UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
FACULTAD 8

Análisis, Diseño e Implementación del módulo

Experticias Criminalísticas

del Sistema de Investigación e Información Policial

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

Autores:
Maikel Pereira Ojeda
Yudanis Gago Martinez

Tutor:
Ing. Yadiel Ramos Martinez

Ciudad de la Habana, Junio de 2008

Declaración de autoría

Declaro que soy el único autor de este trabajo y autorizo al _____ de la Universidad de las Ciencias Informáticas; así como a dicho centro para que hagan el uso que estimen pertinente con este trabajo.

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

Maikel Pereira Ojeda	**Yudanis Gago Martínez**	**Yadiel Ramos Rodríguez**
_____	_____	_____
Firma del Autor	Firma del Autor	Firma del Tutor

Agradecimientos

A la Revolución, a Fidel y a la UCI por darnos la oportunidad de formarnos como profesionales.

A nuestro tutor, Yadiel por hacer un hueco siempre en su apretada agenda.

A nuestras novias Mayliuvis y Lili por ser tan comprensivas.

A todos nuestros compañeros de aula que de una forma u otra formaron parte de nuestros momentos difíciles y alegres, posibilitando que pudiéramos llegar a este día: Armando, Oliday, Alexis, Yendy y Enrique, a Martín que aunque ya no esté cerca de nosotros, siempre será un gran amigo, a Jorge Amado, Humberto y al resto de nuestro grupo de cuarto y quinto año, a los de los antiguos grupos también.

A Yeleine, Yaimara, Yunexis, Reinaldo, Raciel, Lara, Luis del equipo de Criminalística, por brindarnos la mano en el momento oportuno y por ser los verdaderos artífices del módulo Experticias Criminalísticas.

Pero especialmente a nuestros padres que siempre nos apoyaron en todo y que hoy más que nunca se merecen que se les agradezca por saber guiarnos por el camino correcto.

Dedicatoria

A nuestros padres, para darles otro motivo que los haga sentirse orgullosos de sus hijos.

Resumen

Actualmente en el Cuerpo de Investigaciones Científicas, Penales y Criminalísticas de Venezuela, una de las instituciones más importantes en la investigación del delito en este país, se utiliza el Sistema Integrado de Información Policial para gestionar la información que se maneja digitalmente. Éste resulta ineficiente, de tecnología obsoleta y no cumple con las necesidades de velocidad y buen tratamiento de la información, lo que afecta la correcta realización de los procesos que se ejecutan a diario. Como parte de un contrato firmado bilateralmente con nuestro país, se pretende desarrollar un sistema que cumpla con los requerimientos obtenidos del estudio del funcionamiento de la institución. El presente trabajo propone el análisis, diseño e implementación de una porción de *software* que cumpla con los requisitos relacionados con las experticias de la Coordinación Nacional de Criminalística (CNC), coordinación del Cuerpo de Investigaciones Científicas, Penales y Criminalísticas de Venezuela que no es cubierta por el actual software de gestión de información. El resultado obtenido debe mejorar los procesos de la CNC, e incluye una explicación vasta de las actividades realizadas, y los artefactos generados en cada una de las partes del desarrollo de un software.

Contenido

Contenido

Contenido

Introducción

En los últimos años América Latina ha experimentado un auge de revoluciones de corte social y político, Venezuela ha sido la muestra más trascendente de este proceso. La situación creciente de hechos punibles en todo el territorio, puestos al descubierto por el gobierno bolivariano, es el resultado de una buena cantidad de años sufriendo la desatención de los gobiernos precedentes.

Venezuela cuenta actualmente con varios grupos de investigación contra el crimen y la corrupción que los antiguos gobiernos habían dejado, y de ellos el más importante es el Cuerpo de Investigaciones Científicas, Penales y Criminalísticas(CICPC); de ahí la necesidad de dicha institución de dar una respuesta rápida y confiable a todo hecho cometido fuera de la ley en dicho país, de manera que su compleja organización interna no dificulte el tratamiento de la información y el accionar de la prestigiosa entidad en general.

El CICPC es la institución que se encarga a nivel nacional de desarrollar la investigación científica del delito, y cuenta con una aplicación informática que se conoce como Sistema Integrado de Información Policial, que posee una tecnología obsoleta y no da soporte a todos los procesos que se llevan a cabo a diario en la institución, factores que frenan su correcto funcionamiento y acarrean ineficiencia en la primordial tarea de esclarecer los hechos delictivos. El CICPC y sus coordinaciones se ven en la necesidad de hacerse de un sistema de gestión competitivo, con tecnología de punta, que almacene y maneje la información generada.

Dicha institución se compone de varias coordinaciones (nivel de la estructura organizativa en el que se delegan procesos especializados) como pueden ser la Coordinación Nacional de Investigaciones Forenses, la Coordinación Nacional de Investigaciones Penales, entre otras. Una de ellas es la Coordinación Nacional de Criminalística, la cual tiene como función fundamental planificar, coordinar y dirigir todos los procesos técnico-científicos de las investigaciones. Esta Coordinación es la encargada de definir las políticas, normas, procedimientos y planes estratégicos que se implementan para el correcto trabajo investigativo, además de diseñar las estrategias que garanticen la actualización técnico-científica y definir métodos de estandarización en criminalística.

Actualmente en la Coordinación Nacional de Criminalística existe una situación problémica constituida por la lentitud en la fluidez de la información, la poca diversidad, la poca capacidad que brinda el sistema de gestión de información actual, y que la información que se maneja en buena parte de los procesos es física y se deteriora con facilidad. La comunicación de solicitudes entre entidades es

telefónica y a veces debe esperarse el documento físico. La recuperación y consulta de información es difícil y el filtrado y procesamiento de grandes volúmenes de información casi imposible. En casos específicos de áreas que componen esta Coordinación existe falta de seguridad y control en los datos que se registran (1).

En el marco de cooperación del ALBA, Cuba y Venezuela firmaron un contrato con el objetivo de automatizar el CICPC. Como parte de la planificación general, se pretende desarrollar el nuevo Sistema de Investigación e Información Policial (SIIPOL), tomando como referencia el sistema existente, que incremente las posibilidades de uso, gestión de la información, con tiempos de respuesta mejores, y que cubra también los procesos de la CNC.

Posterior al análisis del funcionamiento de la institución y de sus procesos de negocio se han detectado una serie de requerimientos funcionales y no funcionales con los cuales el nuevo sistema debe cumplir para facilitar la sistematización del trabajo en las dependencias del CICPC, y se ha modelado un sistema formado a su vez por subsistemas, y estos por módulos, uno de los cuales es el módulo Experticias Criminalísticas, que ha de ofrecer las funcionalidades para soportar todos los flujos de procesos realizados en esta área.

Por tanto el **problema científico** puede quedar definido a modo de interrogante de la siguiente forma: ¿Cómo garantizar el cumplimiento de los requisitos funcionales y no funcionales asociados al módulo Experticias Criminalísticas del SIIPOL?

El **objeto de estudio** fijado es el proceso de desarrollo del *software* de gestión de información SIIPOL, y el **campo de acción** en que se enmarca el trabajo es el Análisis, Diseño e Implementación del módulo Experticias Criminalísticas del SIIPOL.

El **objetivo general** de este trabajo es: Analizar, diseñar e implementar un módulo para el subsistema Investigación Criminalística que satisfaga los requisitos funcionales y no funcionales obtenidos como resultado de la aplicación de la Ingeniería de Requerimientos a los procesos de la Coordinación Nacional de Criminalística relacionados con las Experticias.

De una manera más detallada, los **objetivos específicos** para la construcción del módulo son:

- Investigar las tendencias tecnológicas actuales y estándares más adoptados en la construcción de sistemas de gestión de información a nivel mundial.
- Investigar los resultados de la aplicación de patrones de diseño e implementación en proyectos de gestión de información de gran envergadura.

- Estudiar y valorar los resultados obtenidos por la ingeniería de requerimientos.
- Analizar, definir, estructurar y detallar mecanismos de diseño que garanticen el cumplimiento de las necesidades del módulo de software.
- Implementar el módulo diseñado respetando la arquitectura definida para el nuevo SIIPOL.
- Lograr la integración satisfactoria al subsistema Investigación Criminalística y al SIIPOL del módulo implementado.

La **idea a defender** afirma que: Si se aplica correctamente la metodología de desarrollo RUP para el análisis, diseño e implementación del módulo Experticias Criminalísticas, entonces es posible elaborar un módulo de software de gestión que cumpla con los requisitos funcionales y no funcionales obtenidos como resultado de la Ingeniería de Requerimientos.

Para dar cumplimiento a estos objetivos se planificaron una serie de **tareas investigativas**, tales como:

- Investigar acerca de otras aplicaciones o soluciones similares y de los lenguajes y metodologías de desarrollo de software existentes, así como de sus *frameworks (marcos de trabajo)* y herramientas de desarrollo.
- Describir la metodología, herramientas y lenguaje a utilizar en el análisis, diseño e implementación del módulo de Experticias Criminalísticas de SIIPOL.
- Estudiar la propuesta de Arquitectura para el Sistema SIIPOL.
- Estudiar el modelo de Casos de Uso que da cumplimiento a los requisitos funcionales y no funcionales asociados al Módulo de Experticias Criminalísticas del SIIPOL.
- Estudiar detalladamente la especificación de los Casos de Uso del Sistema.
- Investigar la aplicación de Patrones de Diseño.
- Realizar diagramas de clases de diseño para cada Caso de Uso y diagramas de contrato.
- Aplicar los patrones de diseño para el refinamiento del modelo de diseño.
- Implementar los componentes necesarios que dan solución a los objetivos propuestos.
- Integrar la solución al sistema a la vez que se desarrolla.

Las expectativas de este trabajo son contribuir al procesamiento y gestión eficiente de la información de la Coordinación Nacional de Criminalística, así como ampliar las funcionalidades de la CNC,

aumentando también su eficiencia mediante el desarrollo de una herramienta que vincule el trabajo investigativo y científico del CICPC, e incluso automatizar procesos que no eran cubiertos por el antiguo sistema informático. Por otra parte, en el campo del conocimiento se espera formar a los desarrolladores en el perfil de software de gestión, acorde con las políticas de la institución a la que pertenecen, y sentar las bases del conocimiento para el desarrollo posterior de una aplicación que optimice el funcionamiento de la CNC, así como para la construcción de sistemas similares en otros contextos.

En este trabajo se aborda el análisis, diseño e implementación del módulo Experticias Criminalísticas del SIIPOL, partiendo de la necesidad de cumplir los requerimientos capturados previamente (que no son alcance de este documento), ajustándose a la línea base de la arquitectura definida anteriormente y aplicando las técnicas y la teoría estudiada. Para su mejor comprensión, está estructurado en tres capítulos, el primero enfoca la fundamentación del tema a tratar, haciendo un estudio teórico del contexto del desarrollo en el proyecto, el segundo capítulo recoge los principales resultados del trabajo en la práctica, dividido en dos de las tres partes fundamentales que conforman el objetivo del trabajo: analizar y diseñar un módulo de software; el tercer capítulo trata la implementación del mismo.

Capítulo 1 Fundamentación teórica

Con el presente capítulo se pretende lograr un basamento teórico sólido acerca de la situación actual que genera la idea de realizar este trabajo, del marco en que se desenvuelve, las tendencias actuales en el mundo para resolver problemas como éste y la definición de cómo llegar a su resolución. Para desempeñar la parte práctica con la calidad adecuada y tener un sistema de conocimientos que permita tomar las mejores decisiones en el transcurso de la producción, es preciso dedicar un espacio al estudio que a continuación se refleja.

1.1 Fundamentación del tema.

Esta sección trata de describir el entorno en el cual ha de desarrollarse el software, tratando de introducir conceptos y mostrar explicaciones acerca de los orígenes del tipo de trabajo a realizar, del estado actual del tema a nivel mundial y acerca de las causas que da origen al presente trabajo.

1.1.1 Sistema de Gestión de Información

Un Sistema de Gestión de Información puede ser definido como un conjunto de elementos que interactúan entre sí, con el fin de apoyar las actividades que se realizan en una organización o para automatizar los procesos de trabajo que se efectúan dentro de esta. Y un sistema de gestión de información policial, como dice el término, se restringe a automatizar los procesos en entidades policiales. La puesta en marcha de un Sistema de Gestión de Información puede proporcionar numerosos beneficios, tales como la optimización del tiempo que transcurre desde la búsqueda de documentos y la reducción drástica de los riesgos de pérdida del documento físico original. Se puede tener acceso concurrente a un documento. Mejora la atención a los clientes, tanto internos como externos, y al disponer de la información de forma centralizada y rápidamente accesible, es posible solucionar *online* cuestiones relacionadas con una consulta de un cliente; y por otra parte incrementa la satisfacción de los usuarios internos, por agilizar sus procesos de trabajo, y reduce los costes legales. Protege la documentación sensible o de alto valor de deterioro, que puede derivar elevados costes legales por incurrir en demandas y otros procesos legales. Aunque es necesario tener en cuenta que el éxito de los sistemas de Gestión de Información depende en una buena parte de las habilidades y la experiencia de la fuerza de trabajo y pueden llegar a ser difíciles de usar.

Una de las características más comunes de los sistemas de gestión policial es su confidencialidad; el desarrollo de este tipo de *software* no promueve la difusión de su documentación, por razones de

seguridad de la institución que lo solicita, por lo que la información accesible es escueta. Con el fin de fundamentar un estudio del tema tratado se realizó una búsqueda de documentación de *software* de este tipo, para tener un punto de partida al comenzar a desarrollar la propuesta. A continuación se hace referencia a algunos de los sistemas relacionados a la gestión policial y criminalística.

1.1.2 Sistemas de Gestión de Información Policial a nivel mundial

A nivel internacional los índices de temor e inseguridad de la población mundial se han incrementado, paralelamente a la desconfianza que las personas tienen hacia las instituciones policiales. Esta situación ha provocado la necesidad de realizar importantes reformas a las instituciones policiales; es por ello que en la actualidad el mundo es testigo de una creciente incorporación de sistemas de gestión de información en los cuerpos policiales existentes. La asociación de estos sistemas con las fuerzas que combaten a diario el delito, permite dar una respuesta inmediata y certera en el enfrentamiento a los males que ponen en peligro la seguridad social y el orden interior de cada país. Los sistemas de gestión de información policial permiten efectuar búsquedas ágiles y eficaces, por lo que el tiempo de respuesta para combatir los delitos es significativamente breve, ya que poseen numerosas funcionalidades que hacen que el trabajo que se realizaba de forma manual sea considerablemente rápido y efectivo.

Cada vez es más común presenciar naciones que adoptan este enfoque, las cuales se benefician con la cooperación entre los distintos órganos policiales al obtener cualquier información de interés para investigaciones policiales.

1.1.3 Cuerpo de Investigaciones Científicas, Penales y Criminalísticas

El Cuerpo de Investigaciones Científicas, Penales y Criminalísticas de la República Bolivariana de Venezuela (CICPC) es la organización encargada de garantizar que la investigación sobre los hechos delictivos se haga de forma eficiente, y se encarga de realizar todas las diligencias que le sean encomendadas por parte del Ministerio Público. Mantiene una estrecha colaboración con los demás órganos de seguridad ciudadana que permite compartir la información de los servicios de inteligencia, de tipos delictivos como narcotráfico, terrorismo internacional, desaparición de personas, movimiento de capitales ilícitos, delincuencia organizada y otros. Elabora políticas de prevención, orientación, publicidad, colaboración e información a fin de emplear medidas técnicas que permitan reducir y evitar la actividad delictiva. Colabora en la identificación, localización y aprehensión de ciudadanos extranjeros solicitados por otros países (1).

1.1.4 Coordinación Nacional de Criminalística. Actualidad

La CNC es la encargada de planificar, coordinar y dirigir todos los procesos técnico-científicos de las investigaciones. Actualmente en la Coordinación Nacional de Criminalística existe una situación problemática constituida por la lentitud en la fluidez de la información entre las diferentes áreas de trabajo del CICPC. Para realizar alguna experticia es necesario archivar sus detalles mediante libros de control, así como las de evidencias físicas asociadas al estudio. Además, para conocer información referente a datos específicos de experticias y evidencias físicas ya procesadas, es necesario hacerlo por vía telefónica y esperar por el resultado.

Esto trae consigo la falta de información actualizada, oportuna y fiable a las entidades de la dirección del CICPC. Para que los entes solicitantes de las experticias conozcan información sobre los análisis o sus resultados, tienen que esperar por la materialización de la investigación o el informe, para conocer las conclusiones y poder valorar su importancia. En casos específicos existe falta de seguridad y control en los datos que se registran. Otra problemática importante por la que atraviesa la Coordinación es que existen limitaciones en la diversidad de información que se requiere para la investigación de los hechos y en la calidad de uso de la que se almacena.

Por otra parte, está presente la imposibilidad de acceder y utilizar información de interés de otras organizaciones. Si se desea conocer alguna información relacionada con algún caso, esta tiene que ser canalizada a través del despacho que lleva la averiguación de dicho caso. En algunas áreas que componen esta Coordinación existe falta de seguridad y control en los datos que se manejan, debido a que los registros son llevados en un formato Microsoft Excel, lo que posibilita que los apuntes puedan ser modificados o alterados; no existe un supervisor o persona encargada del control y cuidado del registro de experticias, lo que origina grandes posibilidades de pérdida. Algunos de los despachos que componen esta Coordinación cuentan con la ayuda y el servicio de herramientas como el Sistema Automático de Identificación de Huellas Dactilares (AFIS) y el Sistema Integrado de Identificación Balística (IBIS); estos brindan un número considerable de servicios al generar información referente a evidencias, las cuales permiten contribuir al esclarecimiento de las investigaciones de hechos punibles (1).

1.1.4 Proyecto de modernización del CICPC

En el marco de la colaboración de los países del ALBA, se involucran Cuba y Venezuela en un proyecto de Modernización del CICPC que abarca fundamentalmente:

- Proyectos de transformación organizacional.
- Equipamiento con tecnologías de punta.
- Capacitación del personal de la Institución.
- Construcción de un software que maneje la gestión oportuna, ágil y eficaz de sus procesos.

Con el objetivo de dar cumplimiento a la planificación general del proyecto de cooperación, se pretende: desarrollar un nuevo sistema integrado de información policial, tomando como referencia el sistema existente SIIPOL, que incremente las posibilidades de uso, gestión de la información y con tiempos de respuesta mejores o al menos iguales que los del anterior SIIPOL (1).

Teniendo una visión global del funcionamiento de la organización, y habiendo llegado a un entendimiento con los clientes, el equipo de analistas del Proyecto CICPC fue el encargado de proporcionar la descripción de la propuesta del sistema, para llevar a cabo todos los procesos de negocio de manera automática. Como parte de esta propuesta se especificaron todas las funcionalidades que el sistema, dividido en subsistemas y módulos de trabajo, brindará a los usuarios, a través de casos de usos.

1.2 Estudio de sistemas de interés

Los sistemas que pueden servir de documentación y de referencia para comprender el contexto donde se desarrollan los procesos de gestión policial, podrían a su vez ser clasificados para su estudio en dos grupos los sistemas que se utilizan actualmente en el CICPC: los sistemas similares de gestión de información policial a nivel mundial y los que se desenvuelven en el contexto de producción similar al del equipo de desarrollo del SIIPOL. En esta sección se abordan cada uno de los que se consideraron representativos, el resto de la documentación que se encuentra disponible se anexa al documento.

1.2.1 Sistema Integrado de Información Policial

El Sistema Integrado de información es una aplicación informática que el Cuerpo de Investigaciones Científicas, Penales y Criminalísticas de la República Bolivariana de Venezuela utiliza para llevar a cabo las tareas fundamentales. Este sistema está desarrollado sobre una tecnología actualmente obsoleta y no cubre todos los procesos del CICPC; entre los procesos que no están cubiertos se encuentran los relacionados con Experticias Criminalísticas de la CNC.

1.2.2 Sistema Automático de Identificación de Huellas Dactilares

El Sistema Automático de Identificación de Huellas Dactilares (AFIS) es una herramienta computarizada que permite la identificación rápida y confiable de personas, al contar con una base de

datos, la cual es alimentada por los archivos tradicionales de identificación. Este sistema efectúa la comparación de impresiones digitales a través de un software de manera automática, permite el cotejo de huellas dactilares contra imágenes de casos no resueltos que estén almacenados con anterioridad. Además permite la captura de la imagen de la impresión digital (escaneo). El AFIS, en un tiempo relativamente corto, puede localizar una huella o latente solicitada u obtenida de un lugar de los hechos. Es suficiente con introducirla para que el sistema informe si existen antecedentes de ella en su memoria. En caso de que la localice, podrá complementar con información nominal e inclusive proporcionar una fotografía del presunto delincuente. Este sistema permite ahorrar tiempo en las actividades de localización de datos, realizar varias búsquedas de manera simultánea, optimizar el aprovechamiento de los recursos humanos, reducir importantes márgenes de error debido a la forma de la captura y alimentación de la base de datos (1).

1.2.3 Sistema Integrado de Identificación Balística

El Sistema Integrado de Identificación Balística (IBIS) es una herramienta utilizada en esta Coordinación para almacenar, en una base de datos, las características microscópicas de las evidencias relacionadas con proyectiles, armas de fuego y lugar de los hechos, así como la hora y la fecha. El IBIS, en su funcionamiento, compara automáticamente imágenes y sus características con las demás muestras que posea la base datos, y se obtiene en la Estación de Análisis de Firmas un reporte con aquellas que más se asemejen a la nueva muestra; de esta manera se pueden correlacionar varios casos y así lograr establecer, mediante cotejo macroscópico, si un arma de fuego se encuentra involucrada en diferentes hechos delictivos (1).

1.2.4 STEGPOL- Sistema Territorial de Emergencias y Gestión Policial

Es un Sistema de Información Geográfica con tecnología de punta sobre una Plataforma Nacional Común de Información aplicada al "Sistema de Emergencias Nacionales" y al "Sistema Territorial de Gestión Policial" que actualmente operan en Chile. Se caracteriza por ser un sistema con tecnología de punta aplicado al "control territorial de la gestión policial", el cual identifica en forma veraz y efectiva dónde geográficamente se están cometiendo o se han cometido actos delictuales a nivel territorial, apoyado por sistemas de información en línea desde el lugar de los hechos, que permitan la acción rápida y coordinada entre los diferentes actores encargados de la seguridad ciudadana a nivel de país.

Integra a las diferentes entidades, como Carabineros, Investigaciones, Ministerio del Interior y Municipios, entre otros, en una Plataforma Nacional Común de Información que permite el intercambio de datos y que sirve de apoyo a la gestión operacional regional o comunal, donde dichas Instituciones

estén interconectadas entre sí, en especial con diferentes Divisiones o Departamentos de Carabineros, tales como Jefaturas de Zona, Prefecturas, y otros, más el apoyo directo y coordinado entre Carabineros y el área de Seguridad de los Municipios. Vía la Intranet del Estado o vía comunicación en Banda Ancha - Internet se puede llegar a acceder a un Sistema Territorial de Emergencias y Gestión Policial (STEGPOL) montado en la Web con diferentes contraseñas de acceso restringido, tanto para los Usuarios Operadores encargados de ingresar o modificar datos en línea, como para aquellos que solo tengan acceso a consultar (2).

1.2.5 Sistema de Gestión de Penitenciaria

El SIGEP es un proyecto que está en etapa de desarrollo y que comenzó antes que SIIPOL; este proyecto se desarrolla en las mismas condiciones prácticamente que el proyecto CICPC. Tienen la similitud de que ambos son para instituciones del gobierno venezolano y revisten una importancia alta en el marco de las relaciones Cuba-Venezuela. SIGEP utiliza tecnologías similares a las escogidas para el SIIPOL desde las etapas iniciales del proyecto y el equipo tiene iguales características. Prisiones, como también se le llama a dicho proyecto, tiene el marco de desarrollo ideal para ser estudiado, y siempre puede constituir una fuente de información, si están de acuerdo en compartir sus experiencias.

1.3 Proceso de desarrollo

El objetivo de un proceso de desarrollo de *software* es elevar la calidad de este, a través de la transparencia y control sobre su desarrollo, y lograr que dichas pautas sean reproducibles en cada proyecto, independientemente de su magnitud. Aunque su elección para el SIIPOL específicamente se va del alcance de este trabajo, es necesario que sea explicado cuál se escogió y porqué se considera el correcto para el contexto actual. Actualmente existe una gran cantidad de procesos de desarrollo, agrupados en dos tendencias: los métodos ágiles y los métodos pesados.

1.3.1 *Rational Unified Process* (RUP)

RUP es uno de los procesos más generales que existen actualmente, su finalidad no está restringida a guiar desarrollo de *software*, sino cualquier tipo de proyecto. La estrategia de este proceso es conseguir su objetivo por medio de orden y documentación, lo que lo convierte en el más fiel exponente de los métodos pesados. RUP define cuatro fases (inicio, elaboración, construcción y transición) y dentro de cada una de ellas el equipo de trabajo pasa por todos los flujos que son transversales a las fases, inclusive en varias iteraciones.

Flujos de trabajo:

- Modelación del negocio: Describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.
- Requerimientos: Define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.
- Análisis y diseño: Describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requerimientos), por lo que indica con precisión lo que se debe programar.
- Implementación: Define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.
- Prueba: Busca los defectos a los largo del ciclo de vida.
- Instalación: Produce un entregable del producto y realiza actividades como empaque, instalación, asistencia a usuarios, etc. para entregar el *software* a los usuarios finales.
- Administración del proyecto: Involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.
- Administración de configuración y cambios: Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización/actualización concurrente de elementos, control de versiones, etc.
- Ambiente: Contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto; así como el procedimiento para implementar el proceso en una organización.

Fases del proceso*:*

- Inicio: Se describe el negocio y se delimita el proyecto, describiendo sus alcances con la identificación de los casos de uso del sistema.
- Elaboración: Se define la arquitectura del sistema y se obtiene una aplicación ejecutable que responde a los casos de uso que la comprometen. A pesar de que se desarrolla a profundidad una parte del sistema, las decisiones sobre la arquitectura se hacen sobre la base de la comprensión del sistema completo y los requerimientos (funcionales y no funcionales) identificados de acuerdo con el alcance definido.

- Construcción: Se logra un producto listo para su utilización que está documentado y tiene un manual de usuario. Se obtiene una o varias versiones del producto que han pasado las pruebas. Se ponen estos entregables a consideración de un subconjunto de usuarios.
- Transición: La versión ya está lista para su instalación en las condiciones reales. Puede implicar reparación de errores.

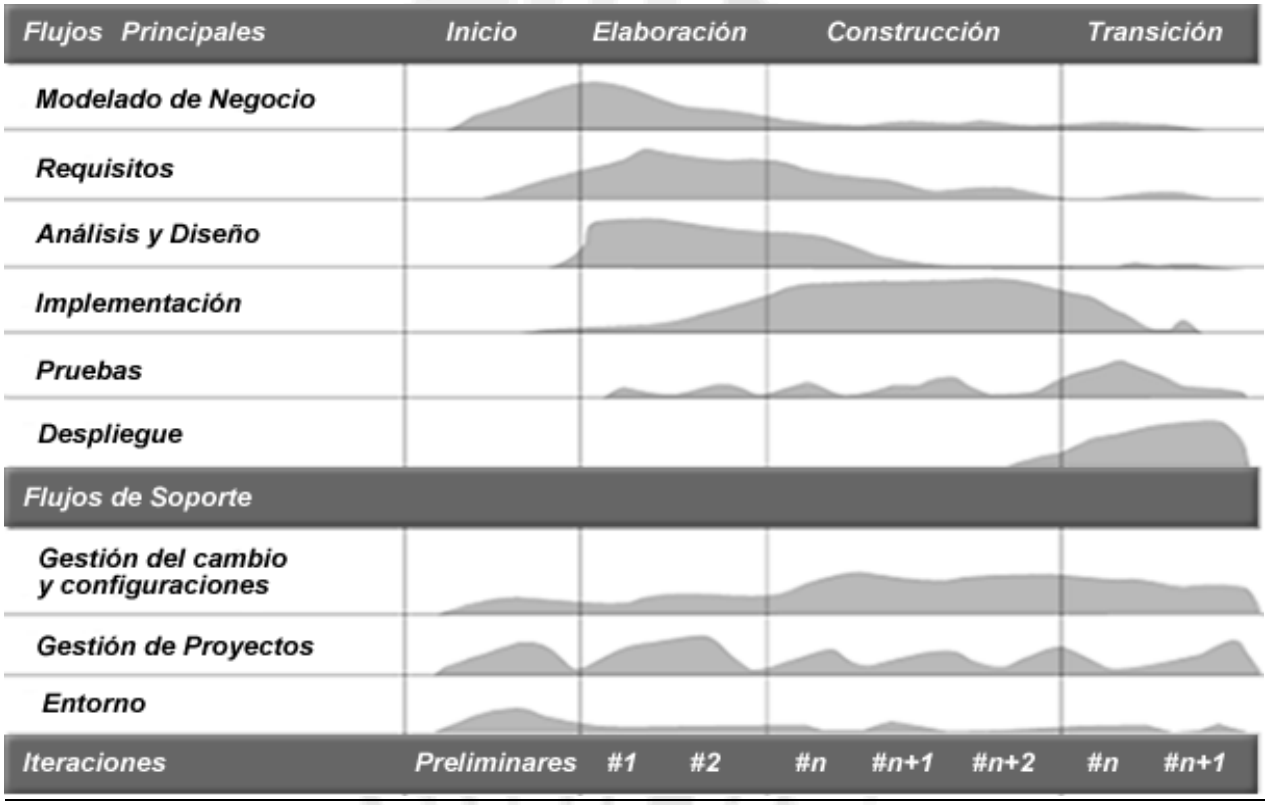

Ilustración 1: Fases y flujos de RUP

Las principales bibliografías que abordan esta metodología coinciden en definir tres características fundamentales al hablar de RUP:

a. Dirigido por casos de uso: Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo, ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso.

b. Centrado en la arquitectura: La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. RUP se desarrolla

mediante iteraciones, comenzando por los CU relevantes desde el punto de vista de la arquitectura. El modelo de arquitectura se representa a través de vistas(4+1), o sea perspectivas del sistema, que logran una abstracción particular en cada uno de los casos, en otras palabras, se trata de que en cada una de las llamadas vistas se represente el sistema en su totalidad teniendo en cuenta solo determinados aspectos.

c. Iterativo e incremental: RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros. Las iteraciones hacen referencia a pasos en los flujos de trabajo, y los incrementos, al crecimiento del producto.

