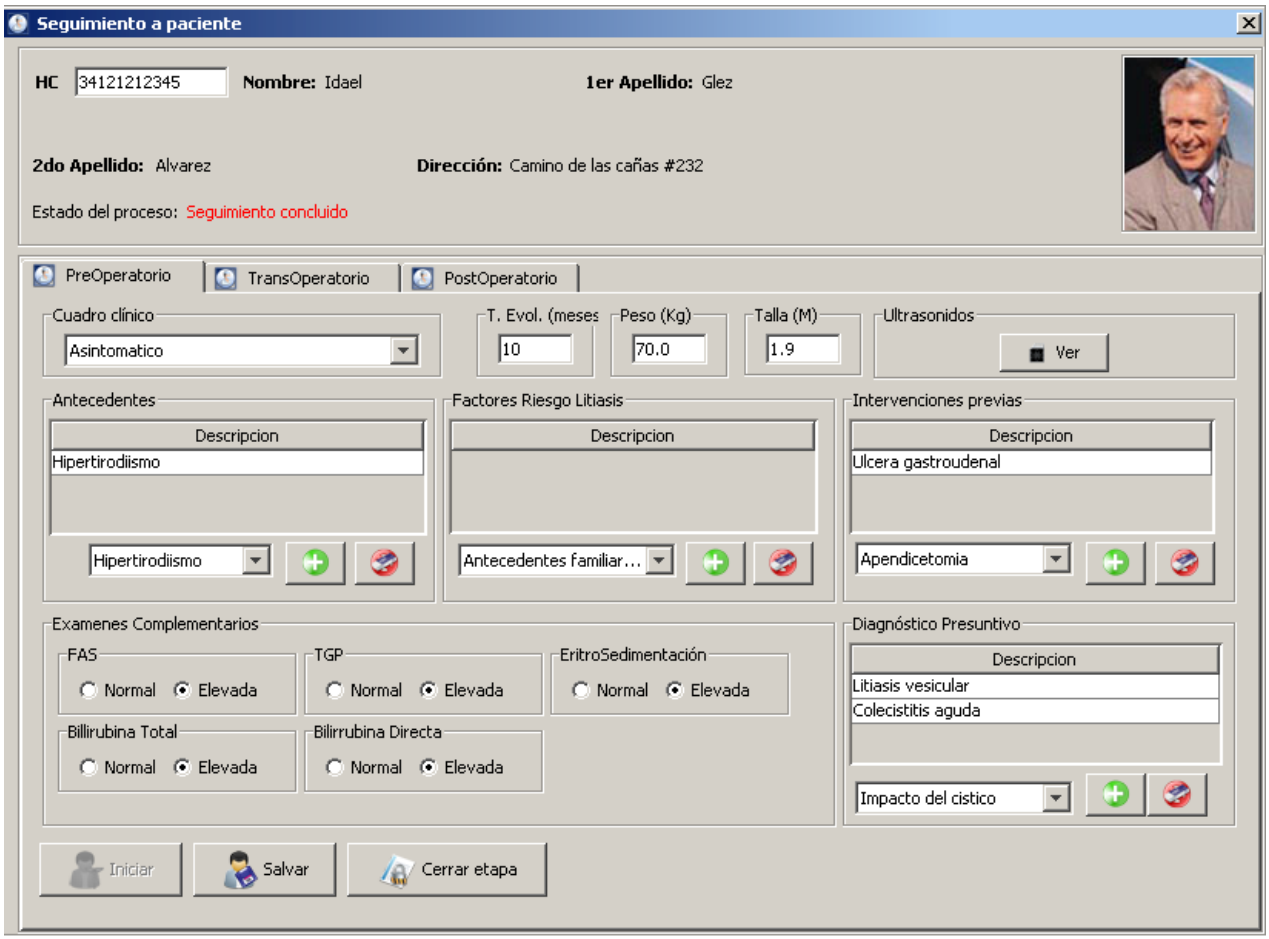

Fig. 22 Vista Seguimiento Pestaña Preoperatorio

En la pestaña de **Preoperatorio** perteneciente a la pantalla de Seguimiento se realiza la gestión de los datos correspondientes a dicha etapa del seguimiento, entre dichos datos se encuentran antecedentes patológicos, factores de riesgo, intervenciones quirúrgicas previas, exámenes, ultrasonidos realizados así como el diagnóstico presuntivo.