RUP es más adecuado para proyectos grandes, dado que requiere un equipo de trabajo capaz de administrar un proceso complejo en varias etapas; en proyectos pequeños es posible que no sea posible cubrir los costos de dedicación del equipo de profesionales necesarios. Los requerimientos de los diversos inversores pueden ser diferentes, contradictorios o disputarse recursos limitados, así que debe encontrarse un balance que satisfaga los deseos de todos y el control de calidad no debe realizarse al final de cada iteración, sino en todos los aspectos de la producción (3).

1.3.2 *Extreme Programming* (XP)

Por otro lado, en el mundo está teniendo un gran impacto el uso de otra metodología de desarrollo de software, XP, que es clasificada como ágil y como tal trata de reducir la complejidad de software orientando el trabajo directamente al objetivo, basado en las relaciones interpersonales y la velocidad de reacción, teniendo como principal peculiaridad la presencia, a tiempo completo en el desarrollo, de un representante del cliente.

XP define *UserStories* (historias de usuario) como la base del *software* a desarrollar; estas historias son escritas por el cliente y describen las interacciones entre los clientes y el sistema, y generalmente son complementadas con otro tipo de descripción. A partir de ellas y de la arquitectura que se utilizará se planifican las entregas, así como los objetivos de cada una y las iteraciones con que contará. Esta planificación será valorada por el cliente y discutida hasta lograr el entendimiento en caso de que no esté de acuerdo desde un principio. Junto a los *UserStories* están los escenarios de prueba que comprobarán el correcto cumplimiento de las necesidades del cliente. El primer paso en una iteración es escribir las pruebas que se harán al código y que serán lo más automatizadas posible, para comprobar dinámicamente el software antes de cada entrega. Se usan mucho las pruebas de unidad para garantizar el funcionamiento de cada parte por sí sola.

Una característica distintiva de XP es la programación en parejas, con el objetivo de que el código sea revisado y validado antes de ser escrito; la refactorización de código está presente durante todo el desarrollo, y no es más que escribir el mismo código fuente nuevamente buscando claridad, pero sin cambiar la funcionalidad resultante. Las parejas no serán siempre las mismas, sino que se pretende que cada desarrollador haya formado dupla al menos una vez con todos los demás, de donde se desprende que el código es de propiedad colectiva y cada uno es responsable por todo el proyecto (4).

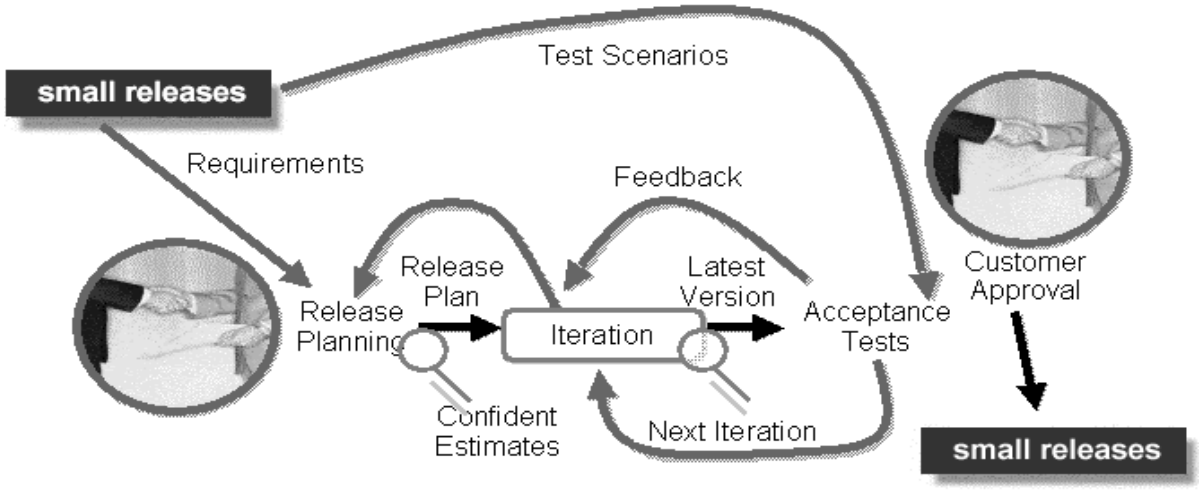

Ilustración 2: Proyecto con XP

En XP se programa solo la funcionalidad requerida para la entrega actual, excepto que se presente la necesidad de programar algo más flexible en un momento dado, se codifica buscando la simplicidad, y se hacen entregas frecuentemente. Es iterativo e incremental. De manera general se puede expresar que es recomendable usar XP en proyectos en los que los requisitos tienen altas probabilidades de cambiar con el tiempo (por ejemplo, porque el cliente no tiene claro lo que quiere, o porque el cambio de requisitos está ligado al dominio del problema a resolver), en proyectos con alto riesgo (por ejemplo, proyectos con una fecha de entrega que es indispensable cumplir, o proyectos totalmente novedosos para la industria) o en proyectos con un grupo pequeño de programadores (entre 2 y 12), aunque el equipo completo sea bastante más extenso (incluye a jefes de equipo y representantes de clientes).

1.3.3 Análisis de la selección del Proceso de Desarrollo

Cada uno de los procesos puede ser adaptado a cada proyecto, pero teniendo en cuenta todos los aspectos abordados previamente; se pueden tomar en cuenta varios puntos definitorios a la hora de tomar la decisión de cuál proceso de desarrollo usar en la solución propuesta, como la magnitud del proyecto, la cantidad de miembros del equipo, el tiempo con que se dispone para su culminación, lo

distante que se encuentran desarrolladores y clientes geográficamente, la facilidad de darle continuidad por parte del personal venezolano, las expectativas del cliente y la experiencia de los desarrolladores, entre otros.

La magnitud del *software* a producir hace que las partes a entregar deban quedar bien documentadas internamente; los formalismos se hacen necesarios para prevenir malentendidos que no son deseables en el marco en el que se desenvuelve la producción del software. La cantidad de miembros del equipo, que supera los 100, deriva en la decantación por un proceso que promueva el orden y el control, tanto de los recursos humanos como de los artefactos producidos. El tiempo estimado de producción, que supera los 16 meses, unido a los factores anteriormente mencionados y a la distancia que separa a Cuba de Venezuela, hacen que la opción más viable económicamente sea producir el software en Cuba, y no es conveniente tampoco mantener a un grupo de clientes en el equipo de desarrollo por las mismas razones; la consecuencia es que la comunicación personal se dificulta y la mejor opción es la formalidad. Por otra parte, el cliente espera un producto que contenga manuales, documentación y que proporcione una detallada guía de uso; este requisito es cumplido de manera más efectiva por RUP. La transferencia tecnológica, otro de los puntos que se deben asegurar en el proyecto, resulta posible gracias a la documentación que genera RUP. La experiencia de los desarrolladores es muy poca para enfrentar un proyecto de esta magnitud, sin pautas bien definidas y una referencia disponible en todo momento. Por todas estas razones se considera que la elección de RUP para desarrollar el SIIPOL es correcta.

1.4 Lenguaje de Modelado

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; aunque todavía no es un estándar oficial, sí lo es *de facto* (5). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un plano del sistema, incluyendo aspectos conceptuales, tales como procesos de negocios y funciones del sistema, y aspectos concretos, como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables. En fin UML es el enlace entre quien tiene la idea y el desarrollador, la comunicación en su principal objetivo.

Es importante resaltar que UML es un lenguaje para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Se

puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como RUP), pero no especifica en sí mismo qué metodología o proceso usar. El UML está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Debido a que el UML es un lenguaje, cuenta con reglas para combinar tales elementos. En UML 2.0, la última versión disponible de este estándar *de facto,* hay 13 tipos diferentes de diagramas. Para comprenderlos de manera concreta, a veces es útil categorizarlos jerárquicamente

Los **Diagramas de Estructura** enfatizan en los elementos que deben existir en el sistema modelado; incluye, entre otros, Diagrama de clases, Diagrama de componentes, Diagrama de objetos y Diagrama de paquetes.

Los **Diagramas de Comportamiento** enfatizan en lo que debe suceder en el sistema modelado; ejemplo son el Diagrama de actividades y Diagrama de casos de uso.

Los **Diagramas de Interacción** son un subtipo de diagramas de comportamiento, que enfatiza sobre el flujo de control y de datos entre los elementos del sistema modelado; aquí se puede ver el Diagrama de secuencia y de Diagrama de colaboración.

El uso de UML y el trabajo continuo que se ha venido realizando sobre él lo convierten en una práctica que asegura la especificación de los procesos en el desarrollo del software, y es muy favorable su uso como vía de comunicación y documentación (6).

1.5 La Plataforma de Desarrollo

En la informática, una plataforma de desarrollo es el entorno de software común en el cual se desenvuelve la programación de un grupo definido de aplicaciones. Comúnmente se encuentra relacionada directamente a un sistema operativo; sin embargo, también es posible encontrarla ligada a una familia de lenguajes de programación o a una Interfaz de Programación de Aplicaciones (API).

Un API es el conjunto de funciones y procedimientos o métodos si se refiere a programación orientada a objetos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Uno de los principales propósitos de un API consiste en proporcionar un conjunto de funciones de uso general. De esta forma, los programadores se benefician de sus ventajas, haciendo uso de su funcionalidad, evitándose el trabajo de programar todo desde el principio.

Con el reto de construir una solución eficiente, fiable, extensible e integrable a otras aplicaciones del sistema gubernamental venezolano, que además pueda aprovechar todas las potencialidades de las redes informáticas, una de las decisiones más importantes que se tomó en el proyecto fue la elección de la plataforma de desarrollo. Dos de las principales representantes son J2EE y .NET. Para la

implementación del SIIPOL se seleccionó J2EE, y se hace necesario el análisis comparativo entre ambas.

Definición de las soluciones

.NET es el nombre genérico bajo el que *Microsoft* agrupa su última generación de productos y servicios; mientras que J2EE no es un producto en sí mismo, sino un estándar de programación en Java desarrollado por *Sun Microsystems*, y adoptado por la mayoría de los principales fabricantes de software del mercado (IBM, Oracle Corporation) para desarrollar sus propios productos (7).

El funcionamiento

En el aspecto funcional ambas plataformas guardan varias similitudes. Se programa en un lenguaje que luego se compila a un código intermedio (*Intermediate Language* en el caso de Microsoft .NET y *Bytecodes* en el caso de Java). Este código se ejecutará en un entorno de ejecución que transformará el lenguaje intermedio a código propio de la máquina en la que se corre la aplicación, *Common Language Runtime* (CLR) en *Microsoft* .NET y *Java Runtime Environment* (JRE) en J2EE.

El entorno de desarrollo

Una de las principales ventajas de desarrollar sobre la plataforma .NET consiste en la posibilidad de programar los distintos componentes de una aplicación empleando distintos lenguajes. Otra ventaja radica en su nuevo entorno de programación, el *Visual Studio* .NET, con el que *Microsoft* sigue ofreciendo a los programadores el mejor entorno de desarrollo que permite una programación más fácil y cómoda, con los ahorros de tiempo y costes que esto supone. Por otra parte, el J2EE sólo puede emplear el lenguaje Java para desarrollar, y son múltiples los productos que existen en el mercado ofreciendo entornos de desarrollo adecuados. Si bien estas herramientas facilitan mucho la labor de los programadores, siguen sin llegar al nivel de integración ofrecido por *Microsoft* (Montejava.es).

Entorno de ejecución

En el entorno de ejecución J2EE presenta una ventaja fundamental sobre .NET: su capacidad de ejecutarse en distintas plataformas de *hardware* y sistemas operativos. Si bien los productos J2EE de distintos fabricantes no son siempre todo lo compatibles entre sí que cabría esperar, debido a las distintas interpretaciones que cada fabricante hace del estándar; es cierto que estos productos ofrecen mucha más portabilidad que .NET, que en estos momentos sólo está preparada para ejecutarse sobre plataformas Win32. En cuanto a otros aspectos fundamentales del entorno de ejecución, como son el rendimiento, la escalabilidad y la seguridad, se dice que J2EE está por delante de .NET, y es por ello

que en los entornos en los que estos son los aspectos fundamentales tales como los entornos transaccionales de las grandes multinacionales suele ser la plataforma elegida (7).

Madurez de la plataforma

La experiencia y madurez de la plataforma están a favor de J2EE, ya que salió al mercado cuatro años antes que .NET, y en dicho tiempo se han ido desarrollando multitud de productos y servicios, a la vez que se han ido corrigiendo errores y cubriendo las carencias y necesidades detectadas, por lo que hoy cuenta con una gama de productos altamente consolidados; mientras que .NET se encuentra aún en su fase de pruebas (7).

1.5.1 Análisis de la selección de la plataforma

Después de haber analizado las características de las dos plataformas tecnológicas más utilizadas en el mundo a la hora de desarrollar una aplicación *web*, se está en condiciones de realizar la selección, siempre teniendo en cuenta el tipo de aplicación que se pretende desarrollar. Para seleccionar la plataforma se tiene como elemento indispensable el entorno de ejecución, ya que se requiere de un sistema multiplataforma, de alto rendimiento, escalabilidad y seguridad, por lo que la opción más acertada para estos requerimientos es la plataforma J2EE; de ahí que se haya decidido seleccionarla como plataforma de desarrollo para dicho sistema.

1.6 Entorno de desarrollo

En inglés *Integrated Development Environment* (IDE) es un programa compuesto por un conjunto de herramientas para un programador (8). Puede dedicarse exclusivamente para un lenguaje de programación o bien para varios.

Teniendo en cuenta que cuando se trabaja sobre la plataforma J2EE el lenguaje de programación a usar es *Java*, las opciones más atractivas para la elección de un IDE son *Eclipse y NetBeans*, que gozan de reconocimiento a nivel mundial.

1.6.1 Eclipse

Eclipse es un IDE de código abierto independiente de una plataforma, es una aplicación de cliente enriquecido, emplea *plug-ins* para proporcionar toda su funcionalidad, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Eclipse es, en el fondo, únicamente un armazón sobre la que se pueden montar herramientas de desarrollo. La arquitectura de *plug-ins* permite, además de integrar diversos lenguajes, introducir otras aplicaciones

accesorias que pueden resultar útiles durante el proceso de desarrollo, tales como: herramientas UML, editores visuales de interfaces, ayuda en línea para librerías, entre otros.

1.6.2 *NetBeans*

NetBeans es de código abierto, soporta el desarrollo de todos los tipos de aplicación Java (J2SE, *web*, EJB y aplicaciones móviles). Existe un número importante de módulos para extenderlo. Es un producto libre y gratuito sin restricciones de uso. Entre sus características se encuentra un sistema de proyectos basado en control de versiones y refactorización. Todas las funciones del IDE son provistas por *plug-ins*, al igual que Eclipse.

La versión 6.0 soporta las tecnologías *Enterprise JavaBeans* (EJB) 3.0, JAX-WS 2.1, *Java Server Faces* 1.2, *Java Server Page* 2.1, JSP *Standard Tag Library* (JSTL) 1.1, el editor es más rápido y más inteligente.

1.6.3 Análisis de la selección del IDE

Aunque ambos IDE son similares en cuanto a las funcionalidades y ventajas que ofrecen para el programador, la dirección del proyecto decidió escoger Eclipse como IDE de desarrollo, debido a que cuando se inició, la versión 6.0 de *NetBeans* no había sido liberada; sólo se contaba con la 5.0, la cual no brindaba mejor soporte que Eclipse para el desarrollo de aplicaciones w*eb*.

1.7 Herramienta CASE

Las Herramientas CASE (*Computer Aided Software Engineering*, Ingeniería de Software Asistida por Computadora) son aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software, reduciendo su coste en términos de tiempo y dinero. Estas herramientas pueden ayudar en muchos de los aspectos del ciclo de vida de desarrollo del software, en tareas como la realización del diseño del proyecto, cálculo de costes, implementación de una parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores, entre otras.

Las herramientas CASE de modelado con UML permiten analizar y diseñar orientado a objetos y abstraer el código fuente, a un nivel donde la arquitectura y el diseño se tornan más obvios y más fáciles de entender y modificar. Cuanto más grande es un proyecto, es más importante utilizar una herramienta CASE.

Entre los principales beneficios que ofrece la utilización de una Herramienta Case se encuentran:

- Mejora la comunicación entre usuario y especialista, ya que al tener incorporada la visualización de diagramas y de otras herramientas del análisis estructurado, actúa como un elemento que acelera la relación usuario/especialista.
- Permite la facilidad de revisión de aplicaciones instaladas.
- Son capaces de generar automáticamente las instrucciones del programa fuente. Esto permite acelerar el tiempo dedicado a la elaboración de programas y asegura además una estructura estándar y consistente para el programa.
- Da la posibilidad de la generación de documentación técnica.

Dentro de la gama de herramientas de modelado utilizadas en el mundo se encuentran *Rational Rose y Visual Paradigm*, que constituyen dos de las herramientas más usadas en la actualidad.

1.7.1 *Rational Rose*

Es la herramienta CASE desarrollada por los creadores de UML (*Booch, Rumbaugh y Jacobson*), que cubre todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases y entregables. El navegador UML de *Rational Rose* permite establecer una trazabilidad real entre el modelo (análisis y diseño) y el código ejecutable. Facilita el desarrollo de un proceso cooperativo en el que todos los agentes tienen sus propias vistas de información (vista de casos de uso, vista lógica, vista de componentes y vista de despliegue), pero utilizan un lenguaje común para comprender y comunicar la estructura y la funcionalidad del sistema en construcción. *Rational Rose* presenta un entorno de diseño muy atractivo y los cambios rápidos en el código a medida que se va desarrollando el modelado son muy buenos. Permite la generación de documentos en formato HTML, XML y PDF de los reportes.

A pesar de que este producto está pensado para un ciclo evolutivo en espiral, se adapta muy bien al ciclo en cascada al que el proyecto se enfrenta. Independientemente del tipo de plataforma o aplicación (Eclipse, Java, .NET, aplicaciones integradas o de informática móvil), Rational complementa todo el proceso de principio a fin.

Tras configurar e instanciar la generación de código sobre una estructura de datos, se comprueba que esta generación se automatiza sobremanera; el programa intenta realizar acciones por su cuenta que no representan la voluntad del programador, por lo que es una tarea de especial cuidado.

1.7.2 *Visual Paradigm(VP)*

Es una herramienta diseñada para desarrollar software con programación orientada a objetos. Busca reducir la duración del ciclo de desarrollo, brindando ayuda a arquitectos, analistas, diseñadores y desarrolladores; permite el uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación y tiene capacidades de ingeniería directa e inversa.

Usa UML como lenguaje de modelado. Presenta una interfaz de uso intuitiva y con muchas facilidades a la hora de modelar los diagramas que soportan la Ingeniería de Requerimientos (9).

Una de las características más importantes de su uso es que brinda la posibilidad de sincronización del modelo de diseño y el código en todo el ciclo de desarrollo una vez que se integra con el Eclipse, permitiendo la facilidad de programar directamente sobre el código fuente generado y a su vez actualizar el diseño con cambios que se realicen en la programación. Tiene una alta disponibilidad de múltiples versiones para cada necesidad, al igual que en las plataformas *Linux,* MacOS y *Windows.*

Presenta un innovador analizador textual donde se introduce texto extraído de conversaciones con el cliente, se definen actores, entidades, casos de uso disponibles para la generación de artefactos posteriores. Así mismo posee una herramienta de generación de reportes en formato PDF o HTML configurable y selectiva, se integra con entornos como Eclipse, *Hibernate* y Subversion, e importa o exporta formatos estándares de otras herramientas CASE como el *Rational Rose*.

1.7.3 Análisis de la selección del la herramienta CASE

El VP constituye un candidato más fuerte para la elección de una herramienta CASE para el proyecto CICPC, debido principalmente a que iguala a *Rational Rose* en cuanto a capacidades y es mucho más integrable con el resto de las herramientas y estándares escogidos.

1.8 *Frameworks* utilizados en la solución.

Entrando en especificidades de la plataforma y las facilidades que brinda, se hace necesario hablar de los frameworks de los que se dispondrá. Una precondición importante para abordar el tema que corresponde a esta sección es el conocimiento del significado del término *framework. Framework* es una estructura de soporte definida en la cual un proyecto de software puede ser organizado y desarrollado. Típicamente, un *Framework* puede incluir soporte de programas, bibliotecas y un lenguaje interpretado para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

La selección de un conjunto de *frameworks* para integrarlos y dar soporte a la arquitectura del proyecto, que se va del alcance de los autores, constituye un paso esencial al concebir un proyecto de

software de gran tamaño; dada la importancia de esta tarea, es necesario justificar su uso; a continuación se presentan los principales *frameworks* por cada una de las capas lógicas de la aplicación.

1.8.1 Capa de Presentación

Es necesario definir el concepto de **Modelo-Vista-Controlador (MVC):** este es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones *web* (10).

Modelo: Esta es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos; por ejemplo, no permitiendo comprar un número de unidades negativo, calculando si hoy es el cumpleaños del usuario o los totales, impuestos o importes en un carrito de compra.

Vista: Presenta el modelo en un formato adecuado para interactuar con este, usualmente la interfaz de usuario.

Controlador: Este responde a eventos, usualmente acciones del usuario, e invoca cambios en el modelo y probablemente en la vista.

Struts es un *framework* de la capa de presentación que implementa el patrón MVC en Java (11). Este separa muy bien la gestión del ciclo de vida de la aplicación, del modelo de objetos de negocio y de la generación de la interfaz de usuario (*Scribd*). Uno de los mayores aciertos que tuvo Struts fue reducir la repetición innecesaria de tareas y código común. El corazón de Struts es el Controlador, implementa el patrón de diseño *Front Controller* y el patrón *Command* para manipular la petición del cliente; tiene solo un manipulador por cada petición, cuestión que mejora potencialmente JSF, el *Framework* que se explicara en la próxima sección.

Struts obliga a extender determinadas clases a quien lo use, define la navegación de forma declarativa usando ficheros XML y es extensible; el único problema es que se usa cada vez menos ante la ascensión de JSF, al que muchos denominan su sucesor.

JSF (Java Server Faces), estándar para aplicaciones web basadas en Java, facilita la construcción de aplicaciones siguiendo también el patrón MVC (Modelo-Vista-Controlador); permite manipular varios eventos en cada página a diferencia de su predecesor Struts; posee un modelo de componentes orientado a objetos, conversión de tipos, ayuda a la separación de responsabilidades (desarrolladores de componentes, desarrolladores de lógica de aplicación, montadores de páginas) y tiene un poderoso sistema de navegación declarativa; usa simples clases Java como controladores; además de que

permite la fácil incorporación de potencialidades AJAX, posee un conjunto prefabricado de componentes de interfaz de usuario, y permite utilizar el modelo de programación orientado a eventos (12).

Al igual que Struts, JSF pretende normalizar y estandarizar el desarrollo de aplicaciones web. Hay que tener en cuenta que JSF es posterior a Struts, y por lo tanto se ha nutrido de la experiencia de este, mejorando algunas sus deficiencias. El soporte de JSF en IDE como Eclipse y Netbeans es mejor. Constantemente se crean nuevos componentes JSF y goza de gran soporte, pues todos los servidores de aplicaciones tienen la posibilidad de contener aplicaciones con JSF.

Ajax4JSF es una extensión *Open Source* del estándar JSF que se integra con este con gran facilidad, evita escribir código *Java Scripts* al brindar una amplia gama de componentes, permite agregar capacidades AJAX (del término en inglés *Asincronous Java Script and XML*) incluso a aplicaciones JSF ya creadas. En fin, las prestaciones que brinda la poderosa combinación de JSF con AJAX, hacen que sea más atractiva por la comodidad que supone para el desarrollador y además por integrarse eficientemente con otros *frameworks* (13).

1.8.2 Capa de Lógica de Negocio

Spring es un *Framework* de código abierto que interviene en todas las capas arquitectónicas de una aplicación J2EE, brinda soporte a *Java Server Faces*, y se integra con varios frameworks de acceso a datos, para formar una poderosa herramienta para el desarrollo de aplicaciones empresariales.

Spring es un *Framework* ligero que puede ser distribuido en un archivo de extensión **jar** (un tipo de archivo que permite ejecutar aplicaciones escritas en lenguaje Java) que ocupa una capacidad de algo más de 1 Mb. Promueve el bajo acoplamiento de las clases conformantes de la solución por medio de la técnica conocida como inversión del control (LoC). Soporta la Programación Orientada a Aspectos (AOP) que complementa la Programación Orientada a Objetos (POO), proponiendo otra manera de pensar sobre la estructura de un programa. Mientras que la POO descompone las aplicaciones en una jerarquía de objetos, la AOP descompone los programas en aspectos o preocupaciones. Dichas preocupaciones se convierten en servicios del sistema, separados de la lógica de negocio y que se ejecutan de manera trasversal a la funcionalidad base, lo que proporciona una definición de responsabilidades superior. *Spring* básicamente es un contenedor, que se encarga de gestionar y administrar el ciclo de vida y configuración de las clases de la aplicación.

Spring es un entorno diseñado para aumentar la productividad, liberando al desarrollador de tareas repetitivas, ayudándolo a hacer diseños más consistentes y limpios (14). Es maduro y muy amplio

(contrariamente a lo que muchas personas piensan, no es sólo para hacer aplicaciones *web*, las grandes compañías lo consideran el *Framework* líder).

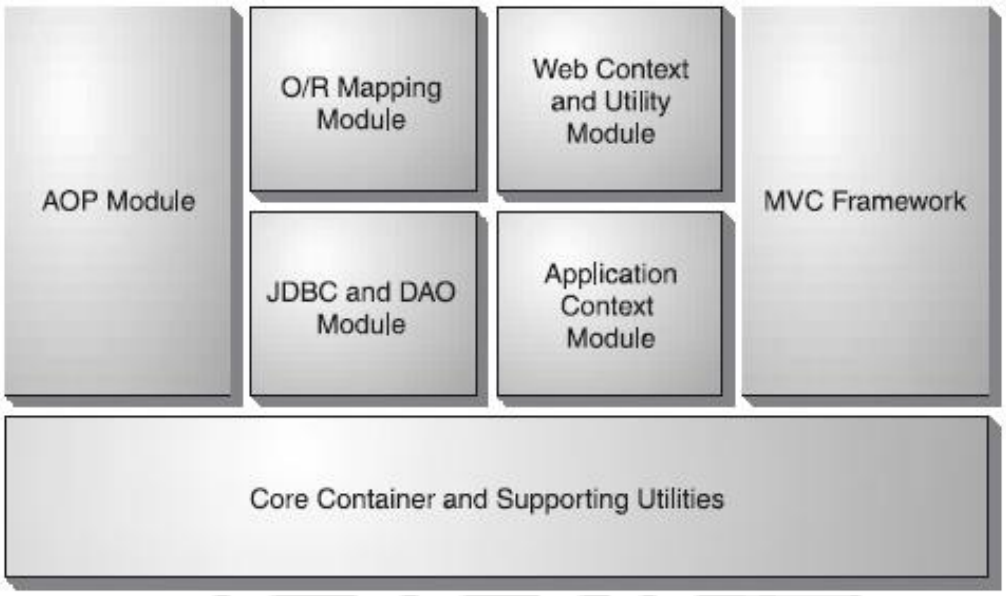

Ilustración 3: Composición de Spring

JBoss Seam es un potente *Framework* de Java que unifica la integración de diversas tecnologías, tales como: AJAX, JSF, *Enterprise Java Beans* y *Web Services*. Surgió con el objetivo de eliminar la complejidad de la arquitectura y del API, habilitando a los desarrolladores para que puedan desarrollar complejos sistemas con simples anotaciones en las clases y poco código XML. Soporta *Rich-Faces,* soluciones *open source* de JSF basado en AJAX. Se inclina más por las anotaciones que por el XML.

La selección de uno u otro *framework* de esta categoría es arbitraria, cualquiera de los frameworks se integraría perfectamente a este trabajo y realmente los responsables de la arquitectura se guiaron por experiencia de otras producciones de software similares que avalan su uso, para seleccionar *Spring*.