4.5.2 Pestaña Transoperatorio

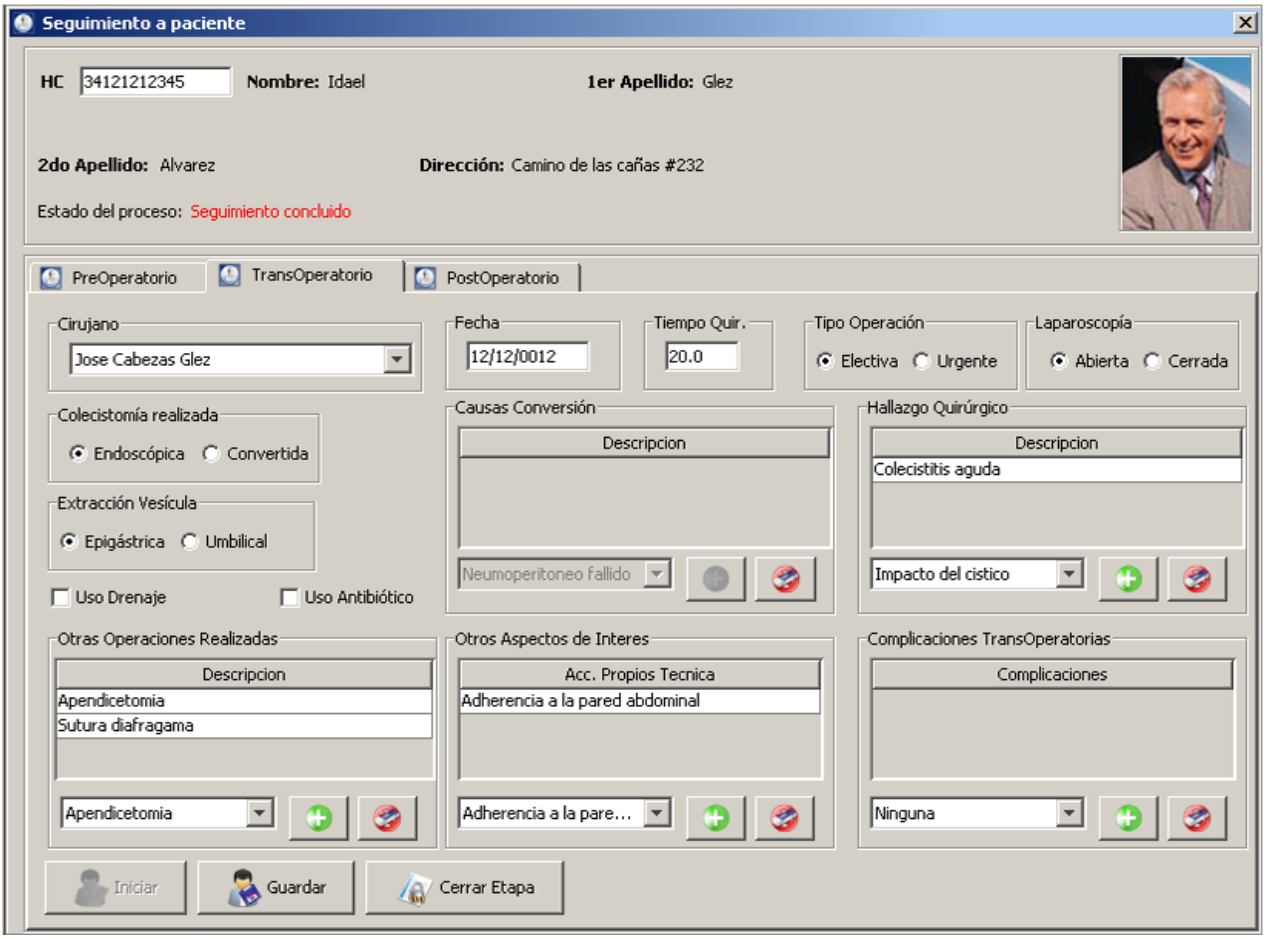

Fig. 23 Vista Seguimiento Pestaña TransOperatorio

En la pestaña de **Transoperatorio** perteneciente a la pantalla de Seguimiento se gestiona la información correspondiente a la etapa que describe el acto quirúrgico en si. Luego de iniciada como tal dicha etapa se procede a introducir datos acerca del cirujano que realiza el acto quirúrgico, la fecha en que se realiza, el tiempo empleado, el tipo de operación, la colecistectomía realizada, el tipo de laparoscopia, el tipo de extracción de vesícula, uso de drenaje, de antibiótico, el diagnóstico quirúrgico al que se llega luego del acto etc.