1.8.3 Capa de Acceso a Datos

Existen en la actualidad diversos *frameworks* y *APIs* de acceso a datos compatibles con la plataforma J2EE que han de ser considerados en el análisis y justificación de los estándares adoptados en la construcción del SIIPOL.

JDBC es el acrónimo de *Java Database Connectivity,* un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice (16).

El API JDBC se presenta como una colección de interfaces Java y métodos de gestión de manejadores de conexión hacia cada modelo específico de base de datos. Para utilizar una base de datos particular, el usuario ejecuta su programa junto con la librería de conexión apropiada al modelo de su base de datos, y accede a ella estableciendo una conexión; para ello especifica la base de datos y los parámetros de conexión. A partir de allí puede realizar cualquier tipo de tareas con las bases de datos a las que tenga permiso: consultas, actualizaciones, creado, modificado y borrado de tablas, ejecución de procedimientos almacenados en la base de datos, etc.

Programar orientado a objetos y trabajar con bases de datos relacionales es complicado; el desarrollador tiene que escribir mucho código para lograr una correspondencia entre la representación relacional y la objetual; otro aspecto poco deseable en una aplicación de gran magnitud es la dependencia extrema del código de acceso a datos con respecto a la base de datos específica que modela la solución, se debe evitar a toda costa tal acoplamiento. Además JDBC no proporciona un sistema de caché (retención de datos en memoria que reduce el acceso a disco), que sí proveen algunos *frameworks* como Hibernate, el cual se analizará a continuación.

iBATIS es un *framework* de código abierto basado en capas desarrollado por *Apache Software Foundation,* que se ocupa de la capa de persistencia. Puede ser implementado en Java y .NET (17).

Permite mapear sentencias SQL hacia clases. Las sentencias SQL son desacopladas de la aplicación, poniéndolas en ficheros descriptores en formato XML. La idea principal detrás de iBATIS es la disminución de líneas de código Java que necesita el desarrollador para acceder a la base de datos por medio del API JDBC.

Es posible subdividir la capa de Persistencia en tres subcapas:

- La capa de *Abstracción* será la interfaz con la capa de la lógica de negocio, hace de fachada entre la aplicación y la persistencia. Se implementa de forma general mediante el patrón *Data Access Object* (DAO), y particularmente en iBATIS se implementa utilizando su *framework* DAO (ibatis-dao.jar).
- La capa de *Framework* de Persistencia será la interfaz con el gestor de Base de Datos, ocupándose de la gestión de los datos mediante un API. Normalmente en Java se utiliza JDBC; iBATIS utiliza su *framework* SQL-MAP (ibatis-sqlmap.jar).
- La capa de *Driver* se ocupa de la comunicación con la propia Base de Datos utilizando un *Driver* específico para la misma.

Como se puede apreciar iBATIS es superior desde el punto de vista de la usabilidad con respecto a JDBC; el mismo concepto de *framework* le hace proveer una abstracción mayor al programador que JDBC, y mejora potencialmente varios de los problemas que supondrían el uso de JDBC, como el acoplamiento y el bajo nivel de reutilización.

El único detalle *a priori* que hace aconsejable buscar otra opción es que el uso de iBATIS obliga a un conocimiento exhaustivo del lenguaje de consultas SQL y el dialecto específico que se esté utilizando; este modelo se hace muy tedioso, repetitivo y propenso a errores, dependiente de la base de datos concreta y poco productivo en modelos de cierta complejidad. De esta manera se estima que el 30% o 35% del código de una aplicación es gastado en el mapeo de datos a la base de datos.

Hibernate es una solución ORM (*Object-Relational Mapping*) para Java, como todas las herramientas de su tipo. *Hibernate* busca solucionar el problema de la diferencia entre el modelo orientado a objetos y el usado en las bases de datos modelo relacional mediante archivos declarativos. Para lograr esto permite al desarrollador detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen. Con esta información *Hibernate* le permite a la aplicación manipular los datos de la base operando sobre objetos, con todas las características de la OOP. *Hibernate* convierte los datos entre los tipos utilizados por Java y los definidos por SQL. *Hibernate* genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todas las bases de datos con un ligero incremento en el tiempo de ejecución (18).

Hibernate está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de un modelo objetual existente.

Se decidió el uso de *Hibernate* para una solución como la nuestra argumentando que es un poderoso medio para mapear clases de Java a tablas de la base de datos, procedimiento que se facilita mediante la configuración de ficheros XML y a la vez desacopla la base de datos de las clases de la aplicación. *Hibernate* también provee con HQL una poderosa vía de comunicación entre el programador y la base de datos, al dotarlo de un lenguaje invariante con respecto a esta y además sintácticamente muy parecido al SQL. *Hibernate* además es *Open Source* y la licencia del producto está eximida de costo.

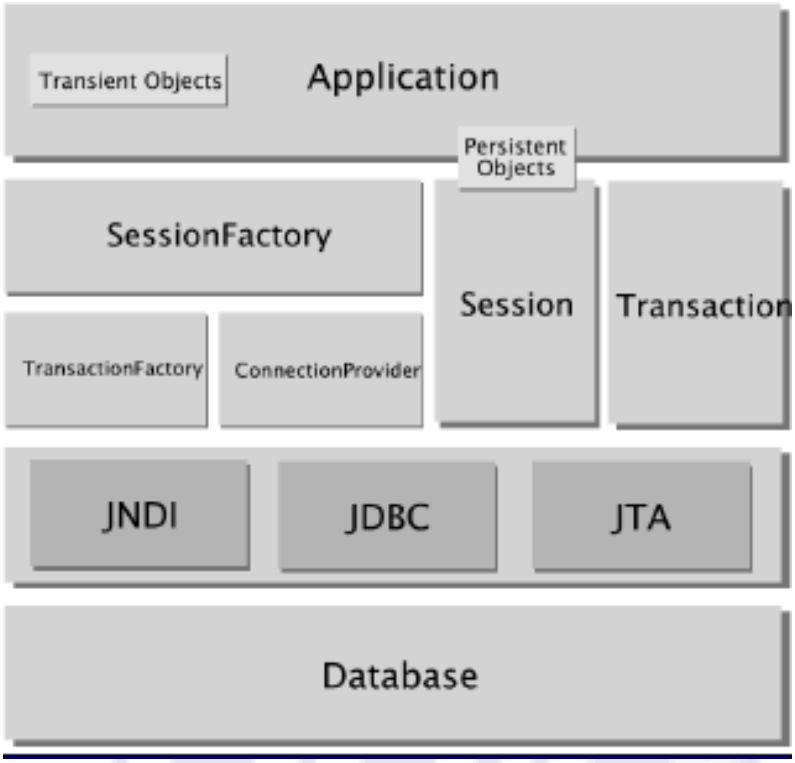

Ilustración 4: Arquitectura de Hibernate

1.9 Sistema Gestor de base de datos

Oracle es un sistema de gestión de base de datos relacional fabricado por *Oracle Corporation*. Se considera uno de los sistemas de bases de datos más completos, destacando su soporte de transacciones, estabilidad, escalabilidad, y que es multiplataforma.

Es un producto vendido a nivel mundial, aunque la gran potencia que tiene y su elevado precio hacen que sólo se vea en empresas muy grandes y multinacionales, por norma general. En el desarrollo de aplicaciones *web* pasa lo mismo: como es un sistema muy caro no está tan extendido como otros, por ejemplo, Access, MySQL y SQL Server.
Para desarrollar en *Oracle* se utiliza PL/SQL, un lenguaje de 5ta. Generación, bastante potente para tratar y gestionar la base de datos, y muy versátil (19).

MySQL es un gestor de base de datos muy popular. Constituye un servidor de bases de datos de código abierto, confiable, rápido, compacto, poderoso y multiplataforma.

Fue desarrollado por la empresa Mysql AB, y una de sus grandes ventajas es que se puede utilizar gratis y además constituye una de las bases de datos más fuertes en nuestro medio. MySQL es rápido y robusto.

Para el desarrollo de una aplicación tan grande cuyo modelo relacional se estima que supere las 400 tablas, se prefiere la seguridad que da un gestor reconocido y usado por las grandes empresas. Aun cuando es mayor la popularidad de sus principales competidores, *Oracle* es el designado para almacenar los datos de la aplicación SIIPOL.

1.10 Propuesta de Solución

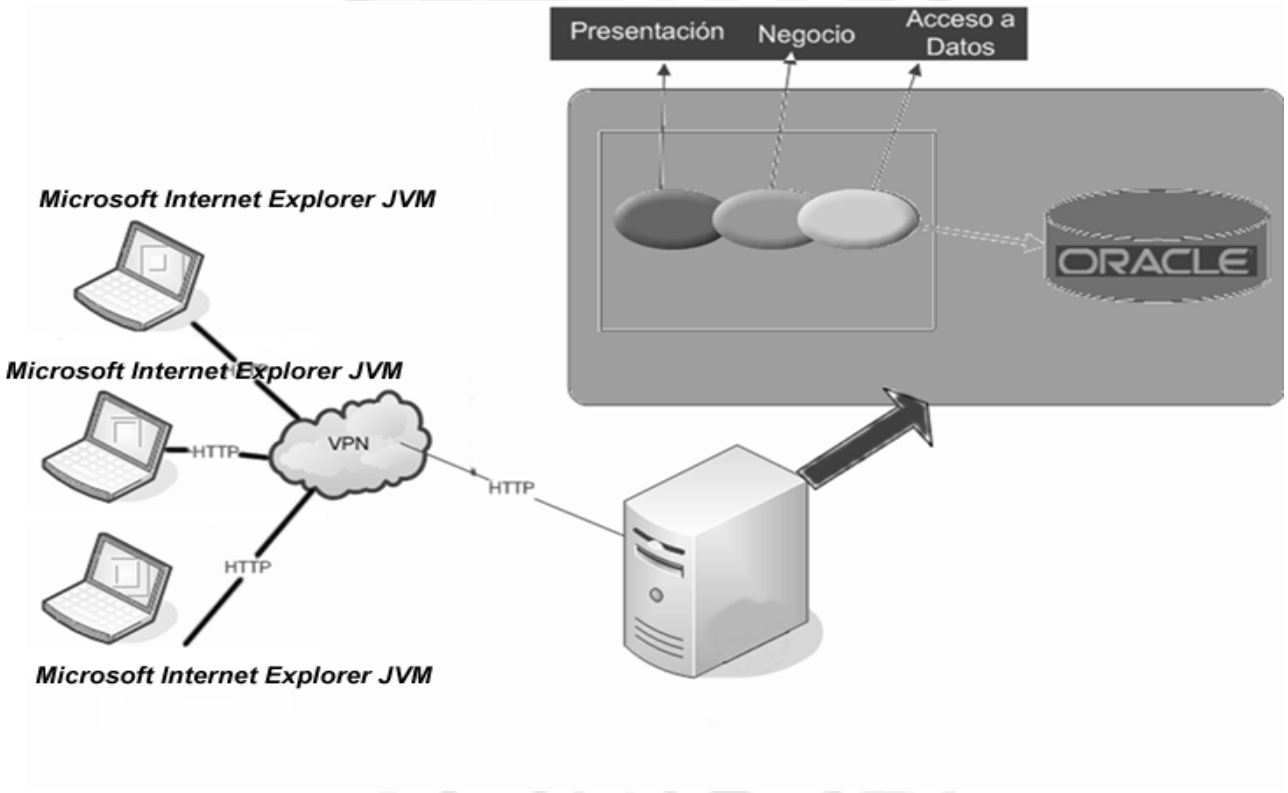

Ilustración 5: Visión de la propuesta de solución

La propuesta de solución a un alto nivel especifica una aplicación cliente servidor a través del protocolo HTTP, con un servidor de aplicaciones *HP Integrity*, en el que compartirán recursos la base de datos *Oracle* y el servidor *web Apache Tomcat*, en el cual estará publicada una aplicación que utiliza el modelo cliente-servidor como estilo arquitectónico.

Una vista más cercana al modelo de N capas que utiliza la aplicación muestra la organización del código fuente; esta organización ofrece modularidad, escalabilidad y facilita tanto el desarrollo como el mantenimiento de la aplicación.

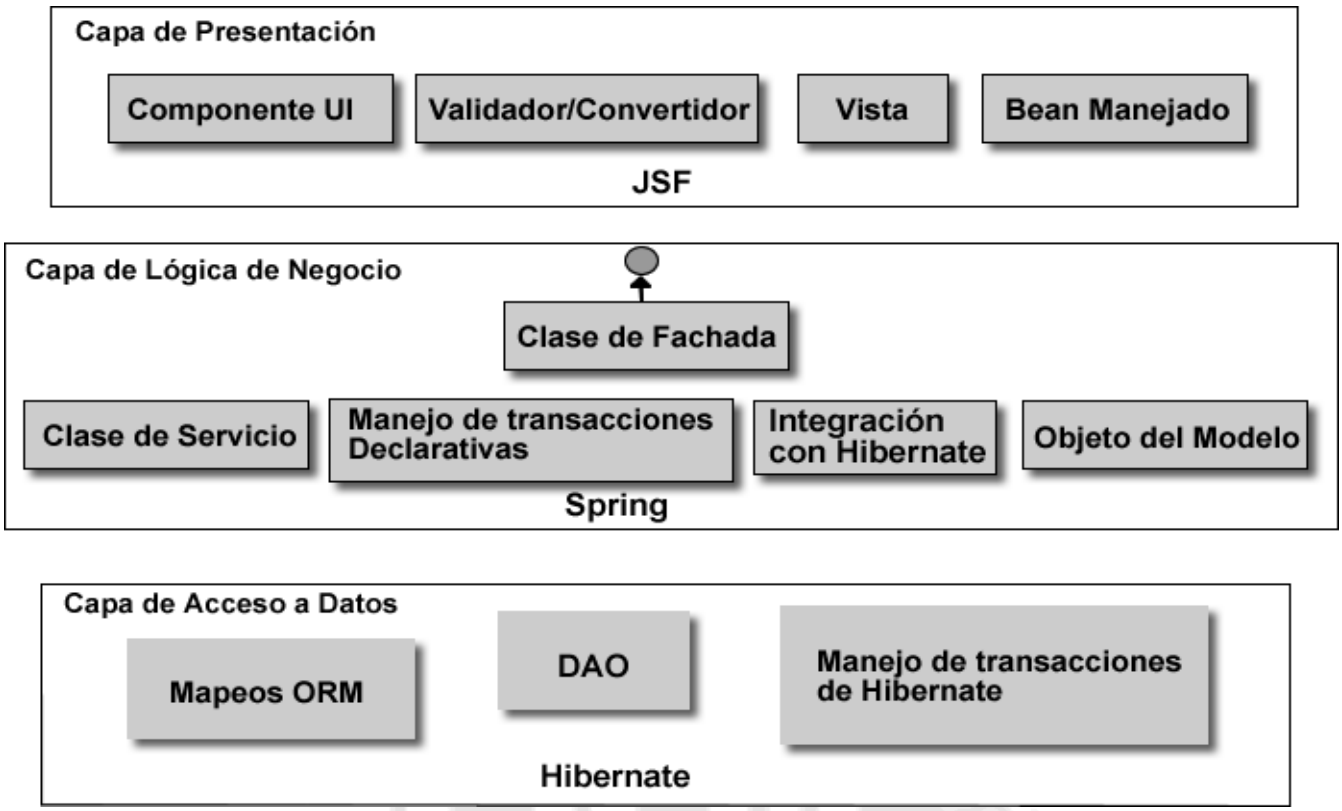

Ilustración 6: Modelo n-capas de la solución

Otro patrón arquitectónico que se utiliza es MVC; que tiene la ventaja de que los cambios en las vistas no afectan el resto de la aplicación, y que se puede mostrar distintos tipos de vista al mismo tiempo usando el mismo modelo.

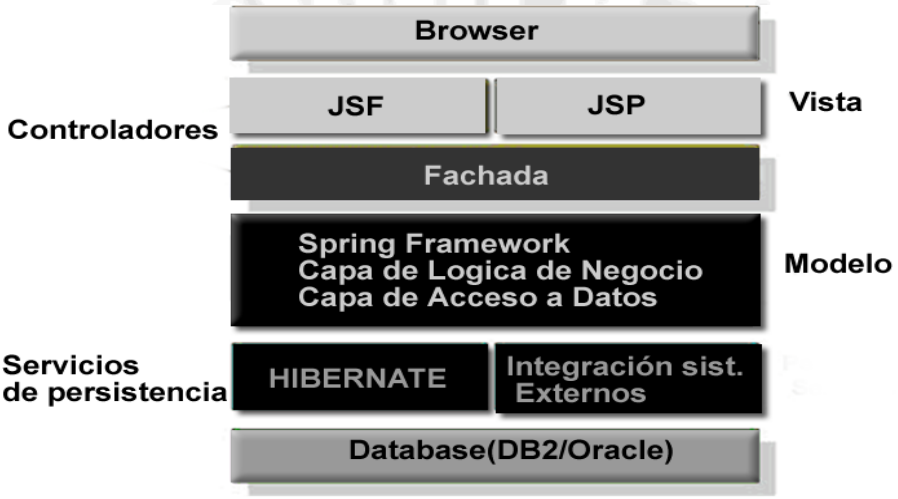

Ilustración 7: Modelo Vista Controlador

Otro tema de importancia, la seguridad, se garantiza con Acegi Security, un framework no intrusivo que abstrae al desarrollador de estas tareas, el acceso a la funcionalidad se maneja paralelamente a la misma, con muy pocos puntos de contacto, a través de filtros que interceptan las peticiones web y determinan si se puede realizar, el aseguramiento de la aplicación no es del alcance de este trabajo, ni responsabilidad de los desarrolladores, por lo que se obvia detallarlo.

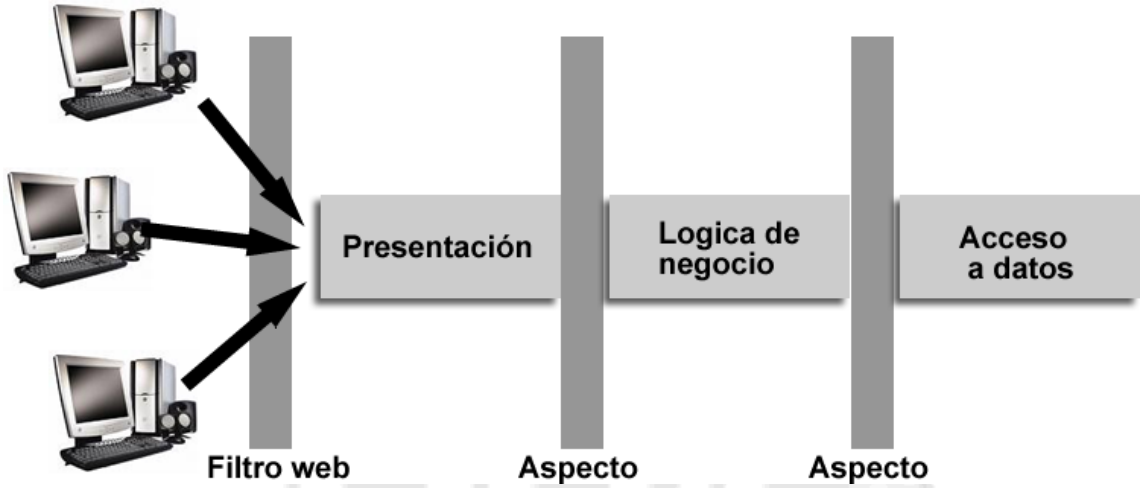

Ilustración 8: Manejo de la seguridad

1.11 Conclusiones

En este capítulo se explicó la necesidad del trabajo, se hizo referencia al estudio de sistemas que fueron interesantes para el diseño de la solución de software, se fundamentó el uso de las diferentes tecnologías, la metodología y los estándares en el desarrollo del producto final, y se propuso una solución que integra los elementos estudiados.

A partir de este análisis se puede comenzar el proceso de producción individual de cada uno de los subsistemas y sus módulos, por lo que partiendo de las bases de este trabajo explicadas, es posible comenzar un adentramiento en los temas específicos del subsistema Investigación Criminalística y específicamente el trabajo de análisis diseño e implementación del módulo Experticias.

Capítulo 2. Análisis y Diseño del módulo Experticias Criminalísticas.

En la solución, el subsistema Investigación Criminalística se dividió en tres módulos, llamados Experticias, Control de Investigaciones Criminalísticas y Solicitudes. Experticias cubre *a priori* alrededor del 60% del módulo, el cual se encarga de la manipulación automatizada de las respuestas de los especialistas en determinadas áreas de las ciencias criminalísticas a las solicitudes que son recibidas a través de una cadena de comunicación provista por el *software* o físicamente.

El diagrama de Casos de Uso del Sistema y la especificación de cada uno de ellos, da un punto de inicio sólido para su realización en términos de clases de análisis, de diseño y colaboraciones. Los artefactos recibidos del flujo que dio origen se encuentran íntegramente en los anexos. En este capítulo se abordará el análisis y diseño de dos casos de uso del módulo, después de la previa explicación de algunos conceptos y su aplicación en el módulo.

2.1 Análisis de la propuesta para el módulo Experticias Criminalísticas

El flujo de trabajo de Análisis es muchas veces obviado al seguir la metodología de desarrollo RUP, debido a que el Modelo de Análisis constituye una abstracción del Modelo de Diseño y muchas veces los diseñadores de clases no lo necesitan para acercarse al diseño final de la solución. Sus principales componentes son las clases del análisis que tienen tres estereotipos, denominados Interfaz, Controladora y Entidad. En dependencia de la función de una clase identificada en el dominio del problema, esta adoptará uno de los tres estereotipos. El resultado de este flujo es menos específico que el de Diseño, pero ayuda a los desarrolladores a afianzar los conocimientos de algunos conceptos y relaciones entre las entidades del sistema a implementar; por esa razón se dedicó en el proceso de producción determinado tiempo a realizar el análisis de los casos de uso a implementar, debido en parte a la inexperiencia de los diseñadores en la producción de *software* empresarial. La otra parte relevante del modelo de análisis, o sea las colaboraciones, se representaron mediante los artefactos apropiados y después del estudio de la descripción de los casos de uso y la obtención de los diagramas necesarios, los diseñadores obtuvieron el conocimiento del sistema necesario para trabajar.

2.1.1 Proceso genérico del módulo Experticias

Como parte del proceso de análisis de los casos de uso se hizo un estudio de cómo funciona el módulo; a continuación se hace una descripción general de las acciones que se suceden en el proceso fundamental del módulo, la creación y manipulación de informes de respuesta a solicitudes.

El módulo Experticias se concibió para trabajar con la precondición de que el usuario autenticado tiene los permisos para acceder a las funcionalidades requeridas. De manera general se definió una propuesta de sistema donde la comunicación entre los miembros de las diferentes entidades se realiza a través de solicitudes, respuestas a dichas solicitudes y notificaciones, tanto del sistema como de un usuario.

La integración con el resto del sistema es de vital importancia, debido a que la mayoría de estos casos de uso son una subfunción, o sea, se accede a ellos desde otro Caso de Uso, para mayor comprensión, a través de una especie de bandeja de entrada, donde cada usuario del sistema puede acceder a las comunicaciones que tiene permisos para manipular, u otros casos de uso de otros subsistemas.

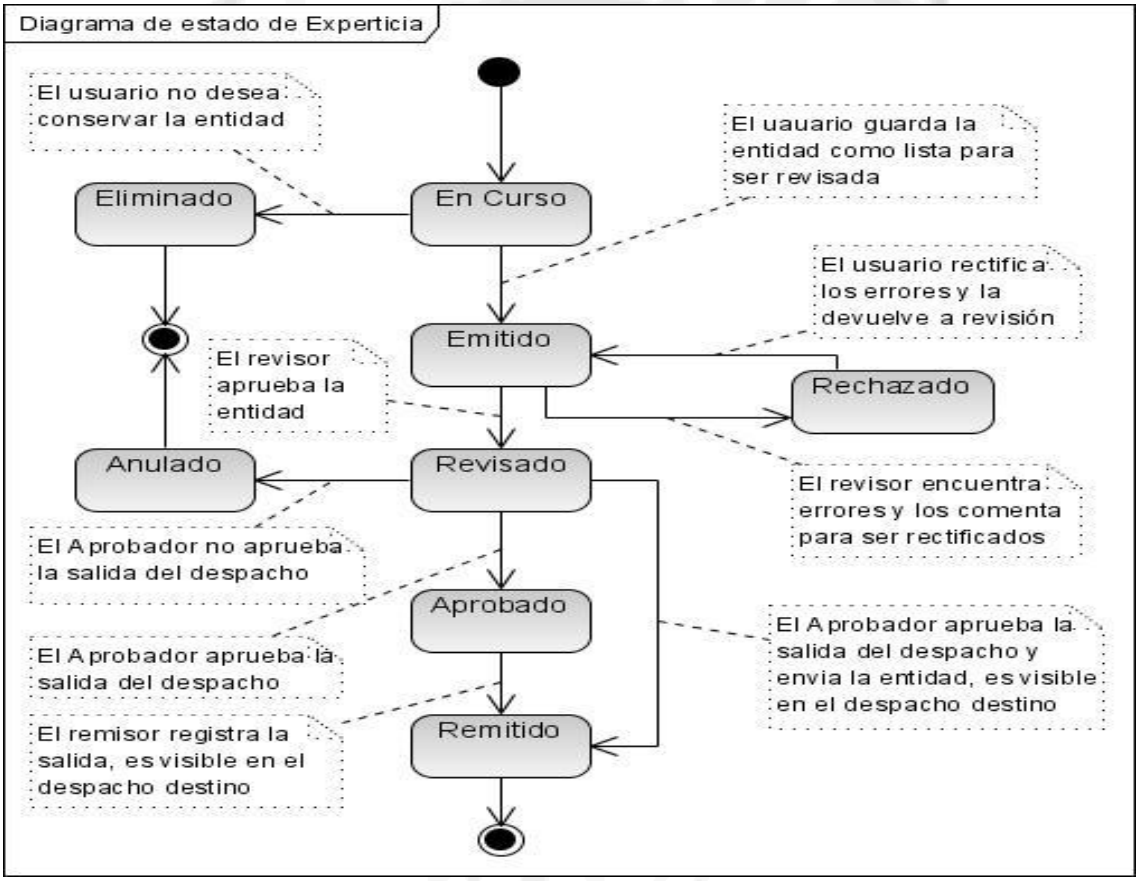

Ilustración 9: Diagrama de estado de una experticia.

El proceso genérico principal comienza cuando las solicitudes que se crean, son enviadas al departamento o dependencia que es responsable de responderlas a través de las Experticias; un jefe o responsable asigna una solicitud a un especialista en la materia, este la responde haciendo un informe

con un formato establecido y dicha Experticia pasa por una serie de estados y la manipulan personas con varios roles en el despacho, y finalmente se envía. Los especialistas son los encargados de responder las solicitudes; de ahí el término `experticia´, debido a que ellos son los expertos en sus áreas.

Este proceso es el que rige el flujo de información del módulo; cabe destacar que puede ser recursivo, o sea, la realización de una experticia puede generar nuevas solicitudes a otros despachos. Otro detalle es que cada uno de los informes y las solicitudes están asociados a un caso o acta procesal que recoge todas las actividades relativas a una investigación criminalística; en el dominio del módulo no tiene grandes implicaciones, por eso solo es válido mencionarlo sin llegar a detalles que desvíen la atención del hilo principal.

Experticias Criminalísticas cuenta con 15 casos de uso y todos entran en la clasificación ‘Gestionar’, que comparten buena parte de la estructura y funcionalidad; están compuestos por los escenarios ‘Incluir’, ‘Modificar’, ‘Vista Previa’, ‘Ver’ y ‘Eliminar’, con un número variable de transacciones en cada escenario, a continuación se listan:

- Gestionar Experticia de Retrato Hablado. En este Caso de Uso se elabora el retrato hablado en función de la información que suministra el agraviado o testigo y se guarda en un informe.
- Gestionar Experticia de Reconstrucción de Hechos. En este Caso de Uso se obtiene como resultado un Informe sobre la reconstrucción en el sitio del suceso de un hecho punible relacionado con trayectorias de balas de las arma de fuego relacionadas en el hecho.
- Gestionar Experticia de Trayectoria Intraorgánica. Este Caso de Uso se efectúa con el objetivo de enviar un informe con el estudio efectuado a un protocolo de autopsia de un cadáver.
- Gestionar Experticia de Siniestro. Permite a los expertos plasmar el análisis criminalístico pertinente escribiendo las causas que dieron origen al siniestro a través de la elaboración del informe.
- Gestionar Experticia Contable. Este Caso de Uso se efectúa con el objetivo de crear un Informe acerca del análisis de la documentación de una institución o persona que esté sujeta a una investigación determinada.
- Gestionar Experticia de Levantamiento Planimétrico. Este Caso de Uso se efectúa con el objetivo de elaborar el plano de un sitio determinado a partir de las observaciones obtenidas en el lugar y dejar constancia en un informe.
- Gestionar Experticia de Microscopía Electrónica. Permite elaborar una experticia para emitir los resultados del análisis de muestras para identificar sustancias que estén presentes en una persona o evidencia determinada.