4.5.3 Pestaña Postoperatorio

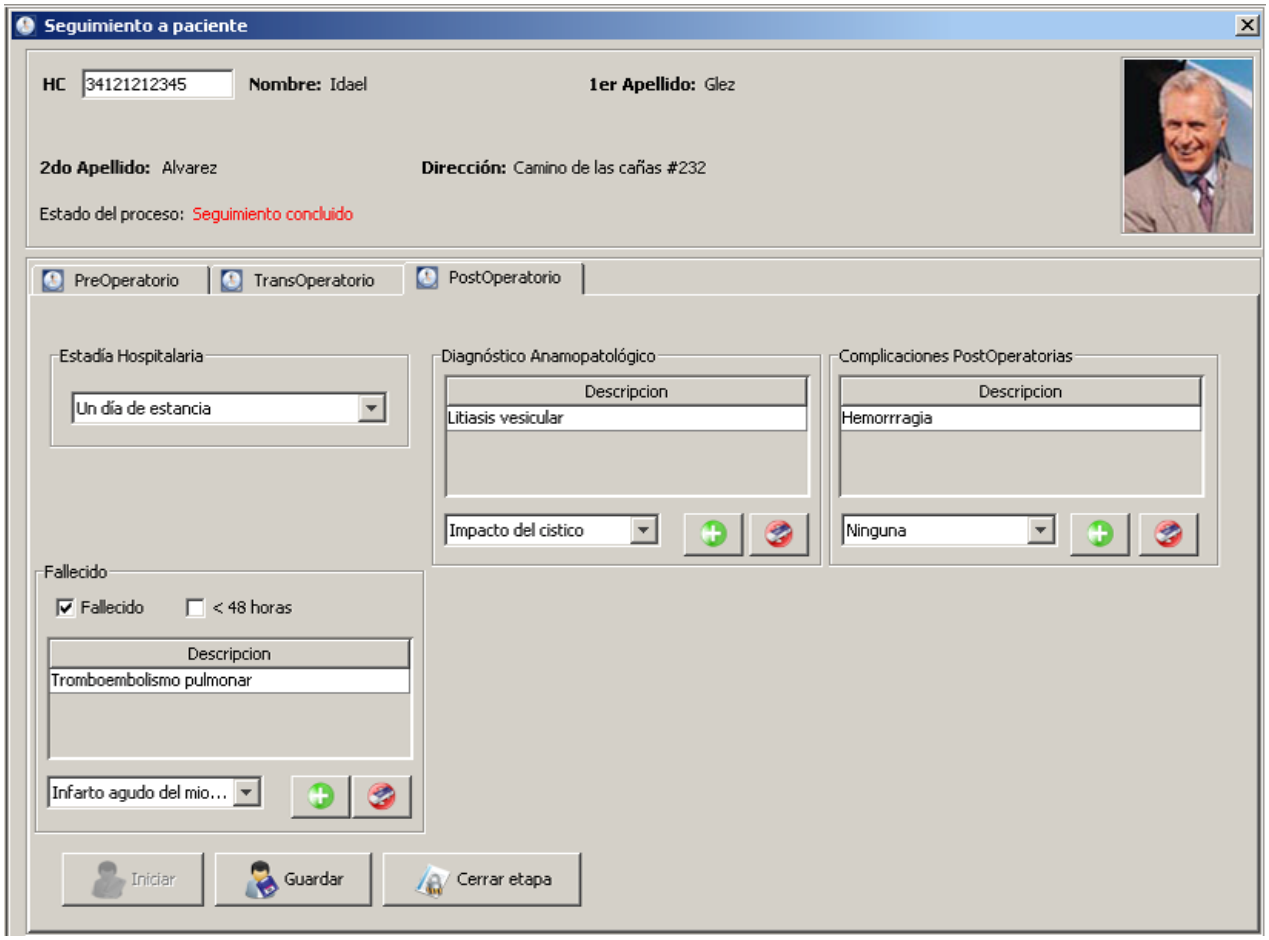

Fig. 24 Vista Seguimiento Pestaña Postoperatorio

En la pestaña **Postoperatorio** de la pantalla de Seguimiento se gestiona la información correspondiente a la etapa que transcurre desde la finalización del acto quirúrgico hasta el alta hospitalaria del paciente, entre estos se encuentra la cuantía de la estadía hospitalaria, las complicaciones presentadas en la etapa recuperatoria, el diagnóstico anamopatológico, las causas de fallecimiento en caso de proceder este, etc.

4.6 Pantalla Generador de Consultas

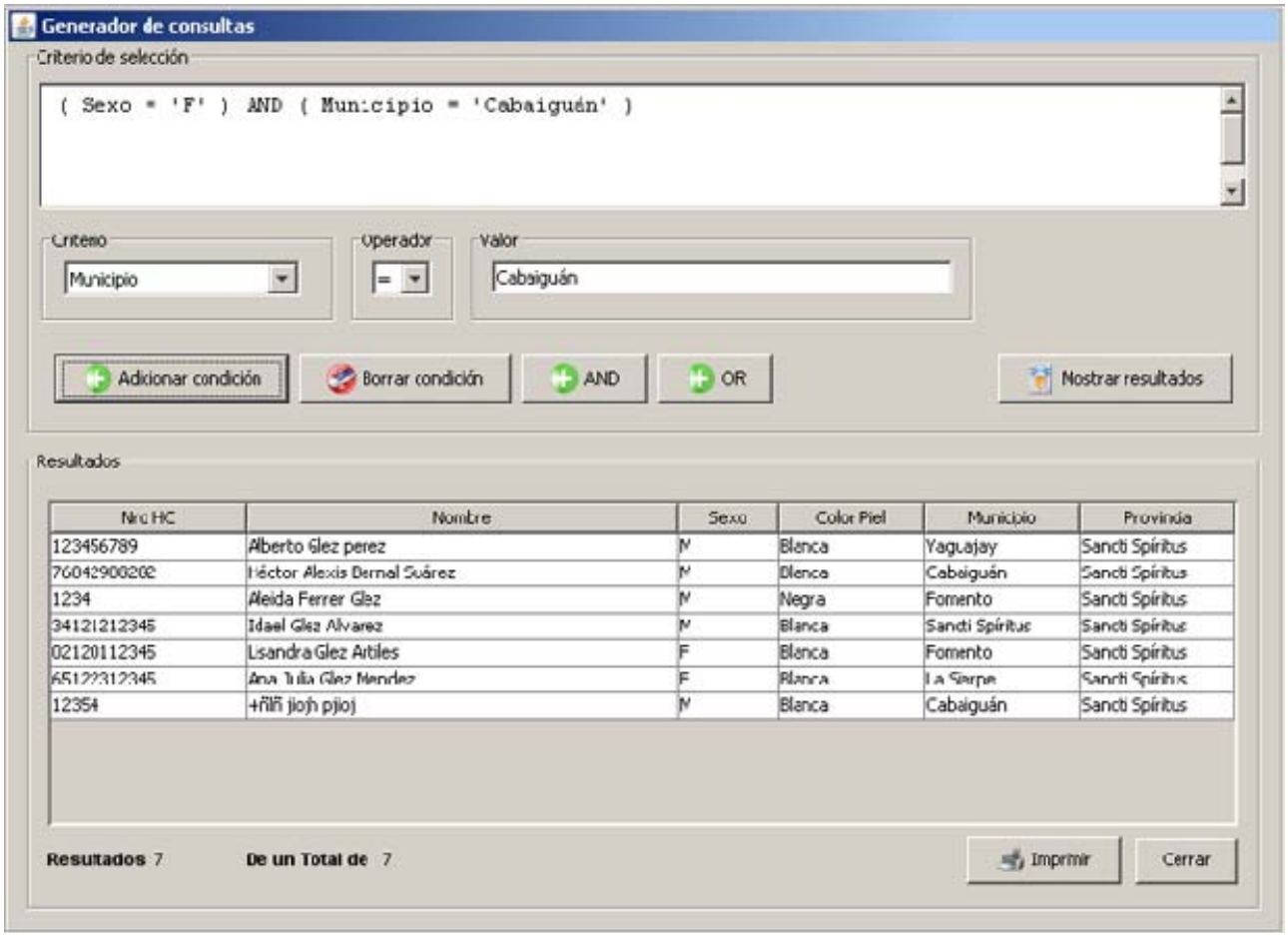

Fig. 25 Vista Generador de consultas

Mediante el uso del menú Consultas o el icono de la barra de acceso rápido se puede acceder a la pantalla Generador de consultas que se despliega en el área de trabajo de la aplicación. Debido a la dinámica de las necesidades de información en la rama hospitalaria se hace necesario implementar una funcionalidad que permitiera al usuario generar consultas sobre la información almacenada adaptables a un entorno cambiante pero restringido al conjunto de valores alimentados en la gestión de la información. Para lograr dicho objetivo se diseña una pantalla que mediante el uso de de criterios, operadores y valores genera criterios compuestos basados en postulados lógicos pero sintácticamente comprensibles para el usuario de perfil no informático. Posteriormente dichas consultas se traducen a JPQL y se devuelve el subconjunto de registros que cumplen con el criterio introducido. Los registros devueltos se pueden llevar a formato impreso en una salida que muestra el criterio formado para su obtención, los datos básicos de los pacientes así como la relación que tiene con respecto al total almacenado.

4.7 Reportes

El reporte principal del sistema lo constituye el resumen del seguimiento del paciente que se genera al finalizar este y resume la totalidad del proceso en un impreso ajustado al formato oficial a la vez que introduce mejoras sustanciales al empleado en el proceso previo a la automatización.