- Gestionar Experticia de Inspección Técnica de Robo de Banco. En este Caso de Uso se obtiene como resultado un Informe sobre la inspección realizada a una entidad bancaria en la cual se ha producido un robo.
- Gestionar Experticia de Inspección Técnica de Inmuebles. En este Caso de Uso se obtiene como resultado un Informe sobre la inspección realizada a un inmueble en el cual se recogen la descripción de lo efectuado por parte de los expertos y además un montaje fotográfico con imágenes del inmueble inspeccionado.
- Gestionar Experticia de Inspección Técnica Especializada. Mediante este Caso de Uso los entes investigativos del CICPC y otras entidades reciben información completa de todo lo referente a un lugar donde se detectó un hecho delictivo.
- Gestionar Experticias de Avalúos. A partir de este Caso de Uso se obtiene un Informe de los avalúos reales de bienes muebles e inmuebles o regulaciones prudenciales.
- Gestionar Experticia de Vehículos. En este Caso de Uso se obtiene un Informe acerca de la experticia realizada a un vehículo para determinar si sus seriales han sido o no alterados.
- Gestionar Experticia de Identificación de Individuo. Este Caso de Uso se efectúa con el objetivo de obtener un Informe sobre la identificación de un individuo, registrándose en este los datos del individuo como resultado del proceso de identificación.
- Gestionar Experticia de Trayectoria Balística. En este Caso de Uso se obtiene como resultado un Informe sobre lo ocurrido en el sitio del suceso de un hecho punible relacionado con trayectorias balísticas de un arma de fuego.

Estos casos de uso son fáciles de comprender debido a la cantidad de puntos en común que tienen; de manera general permiten al experto técnico elaborar un Informe como respuesta a determinada solicitud recibida; estas solicitudes presentan especificidades distintas según su tipo y tienen diferentes motivos en cada instancia, por esta razón se responde a través de diferentes informes.

Cuando el actor quiere incluir una Experticia como respuesta a una solicitud, el sistema muestra de forma predeterminada algunos datos de la Solicitud de Experticia Criminalística seleccionada y brinda la posibilidad de introducir nuevos datos y las impresiones y resultados específicos de la experticia efectuada. El sistema le permite seleccionar las acciones a realizar, terminando el Caso de Uso con la realización de una de las acciones básicas crear, actualizar o borrar, o bien abandonando el Caso de Uso sin realizar acción.

2.2 Diseño del módulo Experticias Criminalísticas

2.2.1 Algunos Patrones de Diseño utilizados

La capa de presentación del SIIPOL está implementada sobre la base del *framework* JSF; este es un exponente del patrón arquitectónico MVC, utilizado para el diseño de aplicaciones con sofisticadas interfaces de usuario. La lógica de una interfaz de usuario cambia con más frecuencia que los almacenes de datos y la lógica de negocio. Si se mezclan los componentes de interfaz y de negocio, la consecuencia será que, ante la necesidad de cambiar la interfaz, habrá que modificar trabajosamente los componentes de negocio. Ello implica mayor trabajo y más riesgo de error (Proactiva). En este esquema el modelo es el responsable de acceder a la capa de almacenamiento de datos, de definir las reglas de negocio (la funcionalidad del sistema). El controlador es responsable de recibir los eventos de entrada (un clic, un cambio en un campo de texto, etc.) y contiene reglas de gestión de eventos, y las vistas son responsables de recibir datos del modelo y mostrarlos al usuario.

JSF implementa el patrón de diseño ***Page Controller*****,** del cual en el SIIPOL se utiliza la variante de tener un controlador por cada página; en este caso el controlador es el *bean* de respaldo de la página que recibe la petición y la procesa invocando al modelo, y actualiza la página o redirecciona para otra vista. Dicha variante también incluye el uso de una clase base para los controladores, como la clase *BaseBean* del proyecto, que evita la replicación de código; este patrón permite la reusabilidad y la extensibilidad del diseño, a la vez que es muy simple de comprender o utilizar (20).

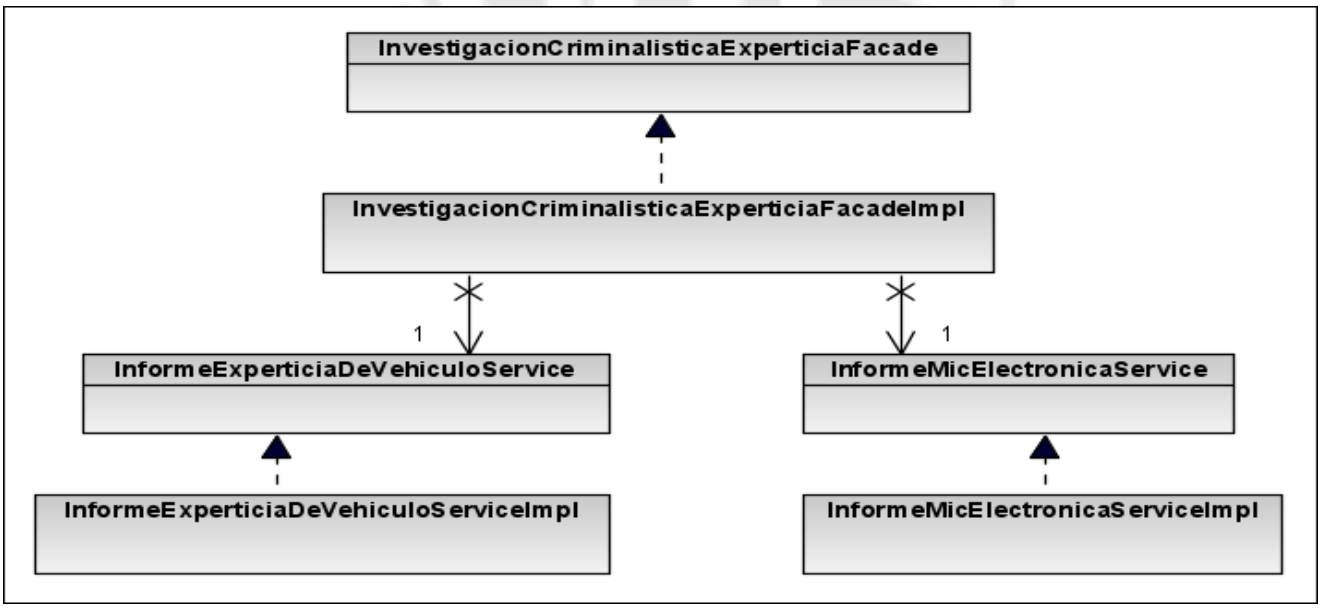

Ilustración 10: Diagrama de clases representativo de la capa Lógica de Negocio.

En la capa de Lógica de Negocio *Spring Framework* soporta un ambiente de trabajo seguro y provee un ambiente flexible. Experticias, como el resto de los módulos, está compuesto por una clase fachada y varias clases de servicios en su capa media, además de archivos de configuración de *Spring* y clases utilitarias.

La clase fachada del módulo es *investigacionCriminalisticaExperticiaFacade* y es la aplicación del patrón de diseño ***Facade***, que sirve para proveer una interfaz unificada y sencilla que haga de intermediaria entre el cliente (clase o conjunto de ellas) y una interfaz o grupo de interfaces más complejas; dichas interfaces son muy disímiles en el módulo, pues representan las clases de servicio.

Spring Framework implementa otros patrones como los de la familia ***Factory*****,** que consiste en utilizar una clase constructora abstracta con varios métodos definidos para la creación de clases de una jerarquía**.** En el proyecto se utiliza este recurso, dejando a *Spring* la creación y gestión de las clases de servicio, las fachadas y las clases de acceso a datos a los que se accede de manera transparente. Los servicios se instancian a través de un ***Service Locator*****,** otro patrón de diseño que provee un punto de acceso centralizado a determinados servicios.

La arquitectura de *Spring* está basada en patrón de diseño llamado ***Dependency Injection***, definido como la manera de externalizar la creación y el manejo de las dependencias de los componentes (21).

Otro patrón muy útil que utiliza el proyecto en combinación con los demás y que provee *Spring* es el ***Proxy*****,** que se usa cuando es necesario crear objetos que consumen muchos recursos, pero no es deseable instanciarlos excepto que el cliente lo solicite o se cumplan otras condiciones determinadas (22). Es el caso de las múltiples fachadas y sus referencias, de las que se carga a través de configuraciones predefinidas de *Spring* un objeto "*Proxy*" que no es realmente el objeto solicitado, sino una representación, y cuyo objetivo, o sea la clase a la que representa, se instancia completamente en el momento que se vaya a utilizar.

La capa de acceso a datos de SIIPOL utiliza el patrón DAO (*Data Access Object*); el problema que viene a resolver este patrón es el de contar con diversas fuentes de datos (base de datos, archivos, servicios externos, etc.). Así se encapsula la forma de acceder a la fuente de datos. Este patrón surge históricamente de la necesidad de gestionar una diversidad de fuentes de datos, aunque su uso se extiende al problema de encapsular no sólo la fuente de datos, sino también ocultar la forma de acceder a los datos. Se trata de que el *software* cliente, o sea los servicios de la aplicación, se centre en los datos que necesita y se olvide de cómo se realiza el acceso a los datos o de cuál es la fuente de almacenamiento; así un cambio en el origen de datos o en la manera de recuperarlos no afecta la capa superior siempre que se implemente la interfaz correspondiente.

2.2.2 Representación de clases externas importantes

Esta sección muestra algunas clases del proyecto a las cuales está muy ligado el funcionamiento del módulo; entre ellas se encuentran la clase Informe, que es la clase base de la cual heredan todos los tipos de informes que se manejan en el proyecto, y contiene los datos básicos de una entidad de este tipo y que a su vez es hija de Diligencia; representa la constancia de cualquier actividad realizada en el contexto del negocio estudiado. La misma tiene una relación de asociación unidireccional con la entidad de tipo Solicitud que dio origen a su creación.

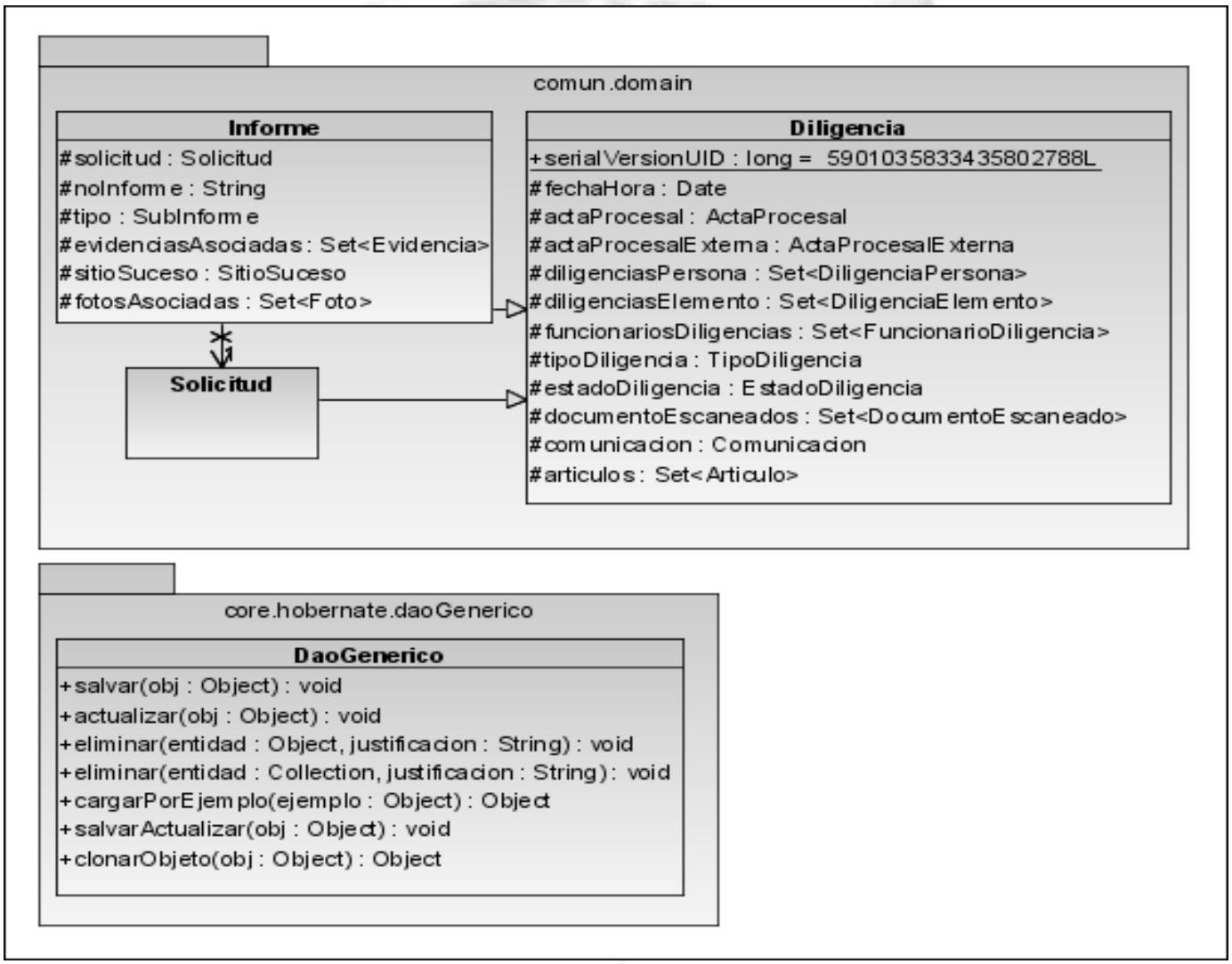

Ilustración 11: Diagramas de clases de importancia para el módulo

La clase *DaoGenerico* es la base de la implementación del patrón de diseño DAO, y encapsula las funcionalidades repetibles en cualquier clase de acceso a datos del sistema.

2.3 Caso de Uso de Muestra: Gestionar Experticia de Identificación de Individuo

En esta sección se muestran los principales resultados del análisis del Caso de Uso seleccionado; seguidamente se introduce al diseño del Caso de Uso, de manera ilustrativa, utilizando diagramas generados por la metodología RUP, conjuntamente con descripciones e imágenes.

2.3.1 Análisis de Caso de Uso: Gestionar Experticia de Identificación de Individuo

Descripción resumida de los eventos

El Caso de Uso se inicia cuando el Experto Técnico accede a realizar una acción sobre un Informe de respuesta a una Solicitud de Identificación de Individuo. En caso de que el actor acceda a la sección donde puede introducir los resultados de la experticia, el sistema muestra de forma predeterminada algunos datos de la Solicitud de Experticia Criminalística y permite entrar otros datos que formarán parte de la experticia. El sistema permite seleccionar la opción de tener una vista previa del informe confeccionado, enviar el informe para que sea revisado o guardarlo temporalmente para en un futuro acceder a los datos de este en su bandeja de borradores. En caso de que el actor acceda a la opción de ver los detalles del Informe de Identificación de Individuo, el sistema muestra su contenido y brinda además la posibilidad de eliminarlo, o modificar sus datos en caso ser necesario si el actor es el responsable del informe, imprimir o exportar a formato PDF.

Entidades:

- Funcionario.
- Informe de Identificación de Individuo.
- Acta Procesal.

Requisitos Funcionales:

- Mostrar datos de la Solicitud de Experticia.
- Incluir datos asociados al resultado de experticia.
- Mostrar vista previa del Informe de Identificación de Individuo.
- Mostrar los datos de un Informe de Identificación de Individuo.
- Modificar datos del Informe de Identificación de Individuo.
- Guardar el Informe de Identificación de Individuo.
- Exportar a PDF.

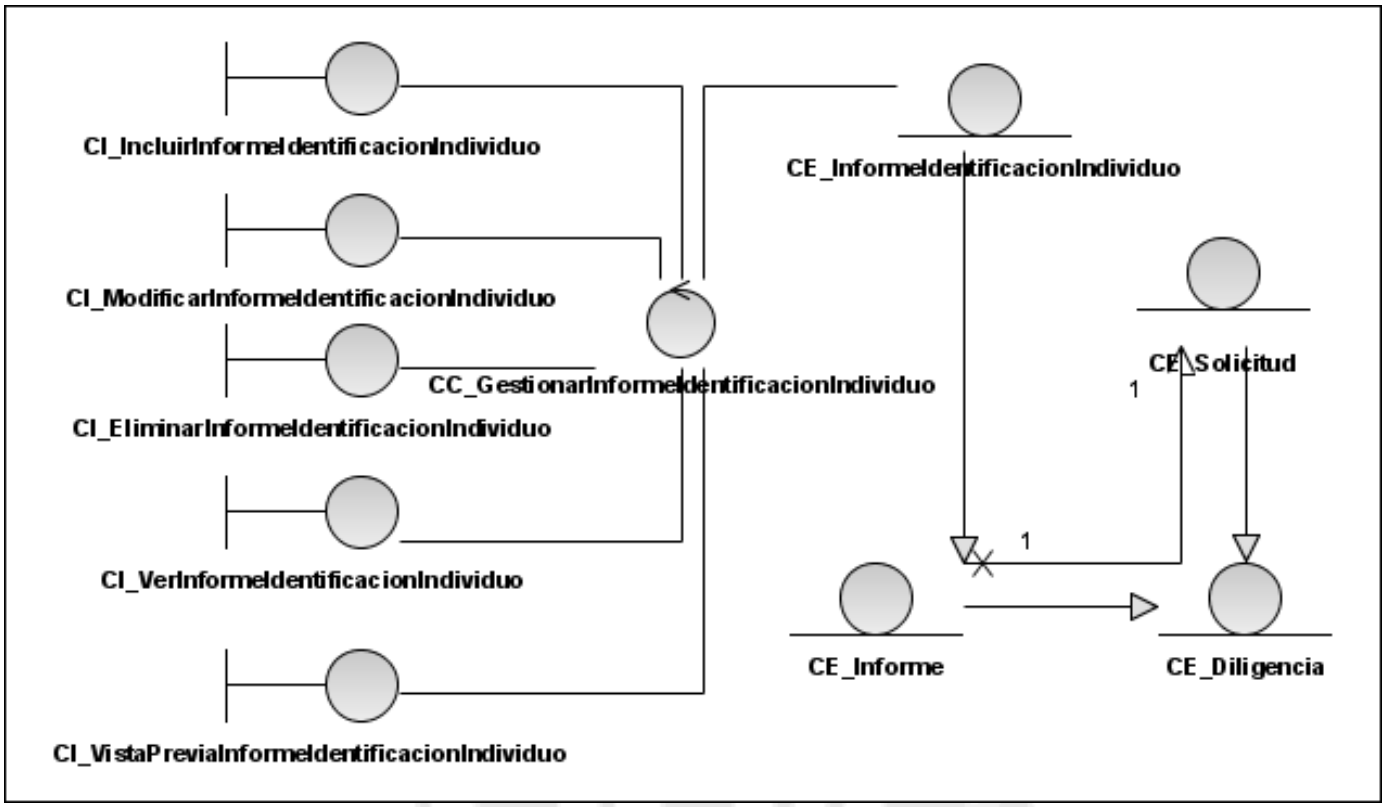

Ilustración 12: Diagrama de clases del análisis del Caso de Uso Gestionar Experticia de Identificación de Individuo

2.3.2 Diseño de Caso de Uso de Muestra: Gestionar Experticia de Identificación de Individuo

En esta sección se muestran los principales artefactos generados en el proceso de diseño del Caso de Uso Gestionar Experticia de Identificación de Individuo. Primeramente se muestra una porción del diagrama de clases del diseño que no pertenece al módulo Experticias Criminalísticas, pero sí tiene estrecha relación con este; son clases comunes a varios subsistemas que, por ende, se encuentran situadas fuera del subsistema Investigación Criminalística. A continuación de la representación gráfica de este entregable de RUP se muestra una vista del diagrama de clases del módulo que componen la capa de Lógica de Negocio y la capa de Acceso a Datos, y seguidamente se divide la capa de presentación relativa al Caso de Uso en escenarios para su mejor organización y comprensión. Por último se incluye el diagrama de secuencia del contrato entre paquetes que representa la colaboración de los paquetes dentro del Caso de Uso en cuanto a intercambio de mensajes. El objetivo es mostrar cómo queda diseñado el módulo de manera ilustrativa. Los entregables de RUP en este flujo se encuentran íntegramente en los anexos.

El diagrama de clases que agrupa la Lógica de Negocio y Acceso a Datos en la Ilustración 13 evidencia la estructura del diseño utilizado. La interfaz *InvestigacionCriminalisticaExperticiasfacade* es un ejemplo de promoción de interfaz, y no hace más que repetir los métodos de clases de servicio de las cuales es fachada; esta es una interfaz de Java y es implementada por una clase de igual nombre con el sufijo *Impl*. La interfaz de servicio *InformeIdentificacionDeIndividuoService* tiene su respectiva implementación, y provee las funcionalidades que son invocadas por la fachada englobando la lógica correspondiente sobre la clase persistente *InformeIdentificaciónIndividuo*, para lo cual se auxilia, en caso de acceder a la base de datos, de la interfaz de acceso a datos; esta también cuenta con su implementación.

De la Ilustración 14 a la 18 para hacer más comprensible el Caso de Uso se muestra la capa de presentación dividida en escenarios; no se grafica por razones de espacio la asociación unidireccional que existe entre el *bean* manejado y la fachada, en la cual el *bean* conoce la fachada pero no viceversa.

Los escenarios Modificar Informe e Incluir Informe tienen diseño y codificación muy similares, el diagrama de clases muestra la pagina servidora .jsp que será la encargada de generar el código HTML, que importa el fichero *java script* para las validaciones del lado del cliente, las peticiones son controladas y distribuidas por el *Faces Servlet*, que reenvía hacia la página responsable, entonces se ejecuta el código asociado a la llamada y se reflejan los cambios en la vista.

La mayor diferencia que existe entre el escenario Modificar e Incluir es que, lógicamente, debe cargarse el informe a modificar en dicho código, además de cumplir ciertas restricciones de campos no modificables.

Los escenarios Ver y Vista Previa son notablemente parecidos también; la idea de estos es la de mostrar los datos asociados al informe; la diferencia es que no muestran los mismos datos de dicha experticias ni son presentados de la misma manera.

El escenario Eliminar Informe está compuesto a grandes rasgos por clases similares, con mecanismos parecidos; la función principal es mostrar los datos del informe a eliminar, permitir entrar una justificación y eliminar el informe.

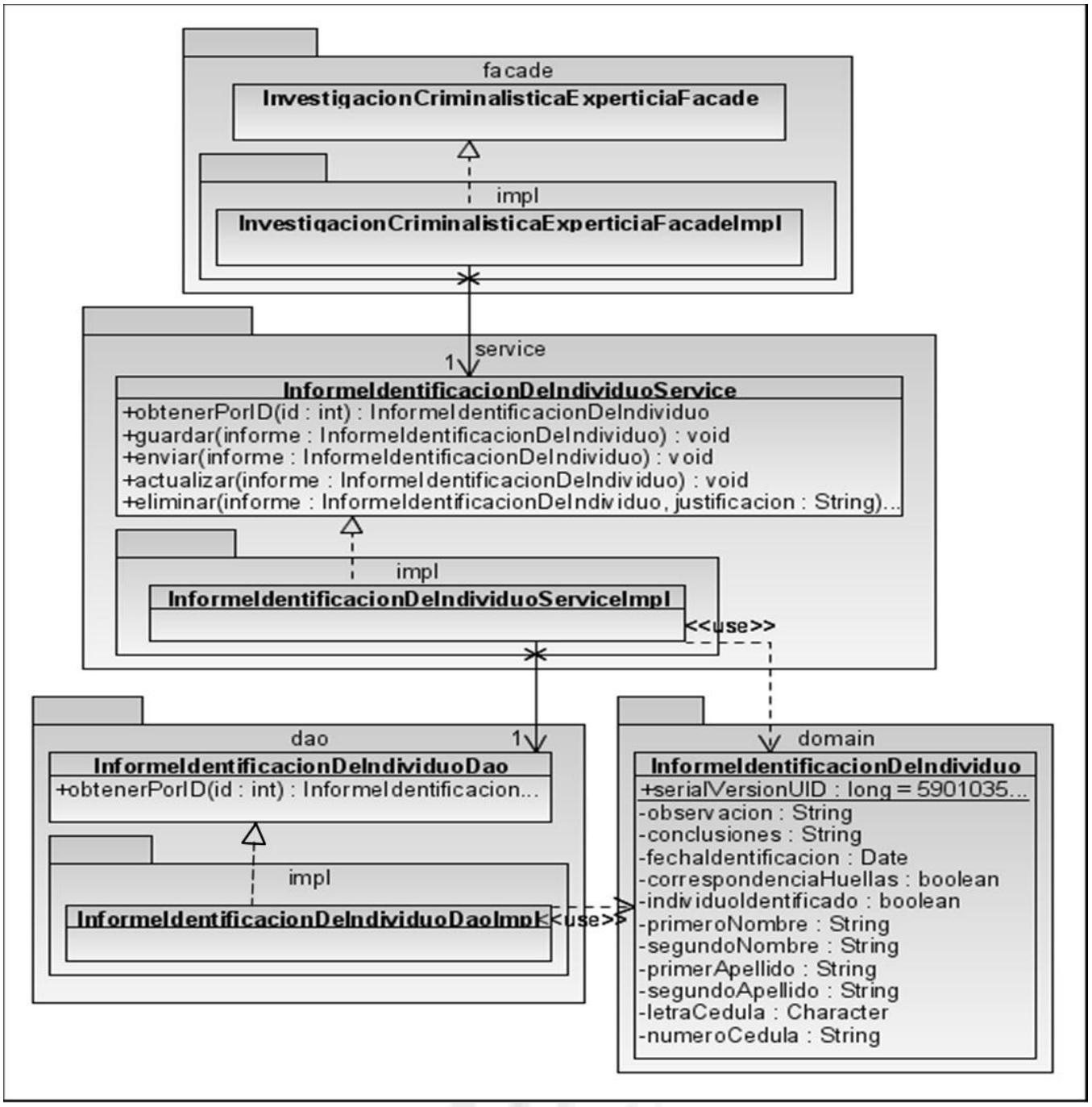

Ilustración 13: Diagrama de clases del diseño de Lógica de Negocio, Caso de Uso Gestionar Experticia de Identificación de Individuo.

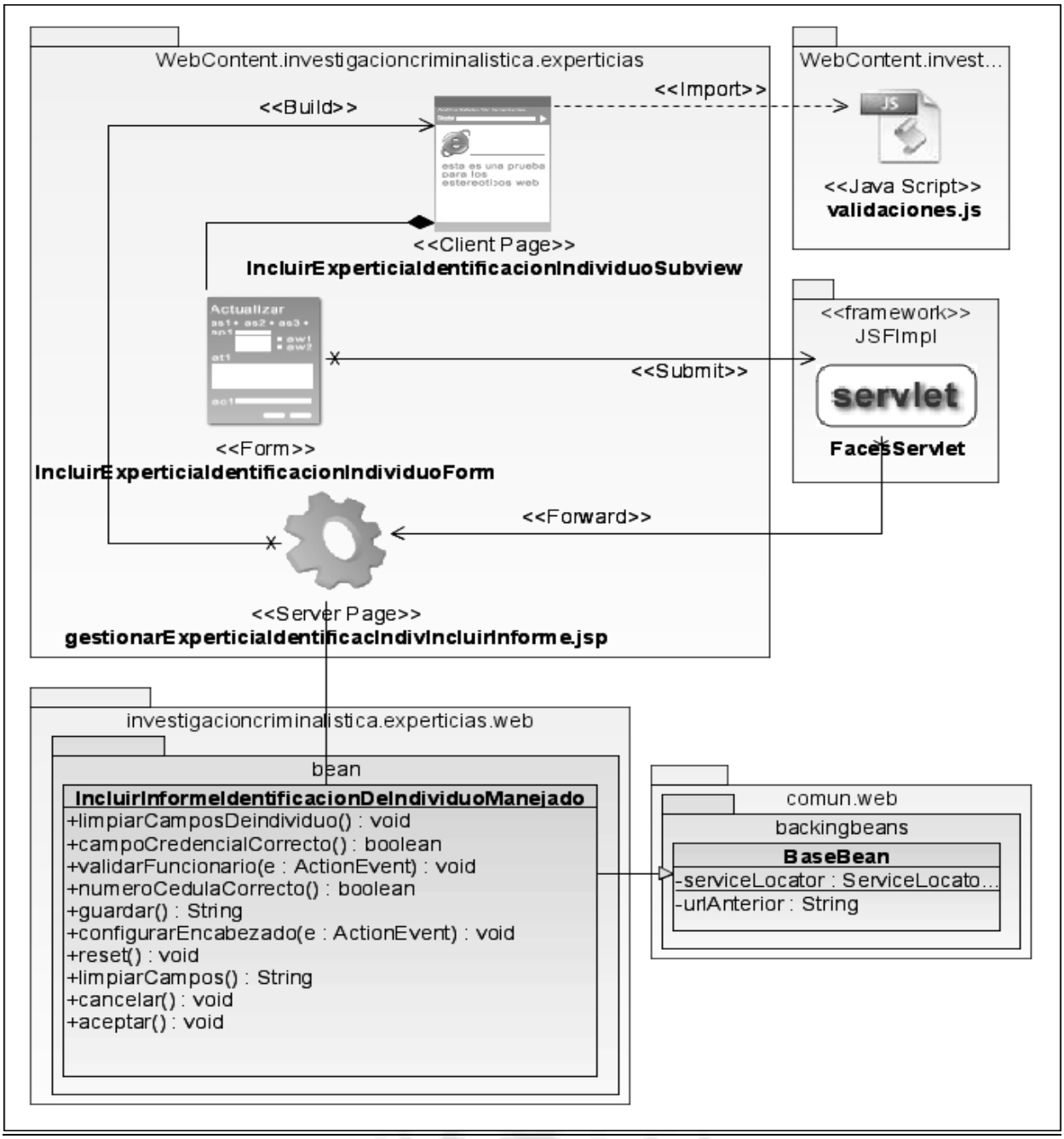

Ilustración 14: Diagrama de clases del diseño de Presentación, escenario Incluir Experticia de Identificación de Individuo.