4.8 Ayuda

Fig. 26 Ventana Ayuda

Mediante el uso del menú de Ayuda, el icono de la barra de acceso rápido o los botones presentes a la derecha de los conjuntos de botones en las diferentes pantallas se puede acceder a la pantalla de **Ayuda del sistema** la cual se despliega como una ventana flotante no modal fuera del área de trabajo de la aplicación. Mediante el uso de un menú de árbol situado a la derecha de la pantalla se puede seleccionar el tema de ayuda deseado que se visualiza en el panel de la parte derecha.

4.9 Criterio de especialistas posterior a la etapa de prueba del sistema

Con vistas a obtener indicadores de satisfacción de los usuarios con respecto a la solución informática implementada se aplicó una encuesta (ver Anexo 3) basada en las debilidades halladas en el proceso de modelado del negocio e ingeniería de procesos a dos usuarios del sistema (cirujanos de la especialidad de Laparoscopía en la provincia). Mediante el uso de dicho instrumento se trata de poner de manifiesto las fortalezas y debilidades encontradas luego de procesar mediante la aplicación aproximadamente 300 pacientes del conjunto volcado a impresos.

Se aplico la encuesta a los dos especialistas implicados en el proceso:

- Dr Rafael Ángel Leiva, Cirujano Especialista 2 Grado, HPCQ “Camilo Cienfuegos” Sancti Spíritus
- Dr Roberto Cabezas Fdez, Cirujano Especialista 2 Grado, HPCQ “Camilo Cienfuegos” Sancti Spíritus

De las interrogantes aplicadas se obtienen resultados que reflejan la opinión de los implicados con respecto a:

1. Ambos coinciden en señalar como muy positivo la automatización del proceso.
2. Uno de los usuarios considera como muy positivo el cambio de formato impreso a formato digital, el otro usuario manifiesta no estar seguro acerca de la factibilidad del cambio.
3. Ambos coinciden en señalar que la confiabilidad de la información almacenada mejoró sustancialmente.
4. Ambos coinciden en señalar que el tiempo de acceso, búsqueda y consulta de información mejoró sustancialmente.
5. El volumen de información que obtiene con la herramienta comparado con el obtenido mediante procesamiento manual es calificado por ambos usuarios como sustancialmente mayor.
6. Uno de los usuarios encuestados considera que la herramienta propuesta contribuye a mejorar el proceso de toma de decisiones en su especialidad, el otro manifiesta no tener una opinión formada al respecto.
7. Los usuarios señalan como aspectos positivos de cara al proceso la facilidad de manejo de la herramienta, la visión de conjunto de la información, la rapidez de búsqueda, la posibilidad de obtener información histórica almacenada, la posibilidad de crear criterios de búsqueda.
8. Los usuarios señalan como aspectos susceptibles a mejorar el manejo de la imagenología médica y la posibilidad de obtener salidas parciales del seguimiento.

CONCLUSIONES

Como resultado de este trabajo se concluye:

1. Se realizó un estudio de los marcos de trabajo en Java y se procedió a la selección de *Swing* para la capa de presentación, *JPA* para la capa de persistencia de datos y capa de dominio, *Beans Binding como capa de enlace* y *Jasper Report* para la generación de informes.

2. Se definieron los procesos de negocio relacionados con el seguimiento al paciente de la especialidad de Colecistectomía Laparoscópica.

3. Se realizó un estudio del estado del arte en la rama de automatización del seguimiento a pacientes de la especialidad y se llego a la conclusión de que existen amplias posibilidades de automatización de procesos en la rama.

4. Se definió la arquitectura de la aplicación basada en 4 capas: persistencia, dominio, enlace y presentación lo que permitió facilidad de implementación y separación del código dando la posibilidad de hacer una aplicación con componentes reusables y extensibles

5. Se implementó la aplicación utilizando marcos de trabajo en Java y se comprobó la efectividad de la puesta en práctica de una estrategia de trabajo y el diseño de una arquitectura de desarrollo que permitió la elaboración de una aplicación de escritorio con tecnologías modernas. Se sugiere, a la vez, la metodología que permita realizar futuros proyectos con herramientas similares.

RECOMENDACIONES

- Implementar la conversión y visualización de imágenes Dicom usando las librerías Dcm4che u otras disponibles en plataforma de código abierto.
- Implementar el seguimiento para el resto de las intervenciones quirúrgicas realizadas con laparoscopia en nuestro país.

BIBLIOGRAFIA

- Albin, S. (2003). *The Art of Software Architecture: Design methods and techniques.* New York: Wiley.
- Alur, D., Crupi, J. & Malks, D. (2003). *Core J2EE Patterns, Best Practices and Design Strategies* (Vol. 2nd): Prentice Hall.
- Arredondo, L. J. P., & (2006). Las Nuevas Tecnologías de la Información (NTIC) en la medicina: la Telemedicina en Cuba *RevistaeSalud.com, Vol 2 No 7 2006*.
- Azcárate, J. C. G. d. (2006). De la historia clínica a la historia de la salud electrónica [Electronic Version]. Retrieved 23/11/2010.
- Bauer, C. (2005). *Hibernate in Action* Greenwich: Manning Publications Co.
- Bauer, C. (2006). *Java Persistence with Hibernate* New York,: Manning Publications Co.
- Bertilsson, F. (2004). The power of table oriented programming. Retrieved 12/09/2010, 2010, from http://www.javaworld.com/channel_content/jw-core-index.html
- Booch, G., Rumbaugh, J. & Jacobson, I.,. (1999). *The UML Modeling Language User Guide*: Adisson-Wesley.
- Campo, M. Á. (2001). Guía de Iniciación al Lenguaje Java.
- Castillero, O. C. (2003). Guía Básica de Referencia Sobre Casos de Uso.
- Chisty, M. M. I. (2005). An Introduction to Java Annotations. Retrieved 12/12/2010, from http://www.developer.com/article.php/3556176
- Deitel, H. M. D. P. J. (2001). *Advanced Java 2 Platform HOW TO PROGRAM*.
- Desarrollo de la Cirugía Endoscópica en Cuba. (2007). Retrieved 11/11/2010, 2010, from www.cce.sld.cu/historia
- Eckstein, R. (2007). Java SE Application Design with MVC. Retrieved 12/10/2010, 2010, from http://www.oracle.com/technetwork/articles/javase/index-142890.html
- Fowler, A. (2008). A Swing Architecture Overview. Retrieved 06/10/2010, 2010, from http://www.oracle.com/technetwork/java/architecture-142923.html
- Frameworks. (2007). Retrieved 10/11/2010, 2010, from http://es.wikipedia.org/wiki/Framework
- Froufe, A. (1997). Tutorial de Java. Retrieved 19/05/2009, from http://www.ulpgc.es/otros/tutoriales/java/Intro/tabla.html
- García, S. (2009). *Hibernate*: Universidad Complutense de Madrid.