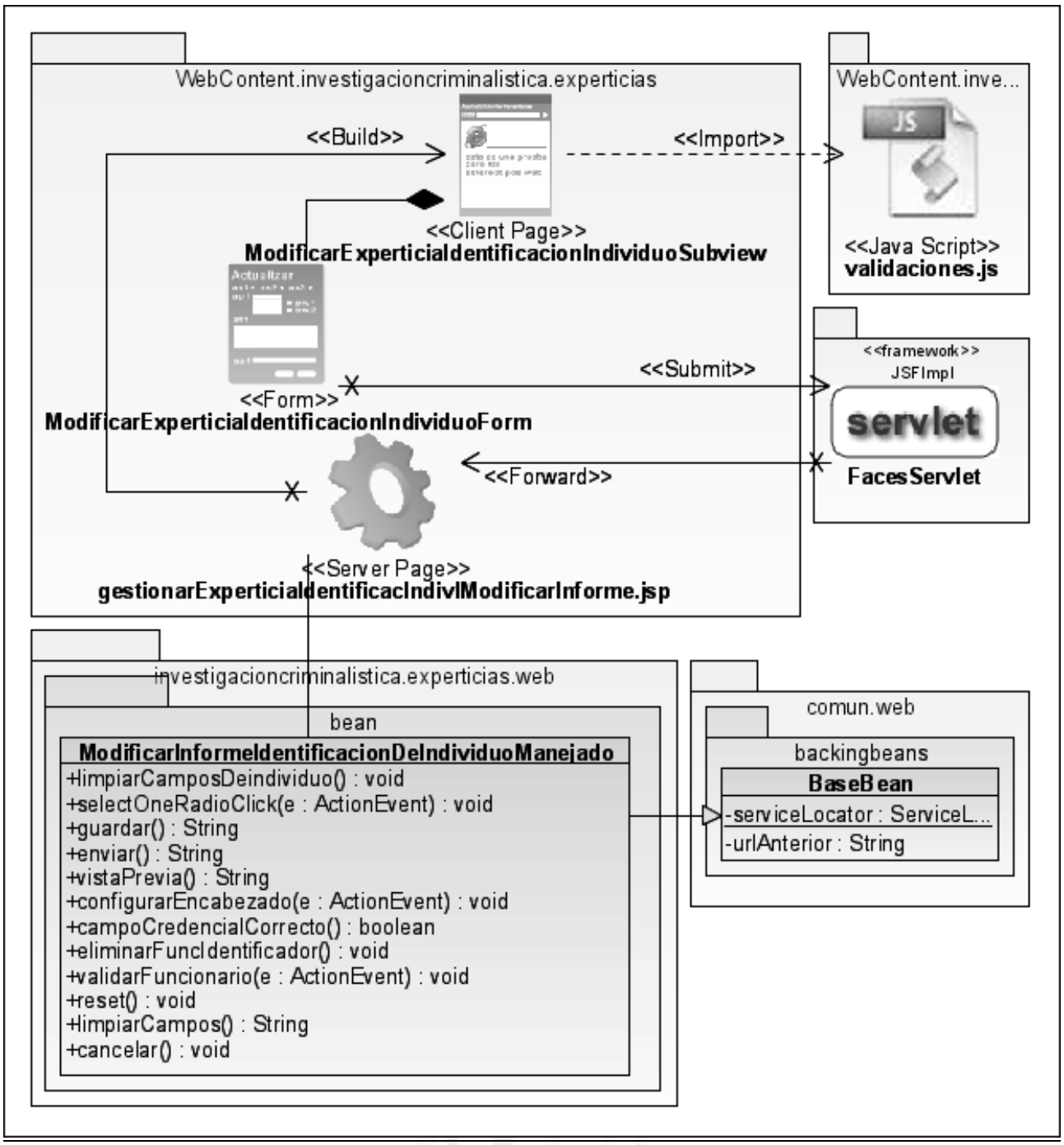

Ilustración 15: Diagrama de clases del diseño de Presentación, escenario Modificar Experticia de Identificación de Individuo.

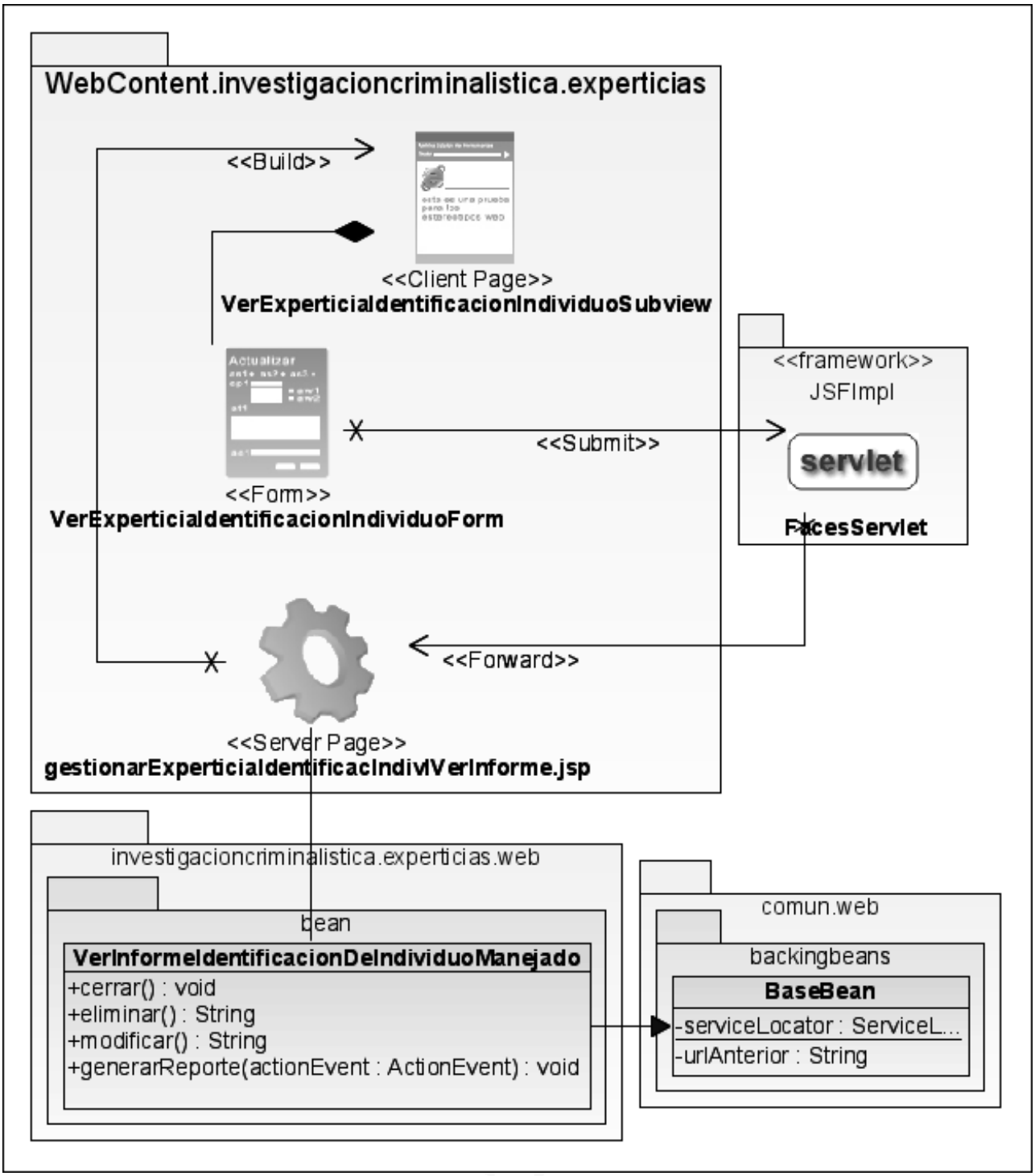

Ilustración 16: Diagrama de clases del diseño de Presentación, escenario Ver Experticia de Identificación de Individuo.

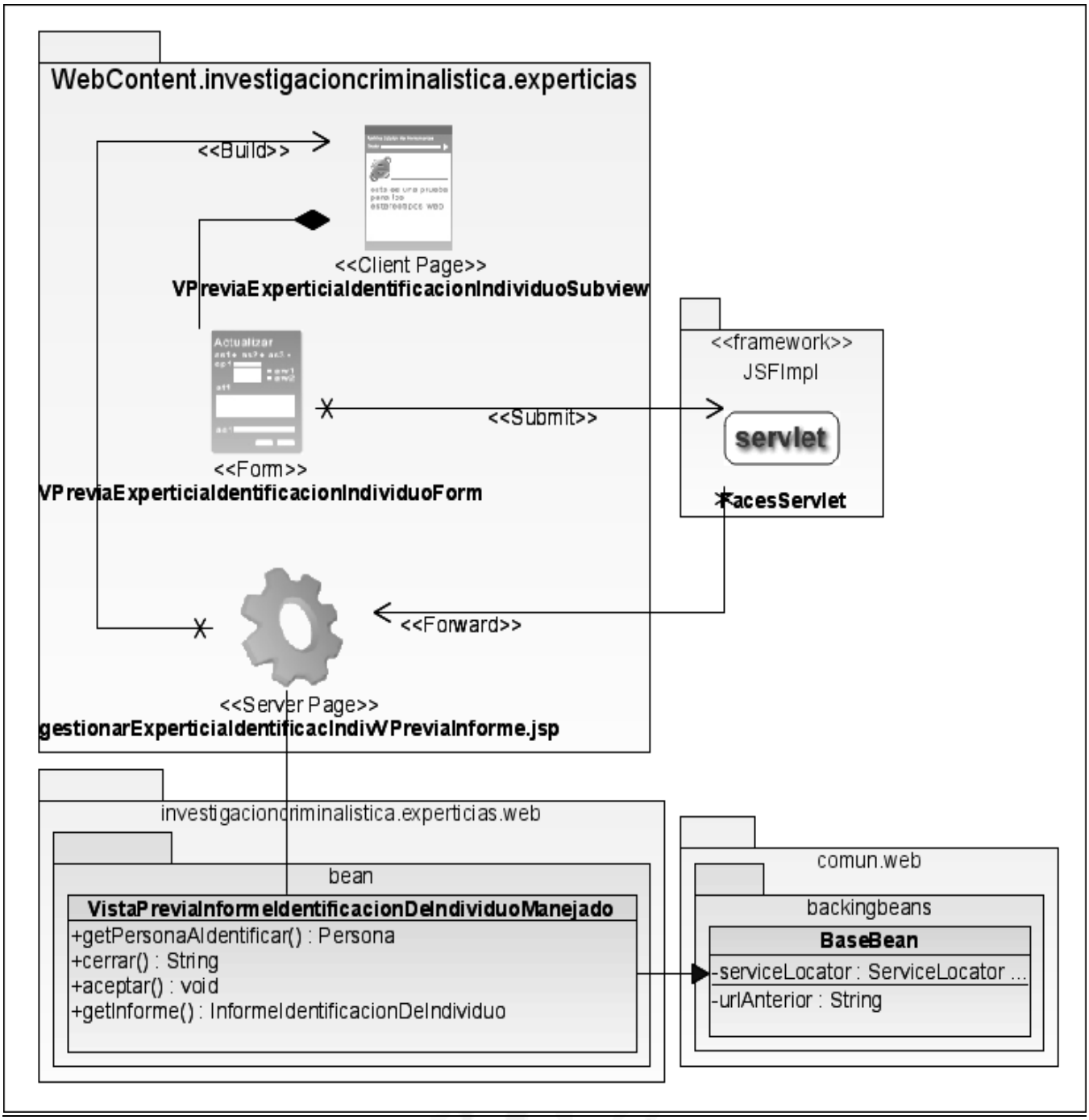

Ilustración 17: Diagrama de clases del diseño de Presentación, escenario Vista Previa Experticia de Identificación de Individuo.

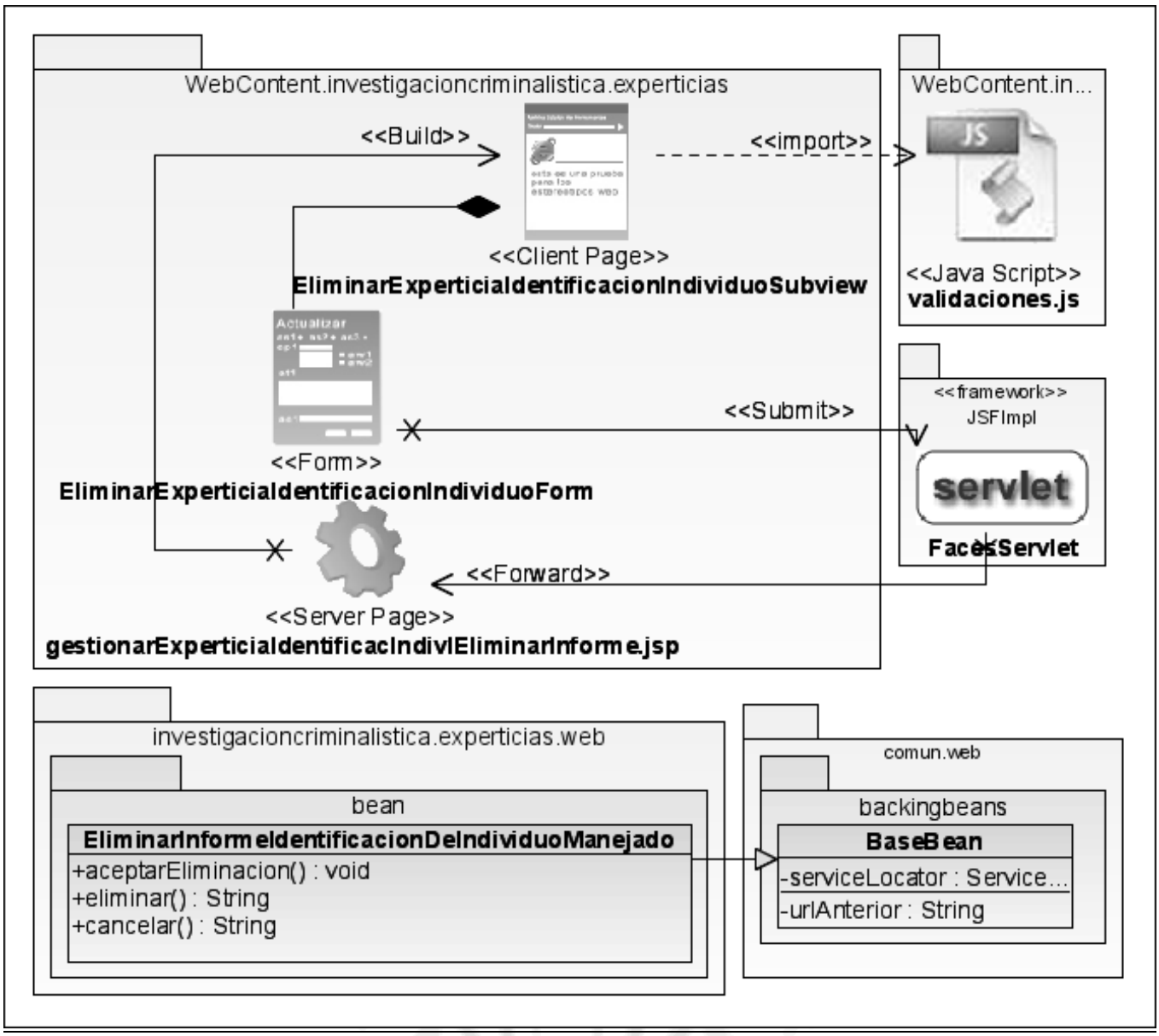

Ilustración 18: Diagrama de clases del diseño de Presentación, escenario Eliminar Experticia de Identificación de Individuo.

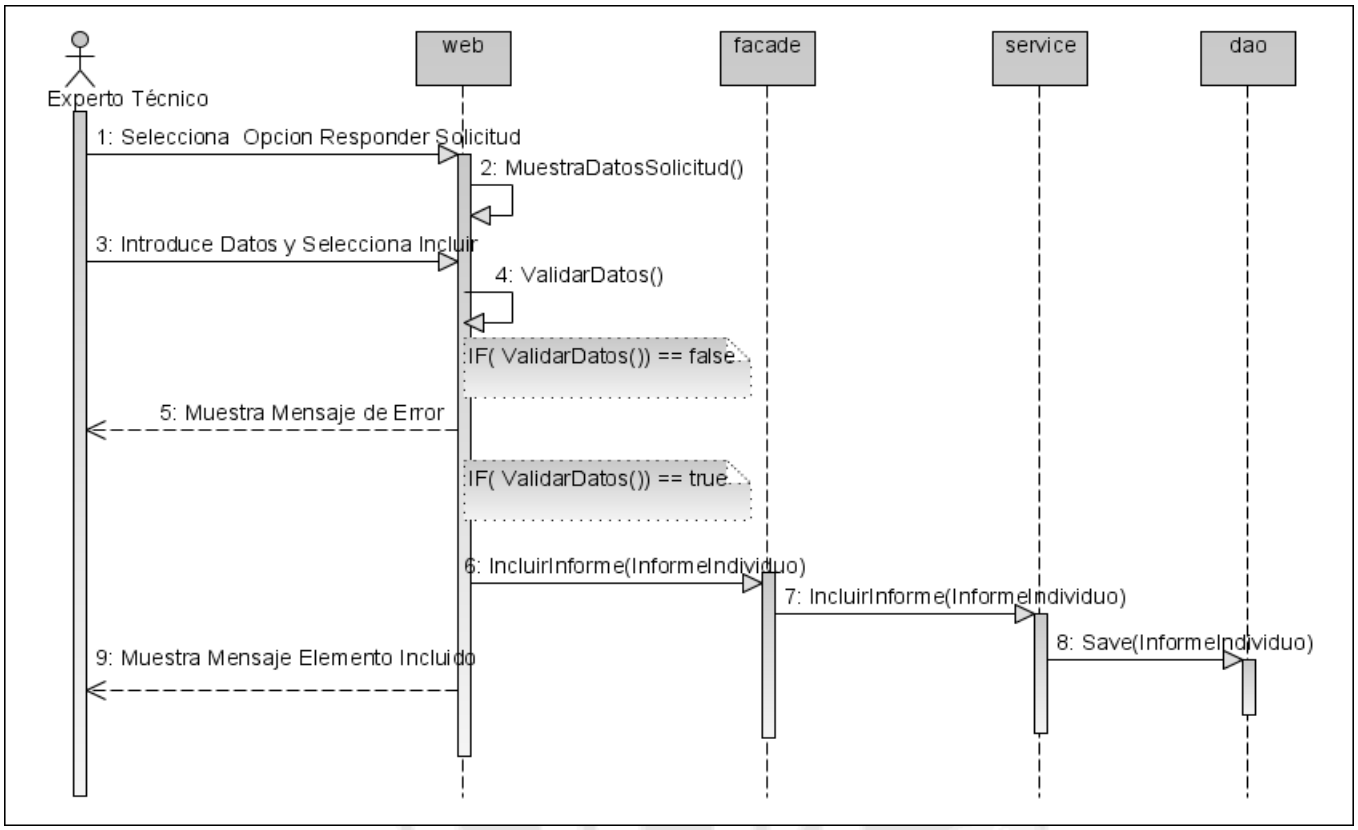

Ilustración 19: Diagrama de secuencia de contrato entre paquetes, escenario Incluir Experticia de Identificación de Individuo.

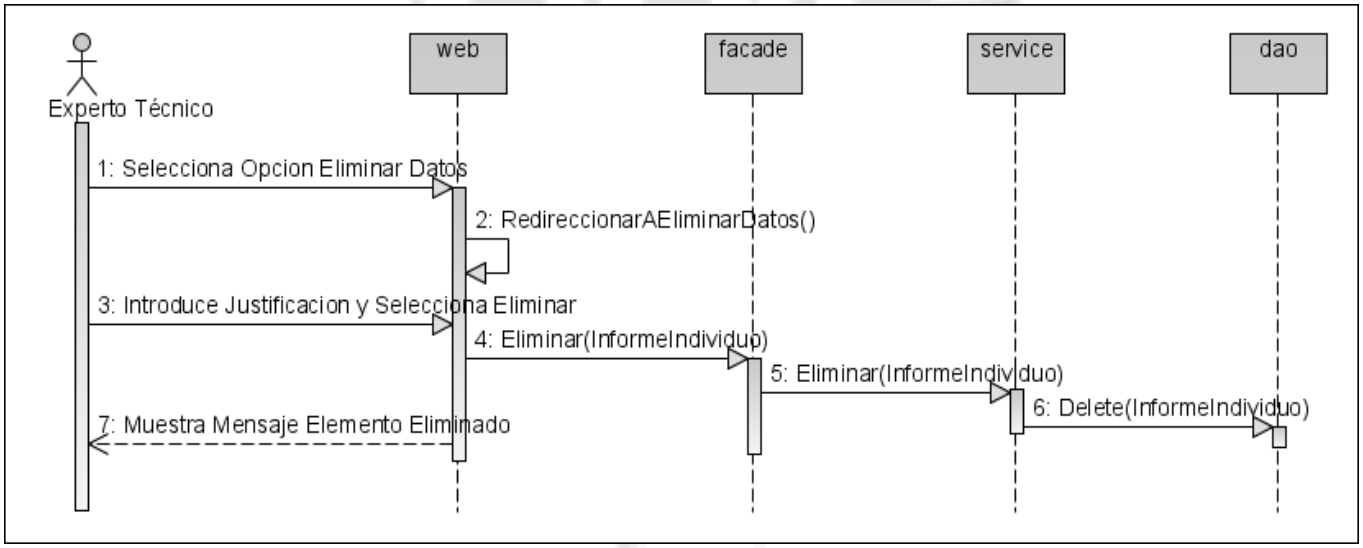

Ilustración 20: Diagrama de secuencia de contrato entre paquetes, escenario Eliminar Experticia de Identificación de Individuo.

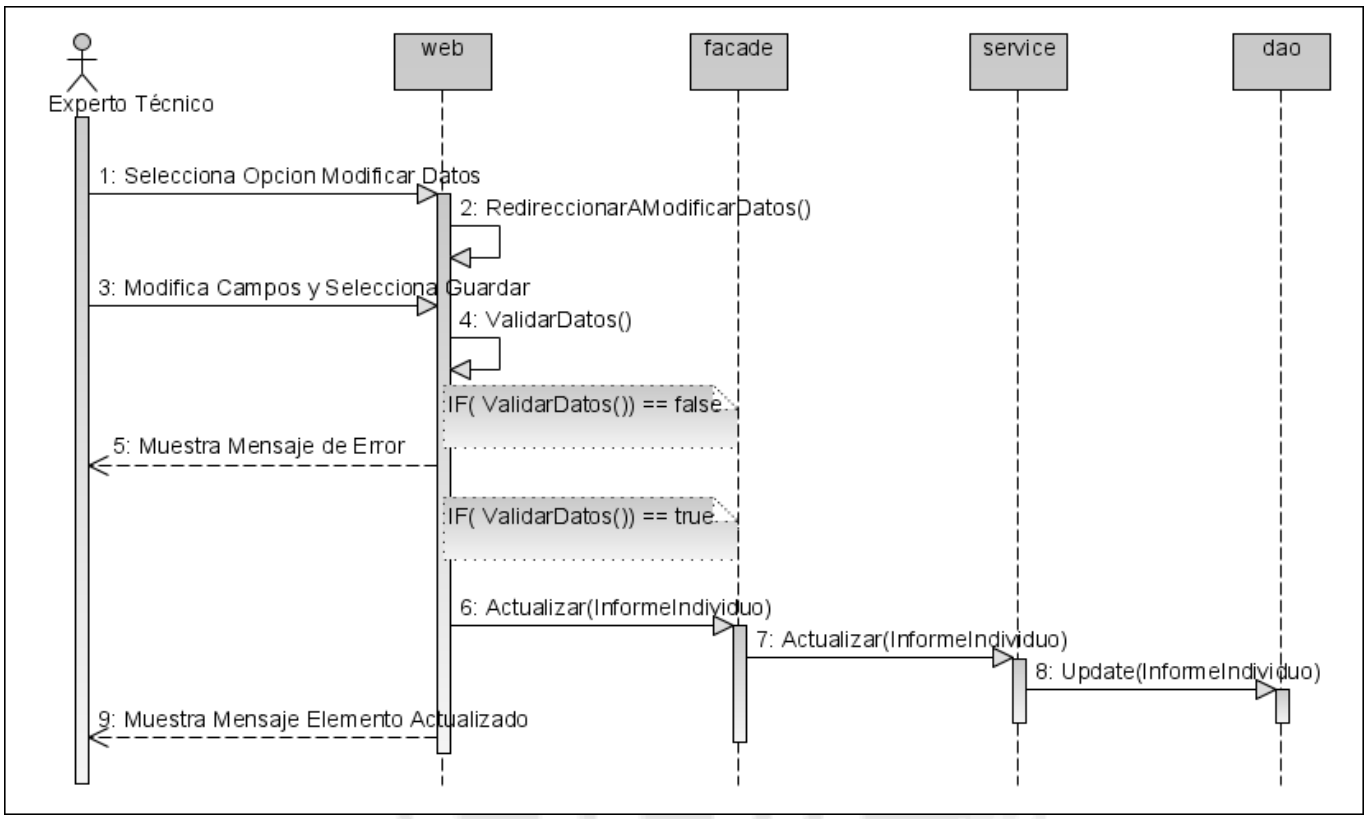

Ilustración 21: Diagrama de secuencia de contrato entre paquetes, escenario Modificar Experticia de Identificación de Individuo.

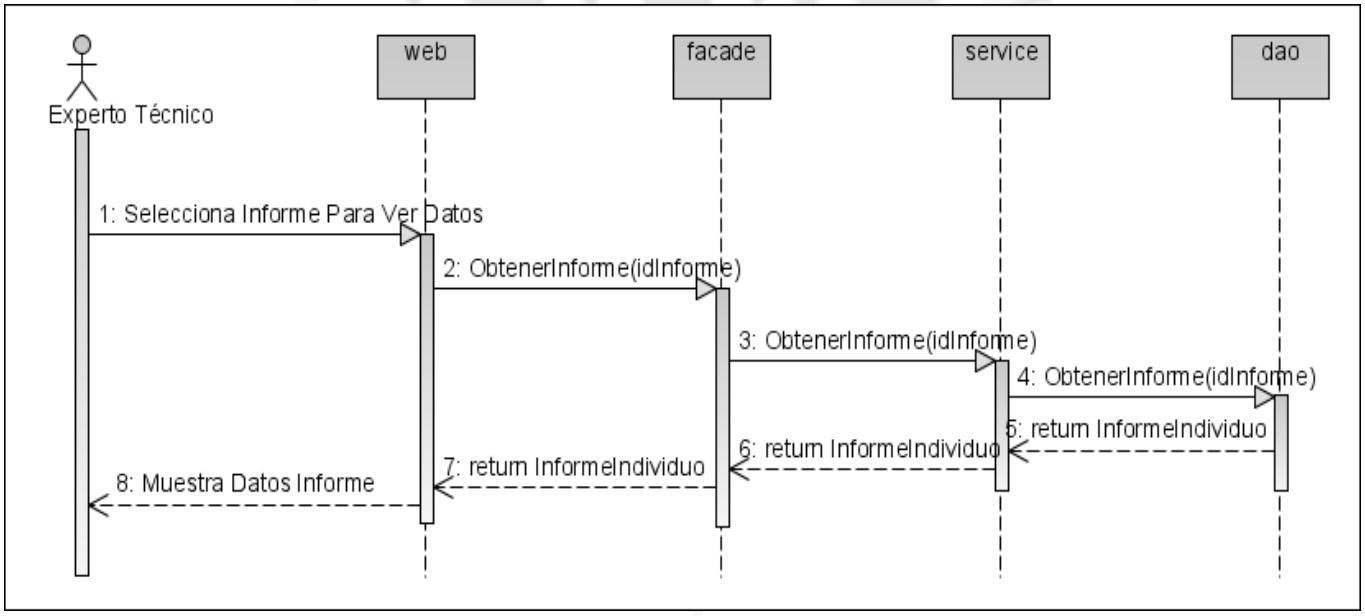

Ilustración 22: Diagrama de secuencia de contrato entre paquetes, escenario Ver Experticia de Identificación de Individuo.

2.3.3 Descripción de las clases involucradas en el Caso de Uso.

En esta sección se describen los elementos fundamentales de cada clase que participa en la realización en términos de diseño del Caso de Uso.