- García., R. F. S. (2006). La Informática Médica en Cuba. *Revista de Ciencias Médicas La Habana.* Retrieved 14/06/2010, 2010, from http://www.cpicmha.sld.cu/hab/vol6_2_00/hab070200.htm
- Heffelfinger, D. (2006). *Jasper Report for Java Developers*. Olton Birmingham: Packt Publishing.
- Krill, P. (2007). Desktop Java faces againt AJAX. Retrieved 23/11/2010, 2010, from http://www.javaworld.com/index.html
- Ledo, A. D. R. y. M. V. (2010). Informática en la salud pública cubana [Electronic Version]. *Rev Cubana Salud Pública Ciudad de La Habana* v.32 n.3 Retrieved 09/10/2010.
- Nilo, R. (2010). ColeSoft. In H. A. B. Suarez (Ed.).
- Nuñez, J. M. (2003). *Investigación de la plataforma J2EE y su aplicación práctica*.
- Original Java WhitePaper. from http://java.sun.com/people/jag/OriginalJavaWhitepaper.pdf
- Pacheco, D. H. (2006). *DICOMPRESOR: Software para la compresión de imágenes médicas basado en el algoritmo JPEG-LS.* Universidad Central de las Villas Marta Abreu, Sancta Clara.
- Padrón., L. J. (2007). “Principales resultados en la aplicación de Las Nuevas Tecnologías de la Información y las Comunicaciones en el campo de la medicina cubana”. RevistaeSalud.com. from http://www.revistaesalud.com/index.php/revistaesalud/article/viewArticle/144/385
- Richardson, C. (2006). *Pojos in Action Developing Enterprise Applications with Ligthweigth Frameworks*. Greenwich: Manning Publications Co.
- Rodríguez, A. P. (2010). *EnerguX Control de Portadores Energéticos.* Universidad Central de las Villas Marta Abreu, Santa Clara.
- Soto, R. I. (2006). Frameworks de persistencia en Java: Universidad de Alicante.
- Weiss, M. (2009). *Data Structures and Problem Solving Using Java.*: Addison Wesley.
- Zukowski, J. (2006). *Programación Java 2 J2SE 1.4*. Volumen 1 Ciudad de la Habana: Editorial Felix Varela.
- Zukowski, J. (2006). *Programación Java 2 J2SE 1.4*. Volumen 2 Ciudad de la Habana: Editorial Felix Varela.
- Zukowski, J. (2006). *Programación Java 2 J2SE 1.4*. Volumen 3 Ciudad de la Habana: Editorial Felix Varela.

ANEXOS

Anexo 1 REQUISITOS NO FUNCIONALES

Requerimientos de Usabilidad

RNF 1. Los usuarios del sistema deben tener conocimientos mínimos de operación de microcomputadoras.

Requerimientos de Confiabilidad

RNF 1 La información debe estar disponible en todo momento. Es importante garantizar este acceso debido a la consistencia y actualización que se requiere tenga la información.

RNF 2 La información manejada por el sistema será objeto de cuidadosa protección contra la corrupción y estados inconsistentes.

RNF 3 Prevenir posibles fallos y recuperarse ante ellos.

RNF 4 En caso de pérdida del control de la aplicación se garantizará que la información anterior en la base de datos no se altere, y solamente se pierda la que estaba siendo actualizada en el proceso en ejecución.

RNF 5 Una vez implantado el sistema será responsabilidad del administrador de bases de datos realizar salvas periódicas de la base de datos en algún medio de salva apropiado que garantice la recuperación de la información en caso de avería del equipo, ataques de virus, etc.…

Requerimientos de Desempeño

En esta sección se hace referencia a las capacidades de desempeño que tiene el sistema basado en los requerimientos siguientes.

RNF 1 Tiempo de respuesta para una transacción

El tiempo de respuesta para una transacción es de 5 seg. siendo 10 seg. el tiempo máximo de espera para una respuesta del sistema en condiciones normales.

RNF 2 Rendimiento

El rendimiento esta a menos de 5/seg.

RNF3 Utilización de recursos.

En cuanto a la utilización de recursos tenemos que necesitará 256 MB de memoria y no excederá los 20 MB de capacidad en disco.

Restricciones de diseño

RNF 1 Lenguaje de Programación y gestor de Base de Datos

El lenguaje de programación a utilizar es Java por las grandes posibilidades que presenta para la programación y como gestor de BD PostgreSQL debido a sus potencialidades y compatibilidad con el lenguaje antes mencionado.

RNF 2 Proceso de Software.

Se usará la metodología descrita por la empresa para todo el proceso de construcción y desarrollo del software usando como lenguaje de modelado UML para la representación gráfica del software.

RNF 3 Tecnologías de Desarrollo.
Como herramientas de desarrollo para el sistema se usará NetBeans IDE para la programación y Adobe Photoshop para el diseño de las imágenes que se requieran para el ambiente gráfico del sistema.

Requerimientos de Ayuda y Documentación de usuario en línea

RNF 1 Manual de Usuario: Se elaborará un Manual de Usuario para el producto, en el cual se explicará como operarlo.

RNF 2 Ayuda: Se desarrollará un sistema de ayuda que facilite la asimilación y explotación correcta del sistema.

Adquisición de componentes.

No procede

Requerimientos de Interfaces

RNF 1 Las interfaces con el usuario deben tener un diseño estético sencillo, evitando el uso excesivo de colores e imágenes.

RNF 2 Las pantallas de trabajo serán semejantes; con el mismo esquema de colores y una interfaz de componentes homogénea lo cual permitirá al usuario una rápida apropiación de la metodología de uso de dicha interfaz.

RNF 3 Muy legible y de fácil entendimiento.

Interfaces de hardware

No procede

Interfaces de software

No existen interfaces de software a ningún otro componente del sistema de software.

Interfaces de comunicación

No existen interfaces de comunicación con ningún otro sistema o dispositivo.

Requerimientos de Legalidad, derechos de autor y otros avisos.

RNF 1 El sistema debe estar actualizado y cumplir con las leyes vigentes que rigen la informática en nuestro país, contemplando en el menor tiempo posible todas las modificaciones que afecten su funcionamiento.

RNF 2 El sistema estará regido por las leyes de derechos de autor vigentes en nuestro país.

Requerimientos de Seguridad

RNF 1 Disponer de un mecanismo de seguridad basado en la Autenticación y Autorización al cliente.

Anexo # 3 ENCUESTA A ESPECIALISTAS

Luego de hacer uso de la herramienta **LaparaGest** para el seguimiento a los pacientes de su especialidad quisiéramos que por favor dedicara unos minutos de su tiempo a responder las siguientes interrogantes que pudieran sernos de utilidad para mejorar nuestro trabajo:

Considera positivo la automatización del proceso de seguimiento:
Muy positivo Positivo Relativamente positivo No estoy seguro Negativo

Considera positivo el cambio de formato impreso a un soporte digital:
Si No estoy seguro No

La confiabilidad de la información puede calificarse como que:
Mejoró Se mantiene igual Empeoró

El tiempo de acceso, búsqueda y consulta de información:
Mejoró sustancialmente Mejoro algo Mejoró Se mantiene igual Empeoró

El volumen de información que obtiene con la herramienta comparado con el obtenido mediante procesamiento manual se puede calificar como:
Sustancialmente mayor Mayor Igual Menor Mucho menor

Considera que la herramienta propuesta contribuye a mejorar el proceso de toma de decisiones en su especialidad:
Muchísimo Mucho Ni mucho ni poco Poco Nada

Enumere 5 aspectos en los que el uso del software mejora la eficiencia del proceso.

Enumere 5 aspectos que considera son susceptibles a mejorar en próximas versiones del producto:

Anexo # 2: DIAGRAMA ENTIDAD – RELACIÓN EXTENDIDO