Nombre de la clase: IncluirInformeIdentificacionDeIndividuoManejado

Descripción: Esta clase es el respaldo de la página encargada de presentar una interfaz al usuario que permita incluir un nuevo Informe de Identificación de Individuo.	
Capa de la Arquitectura: Presentación	
Ubicación: vnz.cicpc.investigacioncriminalistica.experticias.web.bean	
Atributo:	***Tipo***:
solicitudAsociada	Solicitud
informeIdentificacionDeIndividuo	InformeIdentificacionDeIndividuo
listaArticulos	List<Articulo>
criminalisticaExperticiasFacade	InvestigacionCriminalisticaExperticiasFacade
investigacionPenalFacade	InvestigacionPenalFacade
consultarArticuloManejado	ConsultarArticuloManejado
vistaPreviaManejado	VistaPreviaInformeIdentifDeIndividuoManejado
Responsabilidades:	
Nombre: LimpiarCamposDeindividuo() ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad se encarga de dejar los campos relativos a la persona a identificar en blanco o el valor que lo represente.
Nombre: campoCredencialCorrecto() ***Tipo de retorno***:boolean	***Descripción*:** Esta funcionalidad se encarga de validar si el número de credencial introducida es correcto.
Nombre: numeroCedulaCorrecto() ***Tipo de retorno***:boolean	***Descripción*:** Esta funcionalidad se encarga de validar si el número de cédula introducida es correcto, obteniendo soporte para esta acción del subsistema Análisis de Información.
Nombre: validarFuncionario(ActionEvent e) ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad se encarga de validar si los datos del funcionario son correctos, obteniendo soporte para esta acción del subsistema Análisis de Información.
Nombre: guardar() ***Tipo de retorno***:String	***Descripción*:** Esta funcionalidad se encarga de guardar en la base de datos el informe introducido, con el estado *En Curso,* lo que da posibilidad de que sea modificado más tarde. La cadena obtenida en el retorno del método es utilizada para la navegación.
Nombre: aceptar() ***Tipo de retorno***:String	***Descripción*:** Esta funcionalidad se encarga de guardar en la base de datos el informe introducido, con el estado *Emitido,* lo que no da posibilidad de que sea modificado más tarde. La cadena obtenida en el retorno del método es utilizada para la navegación.
Nombre: reset() ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad se encarga de borrar el informe completamente y crear uno nuevo en el acto, en términos programáticos le reserva memoria a un nuevo Informe, perdiendo la referencia al primero.

Nombre:configurarEncabezado(ActionEvent e) ***Tipo de retorno***:void	***Descripción*****:** Esta funcionalidad se encarga de navegar al Caso de Uso extendido configurar encabezado.
Nombre: cancelar() ***Tipo de retorno***:void	***Descripción*****:** Esta funcionalidad regresa a la página que invocó al escenario del Caso de Uso.

Nombre de la clase: VerInformeIdentificacionDeIndividuoManejado

Descripcion: Esta clase es el respaldo de la página encargada de presentar una interfaz al usuario que permita ver los datos de un informe de identificación de individuo.	
Capa de la Arquitectura: Presentación	
Ubicación: vnz.cicpc.investigacioncriminalistica.experticias.web.bean	
Atributo:	***Tipo***:
informeIdentificacionDeIndividuo	InformeIdentificacionDeIndividuo
listaArticulos	List<Articulo>
criminalisticaExperticiasFacade	InvestigacionCriminalisticaExperticiasFacade
consultarArticuloManejado	ConsultarArticuloManejado
Responsabilidades:	
Nombre: generarReporte(ActionEvent e) ***Tipo de retorno***:void	***Descripción*****:** Esta funcionalidad se encarga de crear una instancia de la clase utilitaria *GenerarReporte* y pasarle los datos del informe para generar un reporte en formato PDF.
Nombre: modificar() ***Tipo de retorno***:String	***Descripción*****:** Esta funcionalidad reenvía para el escenario Modificar del Caso de Uso, previamente le pasa los parámetros necesarios para esta acción.
Nombre: eliminar() ***Tipo de retorno***:void	***Descripción*****:** Esta funcionalidad reenvía para el escenario Eliminar del Caso de Uso, previamente le pasa los parámetros necesarios para esta acción.
Nombre: cancelar() ***Tipo de retorno***:void	***Descripción*****:** Esta funcionalidad regresa a la página que invocó al escenario del Caso de Uso.

Nombre de la clase: EliminarInformeIdentificacionDeIndividuoManejado

Descripcion: Esta clase es el respaldo de la página encargada de presentar una interfaz al usuario que permita eliminar un informe de identificación de individuo.	
Capa de la Arquitectura: Presentación	
Ubicación: vnz.cicpc.investigacioncriminalistica.experticias.web.bean	
Atributo:	***Tipo***:
informe	InformeIdentificacionDeIndividuo
justificacion	String
criminalisticaExperticiasFacade	InvestigacionCriminalisticaExperticiasFacade
Responsabilidades:	
Nombre: eliminar() ***Tipo de retorno***:void	***Descripción*****:** Esta funcionalidad ejecuta el método que elimina los datos de la entidad.
Nombre: cancelar() ***Tipo de retorno***:void	***Descripción*****:** Esta funcionalidad regresa a la página que invocó al escenario del Caso de Uso.

Nombre de la clase: InformeIdentificacionDeIndividuoServiceImpl

Capa de la Arquitectura: Lógica de Negocio.	
Descripción: Esta clase se encarga de implementar la interfaz que presenta métodos de igual firma y tiene el mismo nombre sin el sufijo *Impl*, engloba la lógica de interacción entre los objetos del domino.	
Ubicación: vnz.cicpc.investigacioncriminalistica.experticias.service.impl	
Atributo:	***Tipo***:
analisisInformacionFacade	AnalisisInformacionFacade
seguridadFacade	SeguridadFacade
investigacionPenalFacade	InvestigacionPenalFacade
informeIdentificacionDeIndividuoDao	InformeIdentificacionDeIndividuoDao
nomencladorComunFacade	NomencladorComunFacade
analisisInformacionFacade	AnalisisInformacionFacade
seguridadFacade	SeguridadFacade
Responsabilidades:	
Nombre: actualizar(**InformeIdentificacionDeIndividuo** informe) ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad prepara el informe obtenido por parámetro invocando a la clase utilitaria *PobladorInforme,* posteriormente se encarga de acceder al método del DAO que encarga de actualizar el informe.
Nombre:enviar(**InformeIdentificacionDeIndividuo** informe) ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad prepara el informe obtenido por parámetro invocando a la clase utilitaria *PobladorInforme,* posteriormente se encarga de acceder al método del DAO que encarga de salvar el informe con estado *Emitido*.
Nombre:guardar(**InformeIdentificacionDeIndividuo** informe) ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad prepara el informe obtenido por parámetro invocando a la clase utilitaria *PobladorInforme,* posteriormente se encarga de acceder al método del DAO que encarga de salvar el informe con estado *En Curso*.
Nombre: obtenerPorID(**int** id) ***Tipo de retorno***: InformeIdentificacionDeIndividuo	***Descripción*:** Esta funcionalidad obtiene el informe identificado por la propiedad id de la base de datos y lo devuelve con los datos cargados, o sea la propiedad *lazy=false* de hibernate.
Nombre:eliminar(**InformeIdentificacionDeIndividuo** informe, **String** justificacion) ***Tipo de retorno***: void	***Descripción*:** Esta funcionalidad se encarga de almacenar la justificación de eliminación y de realizar una falsa eliminación de la entidad, debido a que es política del cliente no borrar ningún dato, lo que se hace es desactivar su recuperación.

Nombre de la clase: InformeIdentificacionDeIndividuoDaoImpl

Capa de la Arquitectura: Acceso a Datos.	
Descripción: Esta clase se encarga de implementar la interfaz que presenta métodos de igual firma y tiene el mismo nombre sin el sufijo *Impl*, y engloba las utilidades de interacción con la base de datos; las funcionalidades que hereda de la clase *DaoGenerico*, permiten realizar las acciones básicas.	
Ubicación: vnz.cicpc.investigacioncriminalistica.experticias.service.impl	
Atributo:	***Tipo***:
Esta clase no presenta atributos	
Responsabilidades:	
Nombre:obtenerPorID(**int** id) **Tipo de retorno:** void	***Descripción*:** Esta funcionalidad se encarga de recuperar un objeto con el id obtenido por parámetro de la base de datos, invocando métodos que brinda la implementación del framework Hibernate.

Nombre de la clase: InformeIdentificacionDeIndividuo

Capa de la Arquitectura: Lógica de Negocio.	
Descripción: Esta clase es persistente, hereda de la clase informe, y se encarga de modelar los datos del informe de identificación de individuo.	
Ubicación: vnz.cicpc.investigacioncriminalistica.experticias.domain	
Atributo:	***Tipo***:
observacion	*String*
Conclusiones	*String*
fechaIdentificacion	*Date*
correspondenciaHuellas	*boolean*
individuoIdentificado	*boolean*
primeroNombre	*String*
segundoNombre	*String*
primerApellido	*String*
segundoApellido	*String*
letraCedula	*Character*
numeroCedula	*String*
Responsabilidades:	
Nombre:getDatos() **Tipo de retorno:** String	***Descripción*:** Esta funcionalidad se encarga de Formatear los datos de la entidad para devolverlos en una sola cadena de caracteres.
Nombre: getFuncionarioIdentificador() **Tipo de retorno:** Funcionario	***Descripción*:** Esta funcionalidad busca en un atributo de la clase ancestro Diligencia, que representa un conjunto de funcionarios relacionados al informe, al funcionario que identificó a la persona citada en el informe.
Nombre:getObtenerNumeroCedulaPersonaIdentificada() **Tipo de retorno:** String	***Descripción*:** Esta funcionalidad concatena las partes de la cédula para devolverla.
Nombre: geNombreCompletoPersonaIdentificada() **Tipo de retorno:** String	***Descripción*:** Esta funcionalidad concatena las partes del nombre completo para devolverlo.

Nombre de la clase: PobladorInforme

Capa de la Arquitectura: Lógica de Negocio.	
Descripción: Esta clase es utilitaria, tiene múltiples métodos para cambiar valores de atributos en las entidades de tipo Informe pasado por parámetro.	
Ubicación: vnz.cicpc.investigacioncriminalistica.experticias.service.impl.util	
Atributo:	***Tipo***:
Esta clase no presenta atributos	
Responsabilidades:	
Nombre:asociarPersonaACaso(**ActaProcesal** actaProcesal**,Persona** persona) **Tipo de retorno:** void	***Descripción*:** Esta funcionalidad se encarga de asociar la entidad *Persona* a una *ActaProcesal*
Nombre:poblarInforme(Informe informe,**InvestigacionPenalFacade** investigacionPenalFacade, **NomencladorComunFacade** nomencladorFacade, **boolean** emitido) **Tipo de retorno:** void	***Descripción*:** Esta funcionalidad se encarga de asociar a la entidad *Informe* los datos que se necesitan a la hora de insertar/actualizar dicha entidad en la base de datos, recibe los datos de las fachadas de otros subsistemas obtenidas por parámetros.

2.4 Caso de Uso de Muestra: Gestionar Experticia de Microscopía Electrónica.

En esta sección se muestran los principales resultados del análisis y diseño del Caso de Uso Gestionar Experticia de Microscopía Electrónica; se utilizan diagramas del modelo de análisis y del modelo de diseño, generados por la metodología RUP, descripciones e imágenes.

2.4.1 Análisis de Caso de Uso: Gestionar Experticia de Microscopía Electrónica

Descripción resumida de los eventos.

El Caso de Uso se inicia cuando el Experto Técnico accede a realizar una acción sobre un informe de respuesta a una Solicitud de Experticia Criminalística. En caso de que el actor acceda a la sección donde puede introducir los resultados de la experticia, el sistema muestra de forma predeterminada algunos datos de la Solicitud de Experticia Criminalística y permite entrar otros datos que formarán parte de la experticia. El sistema permite seleccionar la opción de tener una vista previa del informe confeccionado, modificar sus datos en caso requerido si el actor es el responsable del informe, imprimir o exportar a formato PDF, enviar el informe para que sea revisado o guardarlo temporalmente para en un futuro acceder a los datos de este en su bandeja de borradores. En caso de que el actor acceda a la opción de ver los detalles del Informe de Microscopía Electrónica, el sistema muestra el contenido del mismo y brinda además la posibilidad de eliminarlo.

Entidades:

- Funcionario
- Evidencia
- Sitio de Suceso
- Informe de Microscopía Electrónica
- Acta Procesal

Requisitos Funcionales:

- Mostrar datos de la Solicitud de Experticia Criminalística
- Incluir datos asociados al resultado de experticia
- Mostrar vista previa del Informe de Microscopía Electrónica
- Modificar datos del Informe de Microscopía Electrónica
- Guardar el Informe de Microscopía Electrónica
- Exportar a PDF.

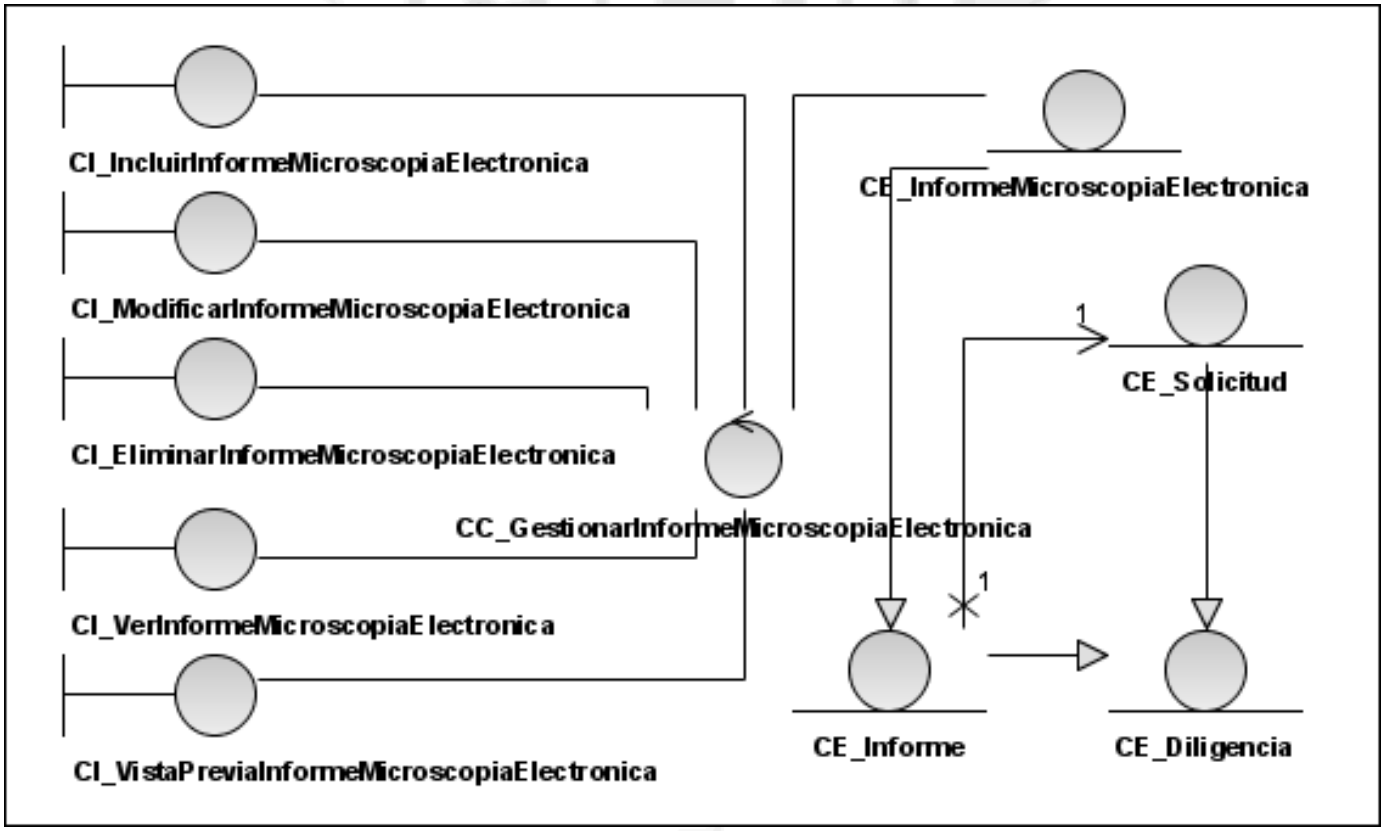

Ilustración 23: Diagrama de clases del análisis del Caso de Uso Gestionar Experticia de Microscopía Electrónica.

2.4.2 Diseño de Caso de Uso de Muestra: Gestionar Experticia de Microscopía Electrónica.

Esta sección muestra una vista del diagrama de clases del módulo que componen la capa de Lógica de Negocio y la capa de Acceso a Datos para el Caso de Uso y seguidamente se divide la capa de presentación relativa al Caso de Uso en escenarios para su mejor organización y comprensión. Por último, se tiene una muestra del diagrama de secuencia del contrato entre paquetes que representa la colaboración dentro del Caso de Uso en cuanto a mensajes.

La Ilustración 24 muestra el diagrama de clases de Lógica de Negocio y Acceso a Datos. La interfaz *InvestigacionCriminalisticaExperticiasfacade* repite los métodos de clases de servicio de las cuales es fachada, esta es una interfaz de Java y es implementada por una clase de igual nombre de Impl.
La interfaz de servicio *InformeMicElectronicaService* tiene su respectiva implementación, y provee las funcionalidades que son invocadas por la fachada, englobando la lógica correspondiente sobre las clases persistentes *InformeMicElectronica*, *KitDeMuestra* y *MarcaImagen,* para lo cual se auxilia, en caso de acceder a la base de datos, de la interfaz de acceso a datos.
Más adelante, de la Ilustración 25 a la 29, se muestra la capa de presentación dividida en escenarios, para hacer más comprensible el Caso de Uso. No se grafican por razones de espacio todos los detalles, en el diagrama de clases íntegro que se encuentra en los anexos sí aparecen representadas todas las relaciones entre las clases.
En la capa de Presentación los escenarios Modificar Informe e Incluir Informe tienen diseños muy similares. En el diagrama de clases se muestra la página servidora **.jsp** que será la encargada de generar el código HTML, que importa el fichero *java script* utilizado para las validaciones del lado del cliente. Las peticiones son controladas y distribuidas por el *Faces Servlet*, que reenvía hacia la página responsable de compilarse para dar respuesta a la petición, se ejecuta el código asociado a la llamada y se reflejan los cambios en la vista.
La mayor diferencia que existe entre el escenario Modificar e Incluir es que lógicamente debe cargarse el informe que se debe modificar, además de cumplir ciertas restricciones de campos no modificables. Para esto se agrega un nuevo método al *bean* manejado que carga la entidad especificada y distribuye los valores por los controles visuales que sean necesarios.
Los escenarios Ver Informe y Vista Previa Informe son notablemente parecidos también; la idea de estos es la de mostrar los datos asociados al informe; la diferencia es que no muestran los mismos datos de dicha diligencia, ni son presentados de la misma manera, visualmente hablando.

El escenario Eliminar Informe está compuesto a grandes rasgos por clases similares, con mecanismos parecidos; la función principal es mostrar los datos del informe a eliminar, permitir entrar una justificación y eliminar el informe.

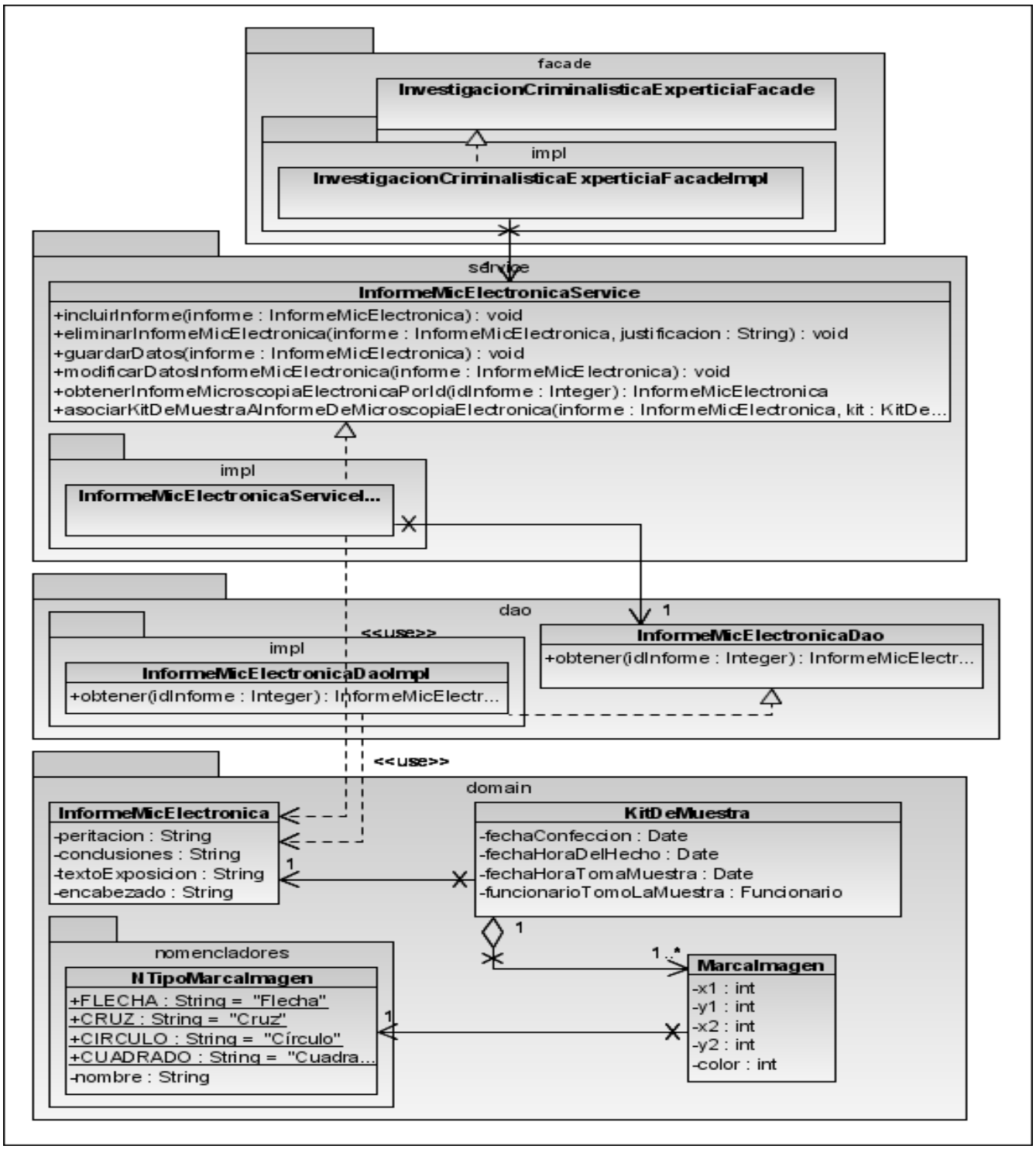

Ilustración 24: Diagrama de clases del diseño de Lógica de Negocio, Caso de Uso Gestionar Experticia de Microscopía Electrónica.

.

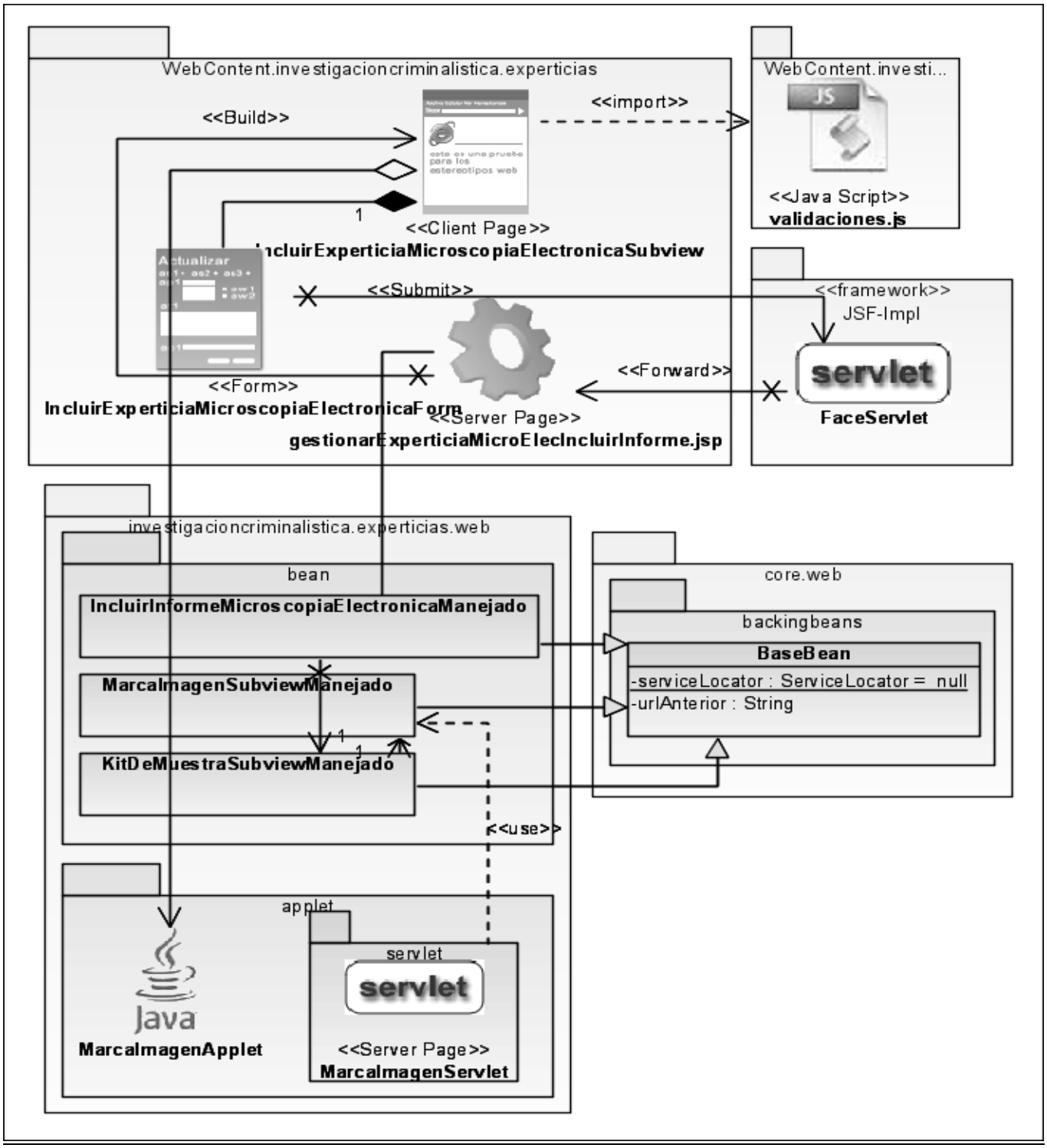

Ilustración 25: Diagrama de clases del diseño de Presentación, escenario Incluir Experticia de Microscopía Electrónica.

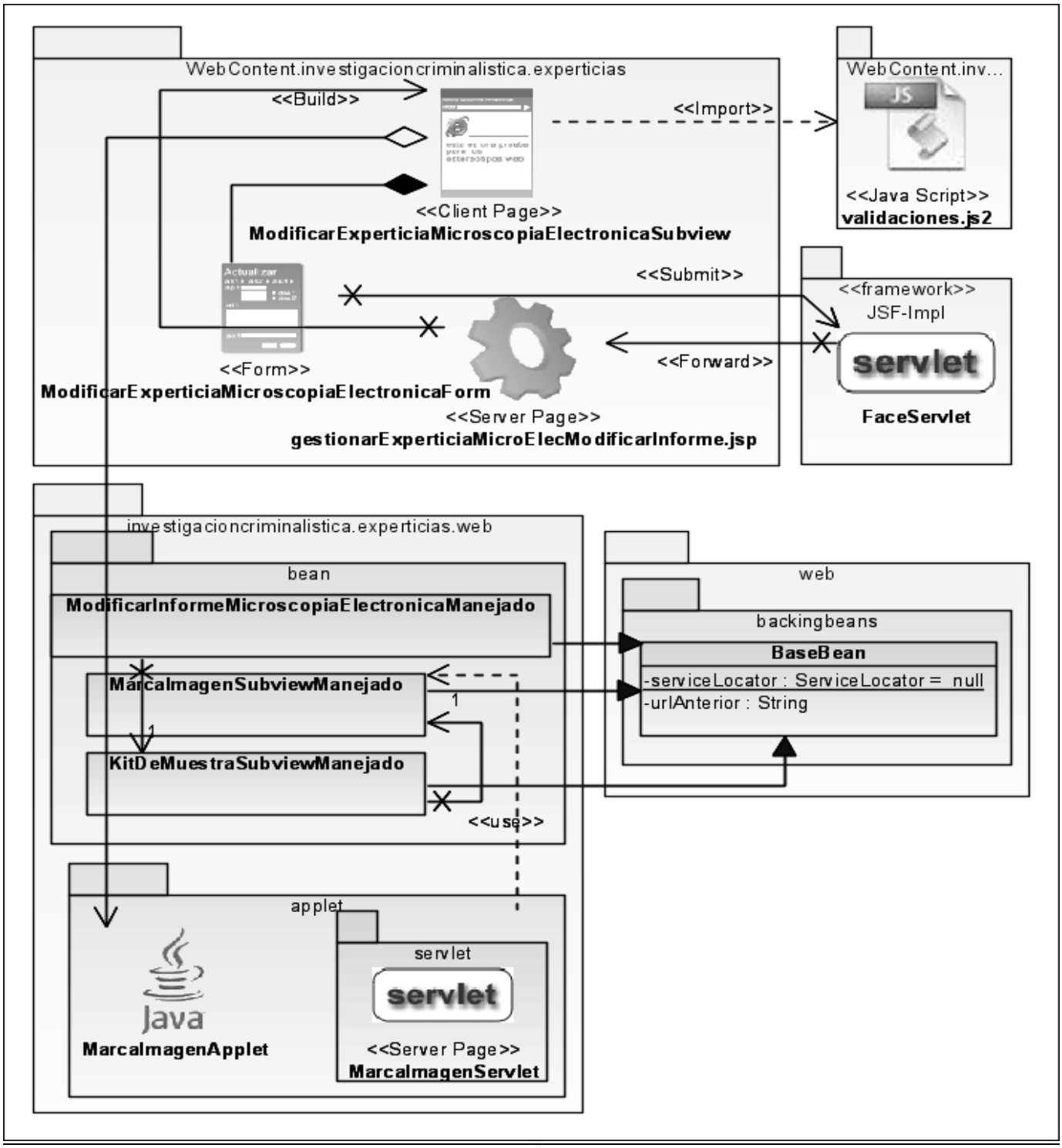

Ilustración 26: Diagrama de clases del diseño de Presentación, escenario Modificar Experticia de Microscopía Electrónica.

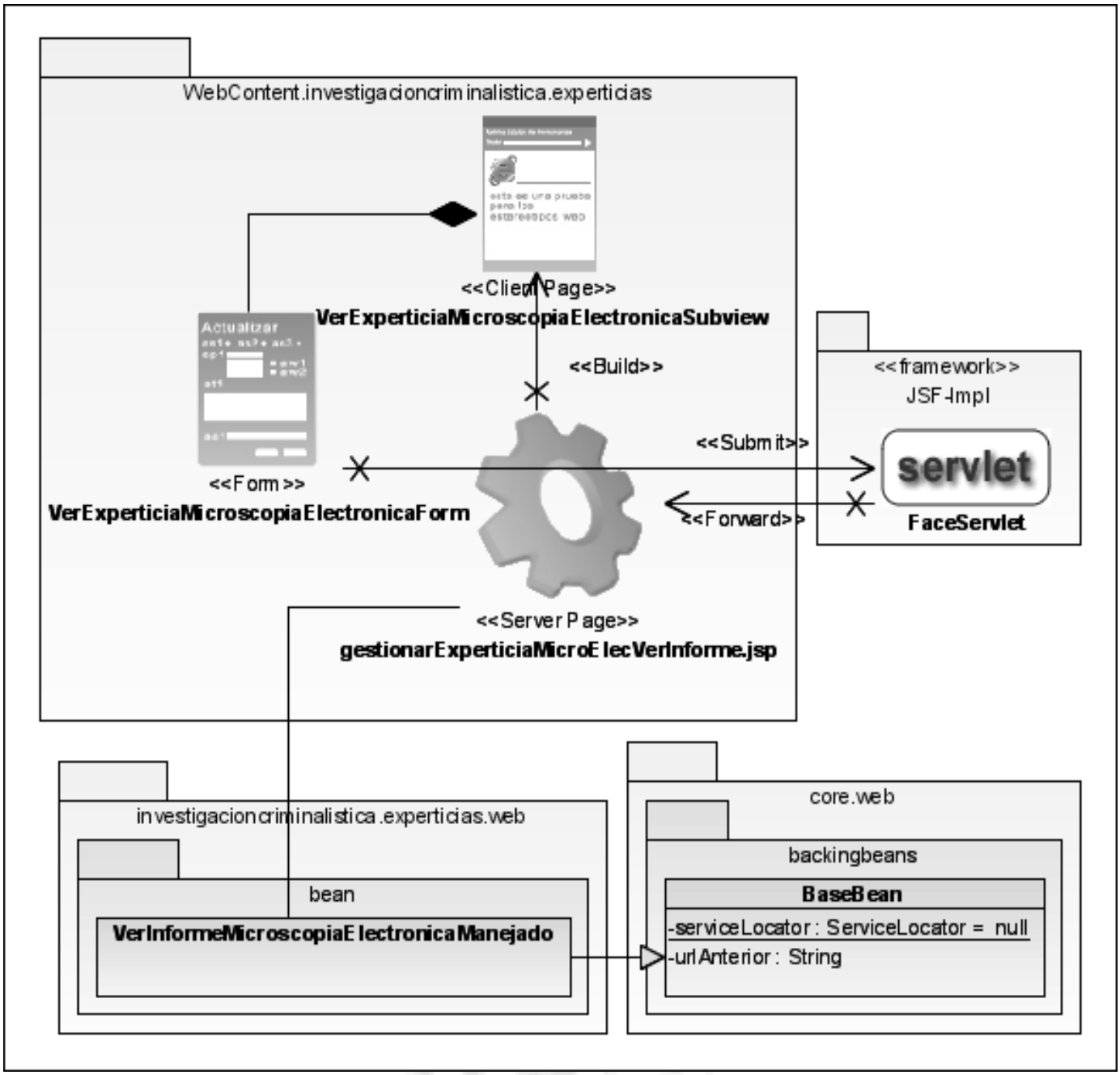

Ilustración 27: Diagrama de clases del diseño de Presentación, escenario Ver Experticia de Microscopía Electrónica.

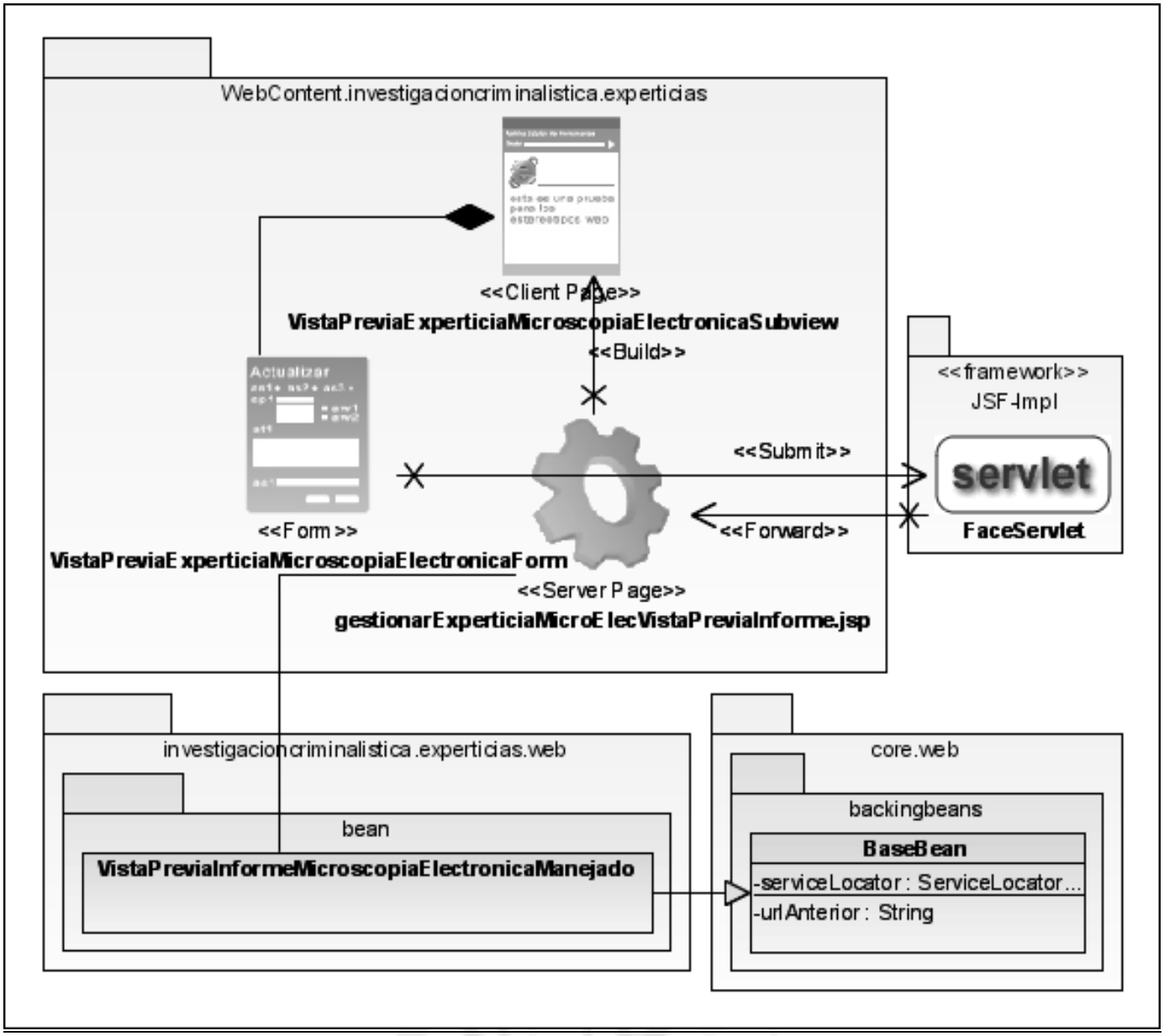

Ilustración 28: Diagrama de clases del diseño de Presentación, escenario Vista Previa Experticia de Microscopía Electrónica.

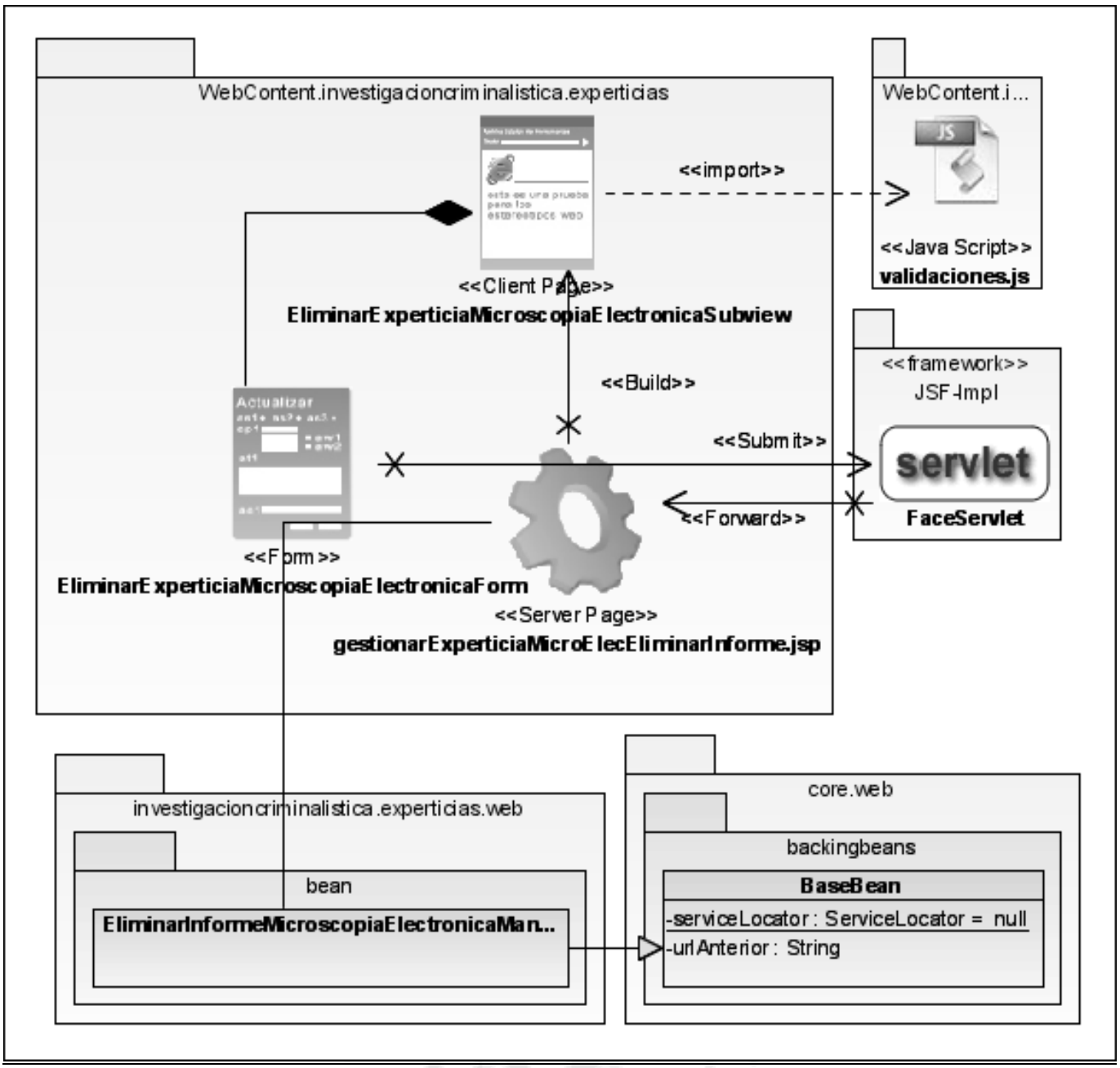

Ilustración 29: Diagrama de clases del diseño de Presentación, escenario Eliminar Experticia de Microscopía Electrónica.

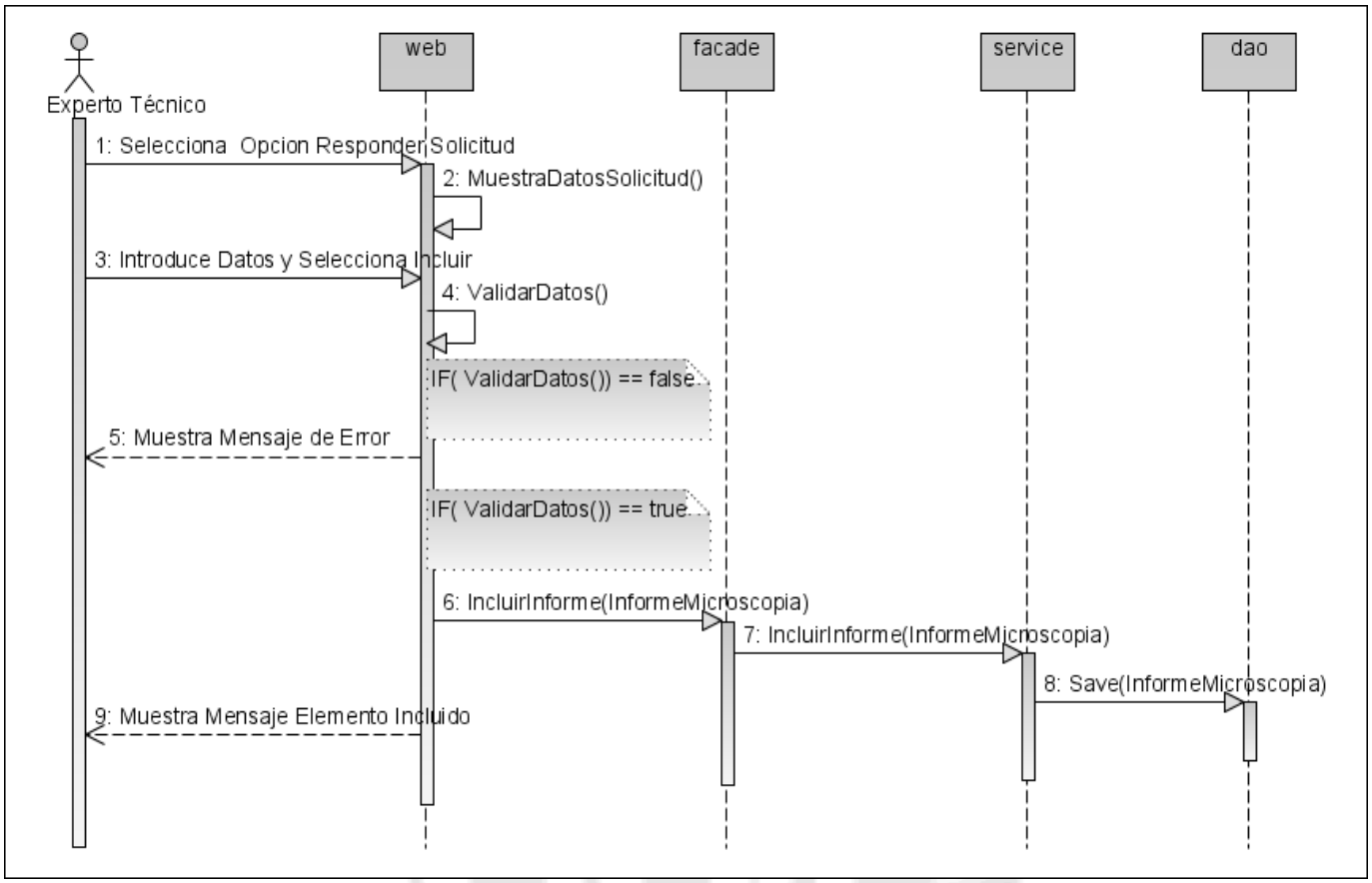

Ilustración 30: Diagrama de secuencia de contrato entre paquetes, escenario Incluir Experticia de Microscopía Electrónica.

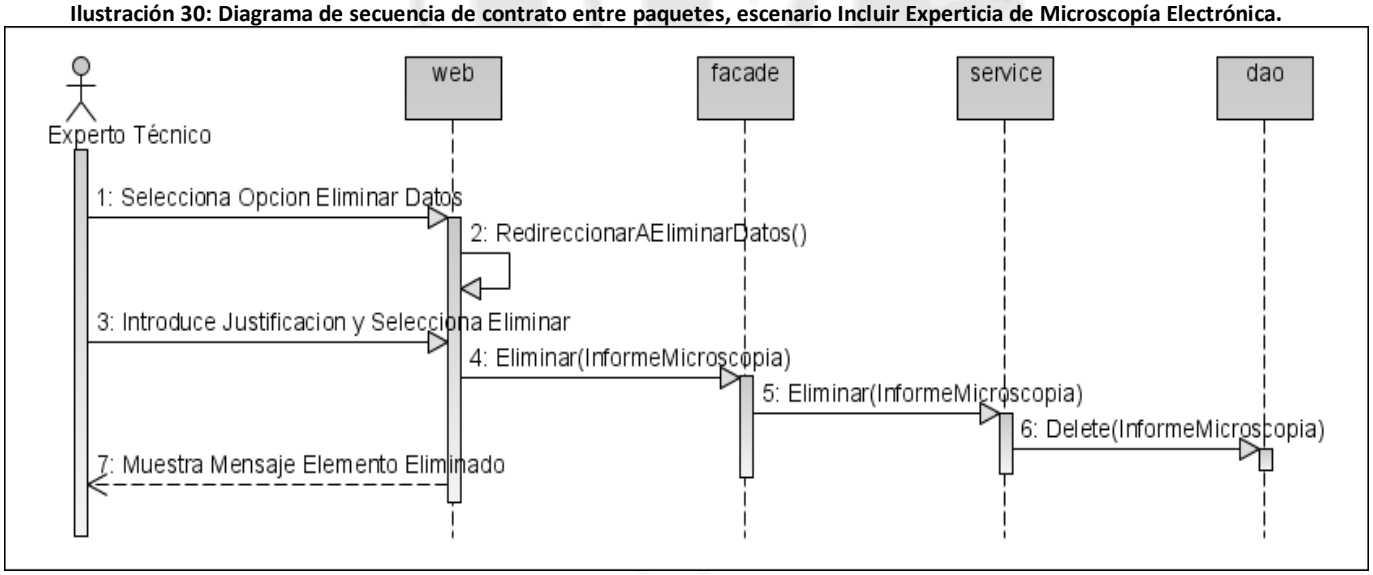

Ilustración 31: Diagrama de secuencia de contrato entre paquetes, escenario Eliminar Experticia de Microscopía Electrónica.

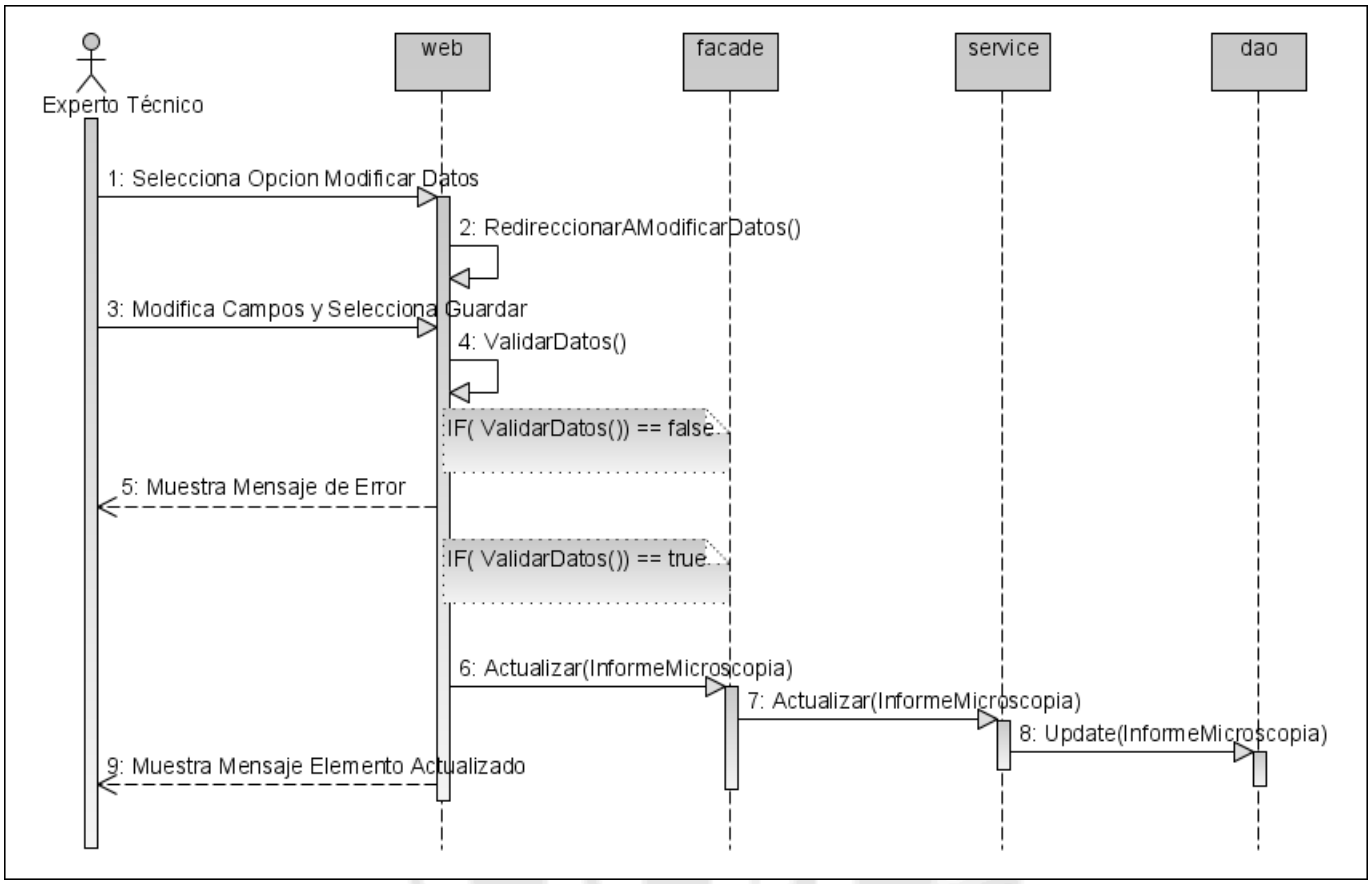

Ilustración 32: Diagrama de secuencia de contrato entre paquetes, escenario Modificar Experticia de Microscopía Electrónica.

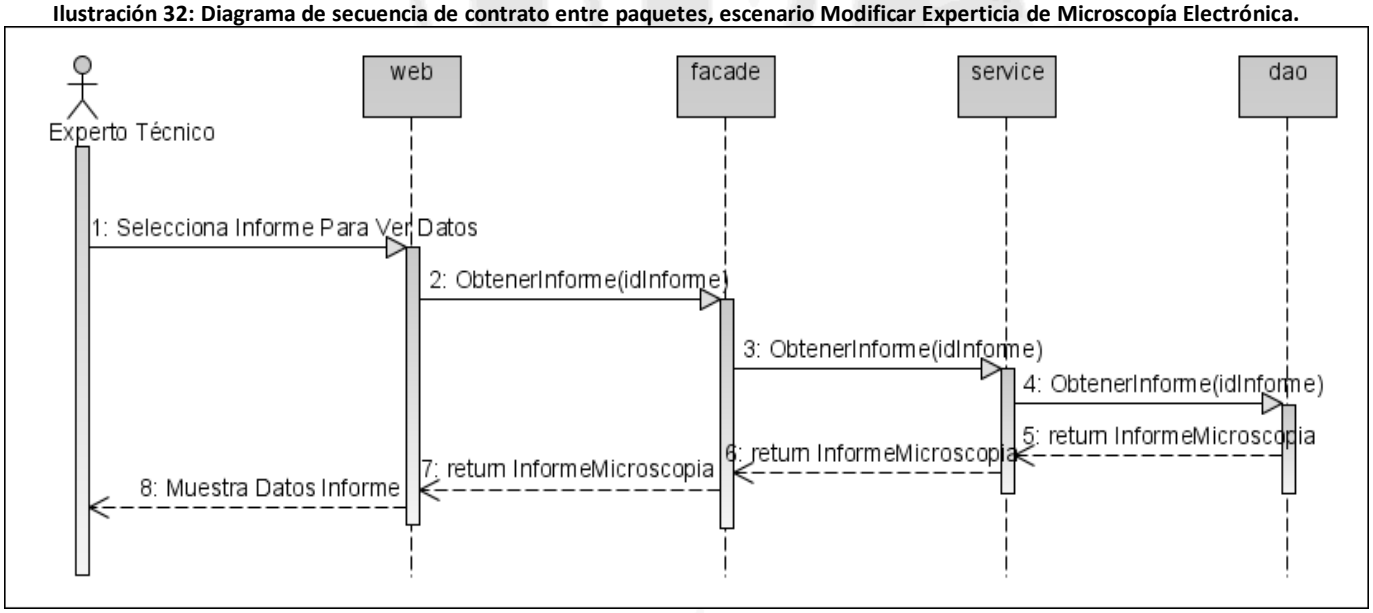

Ilustración 33: Diagrama de secuencia de contrato entre paquetes, escenario Ver Experticia de Microscopía Electrónica.

2.4.3 Descripción de las clases involucradas en el Caso de Uso.

En esta sección se describen los elementos fundamentales de cada clase que participa en la realización en términos de diseño del Caso de Uso.

Nombre de la clase: IncluirInformeMicroscopiaElectronicaManejado

Descripción: Esta clase es el respaldo de la página encargada de presentar una interfaz al usuario que permita incluir un nuevo informe de microscopía electrónica.	
Capa de la Arquitectura: Presentación	
Ubicación: vnz.cicpc.investigacioncriminalistica.experticias.web.bean	
Atributo:	***Tipo***:
solicitudAsociada	Solicitud
informe	InformeMicElectronica
listaArticulos	List<Articulo>
criminalisticaExperticiasFacade	InvestigacionCriminalisticaExperticiasFacade
kitDeMuestraSubviewManejado	KitDeMuestraSubviewManejado
consultarArticuloManejado	ConsultarArticuloManejado
vistaPreviaInformeMicroscopiaElectronicaManejado	VistaPreviaInformeMicroscopiaElectronicaManejado
validarActaProcesalSubviewManejado	ValidarActaProcesalSubviewManejado
personaEstudianda	Persona
Responsabilidades:	
Nombre: guardarEnviar () ***Tipo de retorno***:String	***Descripción*:** Esta funcionalidad se encarga de guardar en la base de datos el informe introducido, con el estado *En Curso o Emitido,* según el botón seleccionado. La cadena obtenida en el retorno del método es utilizada para la navegación.
Nombre: limpiar() ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad se encarga de borrar el informe completamente y crear uno nuevo en el acto, en términos programáticos le reserva memoria a un nuevo Informe, perdiendo la referencia al primero.
Nombre:configurarEncabezado(**ActionEvent** e) ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad se encarga de navegar al Caso de Uso extendido *Configurar Encabezado*.
Nombre: cancelar() ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad regresa a la página que invocó al escenario del Caso de Uso.
Nombre: vistaPrevia() ***Tipo de retorno***: String	***Descripción:*** Esta funcionalidad se encarga de ir a la vista previa del informe actual.

Nombre de la clase: VerInformeMicroscopiaElectronicaManejado

Descripción: Esta clase es el respaldo de la página encargada de presentar una interfaz al usuario que permita ver los datos de un informe de microscopía electrónica.	
Capa de la Arquitectura: Presentación	
Ubicación: vnz.cicpc.investigacioncriminalistica.experticias.web.bean	
Atributo:	***Tipo***:
informe	InformeMicElectronica
listaArticulos	List<Articulo>
criminalisticaExperticiasFacade	InvestigacionCriminalisticaExperticiasFacade

consultarArticuloManejado	ConsultarArticuloManejado
personaEstudianda	Persona
Responsabilidades:	
Nombre: generarReporte(ActionEvent e) ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad se encarga de crear una instancia de la clase utilitaria *GenerarReporte* y pasarle los datos del informe para generar un reporte en formato PDF.
Nombre: modificar() ***Tipo de retorno***:String	***Descripción*:** Esta funcionalidad reenvía para el escenario Modificar del Caso de Uso, previamente le pasa los parámetros necesarios para esta acción.
Nombre: eliminar() ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad reenvía para el escenario Eliminar del Caso de Uso, previamente le pasa los parámetros necesarios para esta acción.
Nombre: cancelar() ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad regresa a la página que invocó al escenario del Caso de Uso.

Nombre de la clase: EliminarInformeMicroscopiaElectronicaManejado

Descripción: Esta clase es el respaldo de la página encargada de presentar una interfaz al usuario que permita eliminar un informe de microscopía electrónica.	
Capa de la Arquitectura: Presentación	
Ubicación: vnz.cicpc.investigacioncriminalistica.experticias.web.bean	
Atributo:	***Tipo***:
informe	InformeMicElectronica
justificacion	String
criminalisticaExperticiasFacade	InvestigacionCriminalisticaExperticiasFacade
Responsabilidades:	
Nombre: eliminar() ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad ejecuta el método que elimina los datos de la entidad.
Nombre: cancelar() ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad regresa a la página que invocó al escenario del Caso de Uso.

Nombre de la clase: MarcaImagenServlet

Descripción: Esta clase extiende de *HttpServlet* y se encarga de mantener sincronizados los datos del Apple relativos a las marcas sobre la imagen y los datos del *bean* que respalda la página.	
Capa de la Arquitectura: Presentación	
Ubicación: vnz.cicpc.investigacioncriminalistica.experticias.web.bean	
Atributo:	***Tipo***:
Esta clase no tiene atributos.	
Responsabilidades:	
Nombre: doHandle() ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad busca el *bean* manejado en el contexto de JSF y obtiene los datos del pedido recibido desde el Apple, y se encarga de actualizar el *bean* manejado. Esta clase es llamada por el doGet() y el doPost();

Nombre de la clase: InformeMicElectronicaServiceImpl

Capa de la Arquitectura: Lógica de Negocio.	
Descripción: Esta clase se encarga de implementar la interfaz que presenta métodos de igual firma y tiene el mismo nombre sin el sufijo *Impl*, esta engloba la lógica de interacción entre los objetos del domino.	
Ubicación: vnz.cicpc.investigacioncriminalistica.experticias.service.impl	
Atributo:	***Tipo***:
analisisInformacionFacade	AnalisisInformacionFacade
seguridadFacade	SeguridadFacade
investigacionPenalFacade	InvestigacionPenalFacade
informeMicElectronicaDao	InformeMicElectronicaDao
nomencladorComunFacade	NomencladorComunFacade
analisisInformacionFacade	AnalisisInformacionFacade
seguridadFacade	SeguridadFacade
Responsabilidades:	
Nombre: incluirInforme(InformeIdentificacionDeIndividuo informe) ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad prepara el informe obtenido por parámetro invocando a la clase utilitaria *PobladorInforme,* posteriormente se encarga de acceder al método del DAO que encarga de salvar el informe con estado *Emitido*.
Nombre: guardarDatos(InformeMicElectronica informe) ***Tipo de retorno***:void	***Descripción*:** Esta funcionalidad prepara el informe obtenido por parámetro invocando a la clase utilitaria *PobladorInforme,* posteriormente se encarga de acceder al método del DAO que encarga de salvar el informe con estado *En Curso*.
Nombre: obtenerInformeMicroscopiaElectronicaPorId (Integer id) ***Tipo de retorno***: InformeMicElectronica	***Descripción*:** Esta funcionalidad obtiene el informe identificado por la propiedad id de la base de datos y lo devuelve con los datos cargados, o sea la propiedad *lazy=false* de hibernate.
Nombre: eliminarInformeMicElectronica (InformeIdentificacionDeIndividuo informe, String justificacion) ***Tipo de retorno***: void	***Descripción*:** Esta funcionalidad se encarga de almacenar la justificación de eliminación y de realizar una falsa eliminación de la entidad, debido a que es política del cliente no borrar ningún dato, lo que se hace es desactivar su recuperación.

Nombre de la clase: InformeMicElectronicaDaoImpl

Capa de la Arquitectura: Acceso a Datos.
Descripción: Esta clase se encarga de implementar la interfaz que presenta métodos de igual firma y tiene el mismo nombre sin el sufijo *Impl*, y engloba las utilidades de interacción con la base de datos;

las funcionalidades que hereda de la clase *DaoGenerico*, permiten realizar las acciones básicas.	
Ubicación: vnz.cicpc.investigacioncriminalistica.experticias.service.impl	
Atributo:	***Tipo***:
Esta clase no presenta atributos	
Responsabilidades:	
Nombre: obtener (**int** id) **Tipo de retorno:** void	***Descripción*:** Esta funcionalidad se encarga de recuperar un objeto con el id obtenido por parámetro de la base de datos, invocando métodos que brinda la implementación del *framework Hibernate*.

Nombre de la clase: InformeMicElectronica

Capa de la Arquitectura: Lógica de Negocio.	
Descripción: Esta clase es persistente, hereda de la clase *Informe*, y se encarga de modelar los datos del informe de identificación de individuo.	
Ubicación: vnz.cicpc.investigacioncriminalistica.experticias.domain	
Atributo:	***Tipo***:
peritacion	String
Conclusiones	String
textoExposicion	String
kitDeMuestra	KistDeMuestra
encabezado	String
Responsabilidades:	
Nombre:getDatos() **Tipo de retorno:** String	***Descripción*:** Esta funcionalidad se encarga de formatear los datos de la entidad para devolverlos en una sola cadena de caracteres.

Nombre de la clase: KitDeMuestra

Capa de la Arquitectura: Lógica de Negocio.	
Descripción: Esta clase es persistente, y se encarga de modelar los datos de las muestras analizadas para el informe de microscopía, almacenando las marcas sobre la imagen. Se encarga solo de representar datos, no tiene responsabilidades de peso.	
Ubicación: vnz.cicpc.investigacioncriminalistica.experticias.domain	
Atributo:	***Tipo***:
fechaConfeccion	Date
fechaHoraDelHecho	Date
fechaHoraTomaMuestra	Date
funcionarioTomoLaMuestra	Funcionario
marcasSobreImagen	Set<MarcaImagen>
informe	InformeMicElectronica

Nombre de la clase: MarcaImagen

Capa de la Arquitectura: Lógica de Negocio.	
Descripción: Esta clase es persistente, y representa las marcas sobre la imagen. Se encarga solo de representar datos, no tiene responsabilidades de peso.	
Ubicación: vnz.cicpc.investigacioncriminalistica.experticias.domain	
Atributo:	***Tipo***:
x1	int
y1	int
x2	int
y2	int
color	int
tipoMarcaImagen	NTipoMarcaImagen

2.5 Conclusiones

El análisis y diseño del módulo, del cual se tomaron para exponer en este documento solo dos casos de uso, es la pieza base para el trabajo posterior. Una vez realizados los mismos se tienen todos los elementos para implementar la propuesta. Aquí se ha reseñado solo una porción del análisis y diseño con los aspectos generales del flujo de trabajo de RUP, haciendo énfasis en ejemplificar los resultados más importantes. El trabajo de analizar y diseñar es eminentemente práctico, y se obtienen muchos diagramas a partir de él, la mayoría de ellos se pueden encontrar en los anexos. De manera general se considera haber realizado las tareas y haber generado los artefactos necesarios para pasar al próximo flujo de trabajo.

Capítulo 3. Implementación del módulo Experticias Criminalísticas

En este capítulo se describe el trabajo fundamental realizado en el flujo de trabajo de Implementación en el módulo Experticias Criminalísticas. Cubre fundamentalmente dos partes: la visualización de las principales estructuras utilizadas en la codificación y la presentación de los diagramas que RUP define para este flujo de trabajo.

3.1 Concepción de la implementación de Experticias Criminalísticas

El propósito de la sección es explicar, gráficamente, el trabajo de implementación en el módulo Experticias Criminalísticas; un diagrama de componentes muestra las organizaciones y dependencias lógicas entre los componentes del *software*, sean éstos componentes de código fuente, binarios o ejecutables. Desde el punto de vista de este diagrama se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del *software*, la reutilización, y las restricciones impuestas por los lenguajes de programación y por las herramientas utilizadas en el desarrollo. Los elementos de modelado dentro de un diagrama de componentes serán componentes y paquetes.

Un componente es una parte modular de un sistema, desplegable y reemplazable, que encapsula implementación y un conjunto de interfaces, y proporciona la realización de estas. Un componente típicamente contiene clases y puede ser implementado por uno o más artefactos (ficheros ejecutables, binarios, entre otros); mientras que un paquete en un diagrama de componentes representa una división física del sistema. Los paquetes se organizan en una jerarquía de capas, donde cada capa tiene una interfaz bien definida (23). A continuación se brinda una porción del diagrama de componentes del modulo Experticias Criminalísticas; está organizado de forma que cubre toda la estructura del módulo de implementación, pero no se visualizan los componentes que forman los paquetes en todos los casos, por razones de espacio. El segundo diagrama de implementación constituye otra abstracción de la implementación del módulo. Para culminar la sección se muestra la estructura de paquetes del módulo, con el objetivo de ilustrar la organización del código fuente; esta imagen no es un entregable de RUP, solamente tiene fines didácticos.

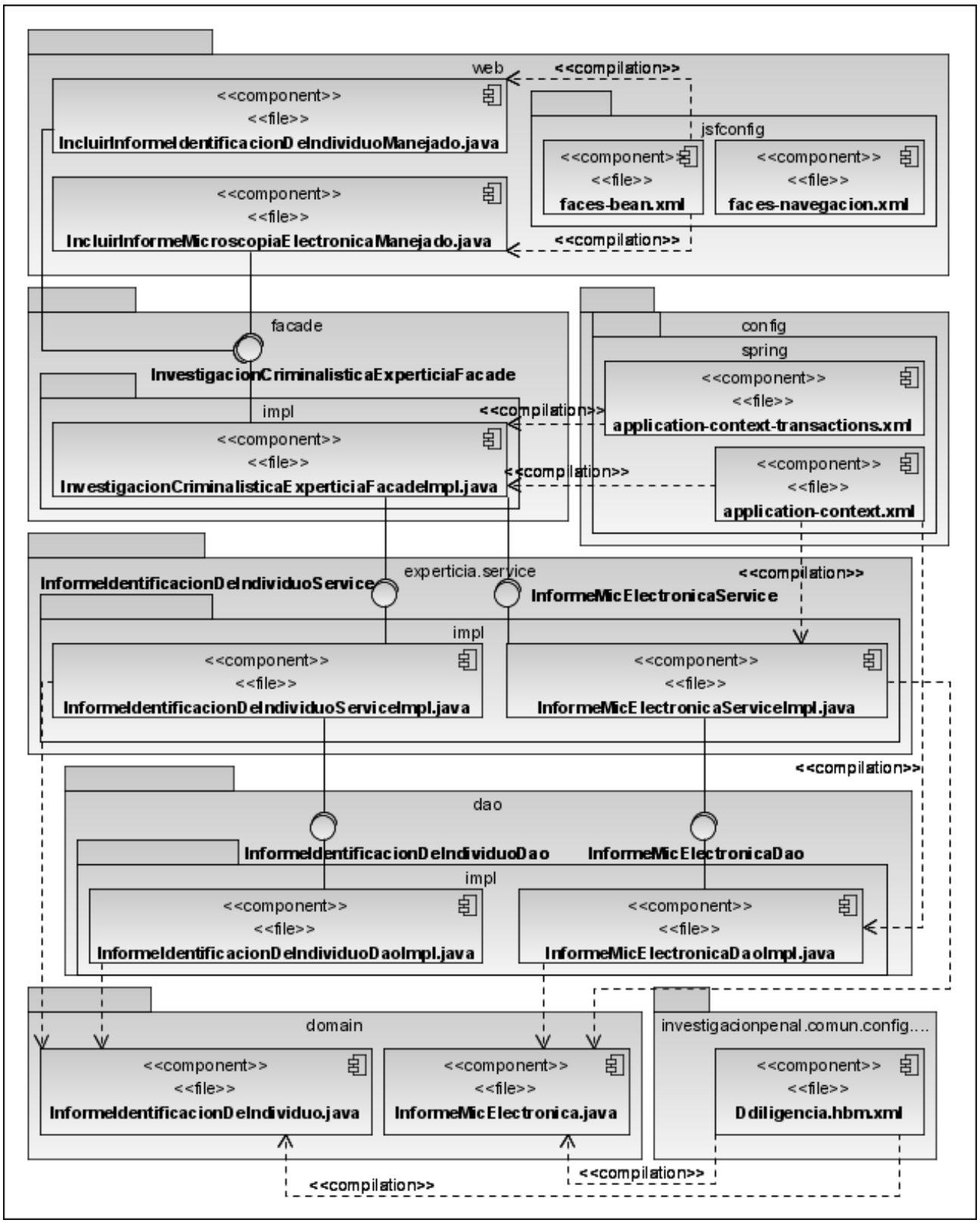

Ilustración 34: Diagrama de Componentes del módulo Experticias Criminalísticas

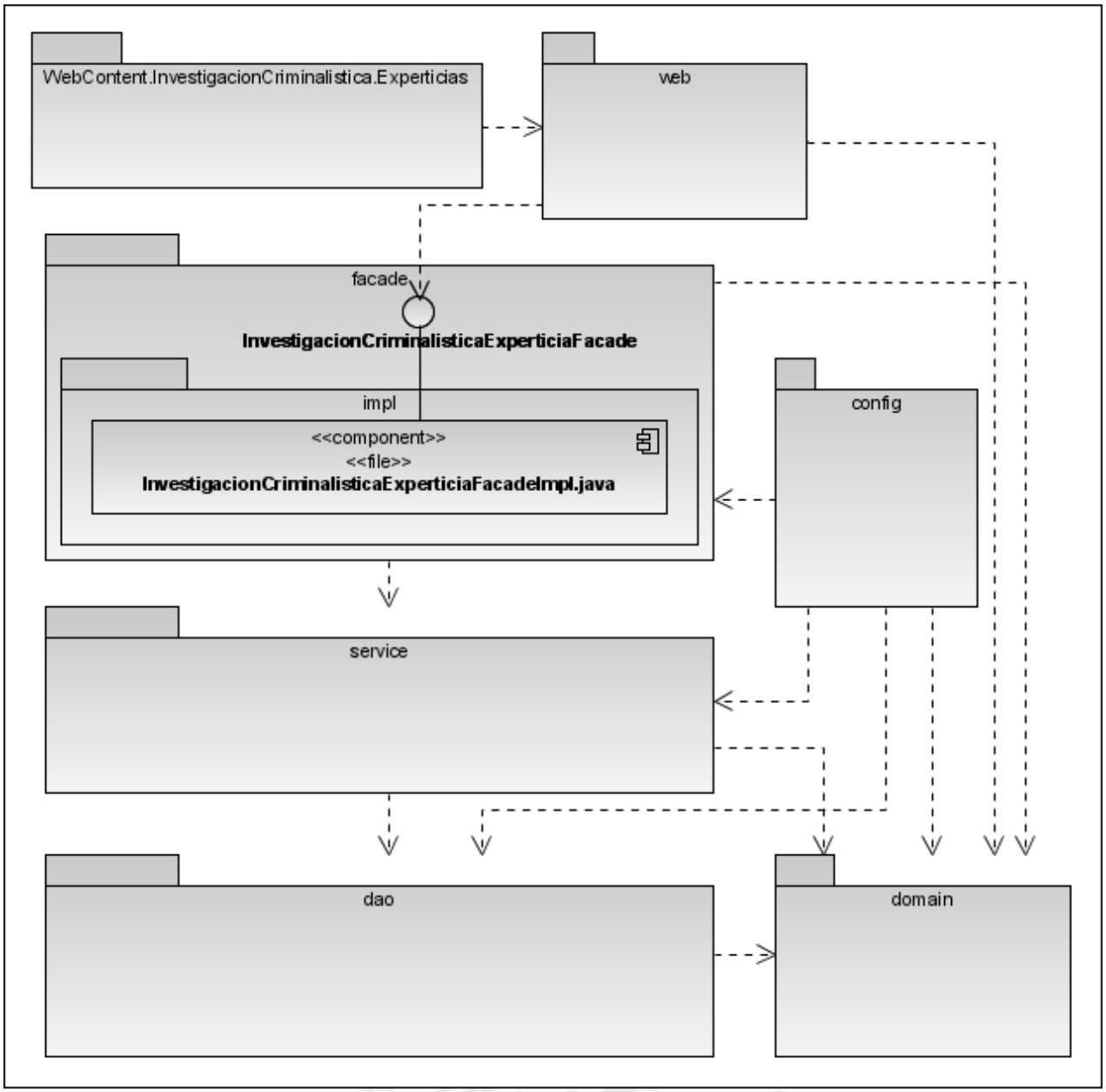

Ilustración 35: Vista General del Diagrama de Componentes del módulo de Experticias Criminalísticas

3.2 Codificación del módulo

Esta sección pretende reflejar el trabajo de los programadores de la aplicación. Está estructurada en dos partes fundamentales; a continuación se explica, por cada capa, la aplicación de las principales características de los *frameworks* a la implementación del módulo Experticias Criminalísticas, y después se da una muestra de los artefactos que se generaron utilizando la metodología RUP, para ilustrar el resultado práctico obtenido de la aplicación del diseño propuesto**.**

Componentes personalizados: Además de las páginas y los *bean* de respaldo, JSF provee una infraestructura para crear componentes personalizados; estos son desarrollados por el equipo de arquitectura y puestos en funcionamiento por los programadores del proyecto. Entre los componentes utilizados para los casos de uso del módulo están: el componente **cicpc:tiempo,** que se utiliza para cumplir la especificación de los casos de uso en cuanto a la manera que el usuario va a introducir una hora en el sistema; y el **cicpc:datascroller,** utilizado para el paginado de tablas con numerosos registros**.** La inclusión de estos componentes en las páginas se reduce a:

<cicpc:tiempo value="#{incluirInformeIdentificacionDeIndividuoManejado.horaIdentificacion}" /> y su utilización permite ahorrar tiempo y esfuerzo. A veces es imposible resolver un problema con los componentes provistos por el *framework* y su implementación y utilización se hace imprescindible.

Convertidores: El proceso de conversión de objetos a código HTML para presentarlos al cliente y el de transformar las entradas del usuario en los objetos con los que se mapean en el *bean* de respaldo, es desarrollado utilizando los convertidores de JSF; su uso implica implementar una interfaz suministrada por el *framework* llamada *Converter,* que tiene dos métodos: *getAsString()* para convertir el objeto en el texto que se mostrará en la página HTML y *getAsObject()* para obtener el objeto que está representado por la cadena de texto seleccionada o introducida, de manera que se desacopla totalmente este código del *bean* de respaldo. El *framework* además brinda algunos convertidores, sobre todo para tipos de datos básicos del lenguaje, que son utilizados muy comúnmente (12).

En el módulo se implementaron algunos, pues son de utilidad; por ejemplo, al mostrar colecciones de elementos en la interfaz de usuario y seleccionar alguno de ellos, o al convertir el valor obtenido en la entrada de una fecha o una hora hacia el sistema.

Validadores: Otra facilidad ofrecida por JSF son los Validadores; la implementación de la interfaz Validator provee maneras de capturar cualquier dato que venga en la petición del cliente, manipularlo para verificar su validez y tratar instantáneamente el error encontrado (12).

Los validadores y convertidores deben asociarse en las páginas con código JSF a los controles visuales que se desee procesar añadiendo su identificador en la propiedad del componente, o sea validator=”idValidador” o converter=”idConverter”.

Navegación: Virtualmente todas las aplicaciones *web* están hechas de un conjunto de páginas. Uno de los principales problemas de un desarrollador de aplicaciones w*eb* es manejar la navegación entre esas páginas (12). El modelo de navegación de *Java Server Faces* facilita la definición de la

navegación de páginas y el manejo de cualquier procesamiento adicional necesario para elegir la secuencia en que se cargan las páginas.

El módulo Experticias Criminalísticas utiliza dos métodos de navegación entre vistas; el primero permite reenviar a cualquier URL desde el código de las clases de respaldo; este método acopla los nombres de las páginas al código Java, pero en ocasiones resulta más efectivo, porque con la configuración adecuada se pueden obtener dinámicamente los valores de las direcciones hacia las que se desea navegar. El mecanismo que facilita desde cualquier *bean* manejado regresar a una URL es un ejemplo en el proyecto; la línea de código *retornarUrlAnterior()* invoca una funcionalidad que permite al programador redireccionar hacia la página anterior y *navegarAUrl(String url)* posibilita navegar a una dirección relativa al proyecto. El otro tipo de navegación que potencia JSF es la declarativa, y el módulo Experticias Criminalísticas se vale de este método en la mayoría de los casos; mediante configuraciones ajenas al código Java se puede especificar con qué valor de retorno de un método ejecutado por la página se llega desde una URL a otra.

Todas estas configuraciones: el registro de los *bean* manejados, los convertidores, los validadores y la navegación declarativa, se realizan sobre uno o varios archivos que el *framework* pone a disposición del programador; específicamente en el módulo son dos: uno para la navegación y el otro para el resto de las configuraciones, *jsf-config-beans.xml* y *jsf-config-navegation.xml,* respectivamente.

Las capas de Lógica de Negocio y de Acceso a Datos están muy ligadas al *framework* de capa media seleccionado para la construcción de SIIPOL. *Spring* soporta un ambiente de trabajo seguro y provee funcionalidades que abstraen al programador de muchas tareas.

Contenedor: *Spring* es un contenedor en el sentido que contiene y administra el ciclo de vida y la configuración de los objetos de la aplicación, o sea, las instancias de las clases registradas en el contexto; se puede configurar incluso si se va a manejar una sola instancia de determinada clase o si se va a producir una nueva instancia cada vez que se necesite; también se definen las asociaciones entre las clases contenidas; un *bean*, en el contexto de *Spring*, es un objeto que es creado y manejado por el contenedor *Spring*, o sea, el *bean* no es una clase que cumple una serie de normas o restricciones, sino que es un objeto (14).

Los paquetes *org.springframework.beans* y *org.springframework.context* proporcionan la base para el contenedor. En el primer paquete se encuentra la interfaz *BeanFactory*, que proporciona la capacidad de gestionar cualquier tipo de objeto. En el segundo paquete está la subinterfaz *ApplicationContext*, construida sobre la base del anterior, añade a *BeanFactory* una mejor integración con AOP, manejo de

internacionalización, propagación de eventos y manejo de contextos *web*, entre otras ventajas. La configuración del contexto se plasma en un archivo XML que contiene la definición de instancias, así como sus dependencias.

El proyecto utiliza una implementación de la interfaz *ApplicationContext* para gestionar el contexto de *Spring* y el módulo Experticias Criminalísticas separa su configuración en dos archivos diferentes, uno llamado *application-context-bean.xml* y otro *aplication-context-transacion.xml*.

Inversión de control: *Spring* promueve el bajo acoplamiento de las clases por medio de esta técnica, al aplicar la inversión de control, específicamente mediante la inyección de dependencia, a través de la cual las dependencias adquiridas pasivamente por los objetos, o sea, el programador no es el encargado de crear un *bean,* sino que se obtiene del contenedor (14).

Las dependencias se manejan declarando las referencias del tipo de la interfaz (no de la implementación) y registrándolas en el archivo de configuración de *Spring,* donde se especifica el nombre del atributo y la referencia que se le va a inyectar, que no es más que otro *bean* registrado en el archivo; de esta manera quedan desacopladas las clases y en ningún momento a una clase le "interesa" quién implementa las funcionalidades que ella utiliza de la referencia inyectada por *Spring*. El registro de un *bean* en el archivo de configuración se ve de esta manera:

```
<bean id="investigacionCriminalisticaExperticiasFacade"
class="facade.impl.InvestigacionCriminalisticaExperticiasFacadeImpl">
  <property name="informeMicElectronica" ref="informeMicrosElectService" />
  <property name="gestionSolicitudService" ref="gestionSolicitudService" />
</bean>
```

AOP: La programación orientada a aspectos promueve la separación de la lógica de ejecución de la aplicación de cuestiones externas, como pudieran ser auditorías y manejo de transacciones. Un aspecto es una funcionalidad transversal al método de clases que se ejecuta en un momento determinado, o sea, no interfiere en el código de lógica de negocio, sino que se ejecuta transparentemente. *Spring* utiliza AOP para ofrecer manejo de transacciones, que es declarativo (14); de esta forma, en el SIIPOL el control de transacciones se aplica a los métodos de las fachadas de los subsistemas y módulos, como es el caso de Experticias Criminalísticas, para convertirlo en un conjunto de operaciones que van a ser tratadas como una única unidad cumpliendo cuatro propiedades fundamentales comúnmente conocidas como ACID (atomicidad, coherencia, asilamiento y durabilidad).

Los métodos que serán transaccionales se declaran en el archivo *aplication-context-transacion.xml*.

Integración con Hibernate: *Spring* no fuerza a sus usuarios a utilizar su potente característica de abstracción JDBC. Se integra bien con marcos de trabajo de mapeo objeto-relacional, especialmente con *Hibernate;* ofrece un manejo seguro y eficiente de sesiones y maneja la configuración de la clase *SessionFactory* de *Hibernate* en el contexto de la aplicación, logrando que esta sea más fácil de testear (24).

La capa de acceso a datos en el proyecto CICPC se asienta sobre *Hibernate*. El modelo relacional trata con relaciones y conjuntos, es matemático por naturaleza; mientras que el paradigma orientado a objetos trata con objetos, sus atributos y asociaciones de unos con otros. Tan pronto como se quiera persistir los objetos utilizando una base de datos relacional, se observará que hay una desavenencia entre estos dos paradigmas, la también llamada diferencia objeto-relacional. Un *framework* de mapeo objeto-relacional, lo que es *Hibernate*, ayuda a contrarrestar esta diferencia.

El módulo Experticias Criminalísticas tiene como estándar, al igual que el resto del proyecto, utilizar una clase de acceso a datos por cada clase del dominio del problema, siempre que tenga lógica, o sea de utilidad la recuperación de la entidad en solitario (24).

El diseño actual de clases obliga a las implementaciones de los objetos de acceso a datos a heredar de una clase *DaoGenerico*, que utiliza los beneficios del API de *Hibernate* que brinda y realiza las acciones básicas sobre las entidades persistentes. El *DaoGenerico* brinda métodos que abstraen aún más al programador, o sea, le brindan acceso a las funcionalidades sin preocuparlo de abrir la sesión por ejemplo.

El API: Para insertar objetos en la Base de Datos a través del API se utiliza el método save(Object objeto) de la clase *Session*, para insertar o actualizar si ya existe saveOrUpdate(Object objeto), etc. Para hacer búsquedas en la base de datos es posible usar find(String consulta) que devuelve una lista con los resultados, o iterate(String consulta) que devuelve un iterador, o bien crear un objeto *Query* usando createQuery(String consulta) y llamar más tarde al método find() del *Query*. La consulta en todos los casos estará escrita en un lenguaje similar a SQL propio de *Hibernate* llamado HQL (*Hibernate Query Language*).

Una consulta aparece de esta manera, nótese que el cuerpo de la consulta trata con clases, no con relaciones de la base de datos:

```
lista = getCurrentSession().createQuery("from InformePericialTrayectoriaIntraorganica i where
i.actaProcesal = :acta").setParameter("acta", actaProcesal).list();
```

Archivos de mapeo: *Hibernate* obtiene los datos necesarios para conectarse al origen de datos de un archivo con formato XML hibernate.cfg.xml, donde se almacenan los parámetros de configuración. Además brinda la posibilidad de mapear las clases con las tablas de la base de datos en las que cada campo se asociará con una columna de la base de datos, y un archivo XML por cada una de estas clases NombreClase.hbm.xml indica el mapeo entre objetos y relaciones.

```
<hibernate-mapping package="investigacioncriminalistica.experticias.domain"
schema="CICPC">
 <class name="SitioSuceso" table="DSitioSuceso">
   <id name="id" type="integer" column="id"/>
   <property name="nombre" type="string" column="Nombre"/>
 </class>
</hibernate-mapping>
```

De esta manera está implementado el módulo Experticias Criminalísticas, acorde con la arquitectura definida y tratando de lograr un balance del cumplimiento de los requisitos funcionales entre la sencillez y la eficiencia. La transferencia tecnológica reviste gran importancia para los equipos de desarrollo, por lo que se realizan constantes refactorizaciones; esta es una práctica que no recoge RUP en sus guías; es más propio de una metodología ágil, pero por definición RUP no obliga, sino propone sus formas de trabajar. Para cumplir este objetivo también se realizan optimizaciones de código, otra de las tareas a cumplir cabalmente por los desarrolladores, pues un *software* de la calidad esperada debe funcionar lo más rápido posible y a la vez debe respetar las convenciones de código, que en Java son particularmente importantes.

3.3 Interfaces del módulo Experticias Criminalísticas.

A continuación se muestran algunas interfaces de usuario, para ilustrar en este documento la parte del sistema que interactúa con el cliente, estas fueron escogidas al azar, y corresponden al Caso de Uso Gestionar Experticia Balística y gestionar Experticia de Microscopía Electrónica.

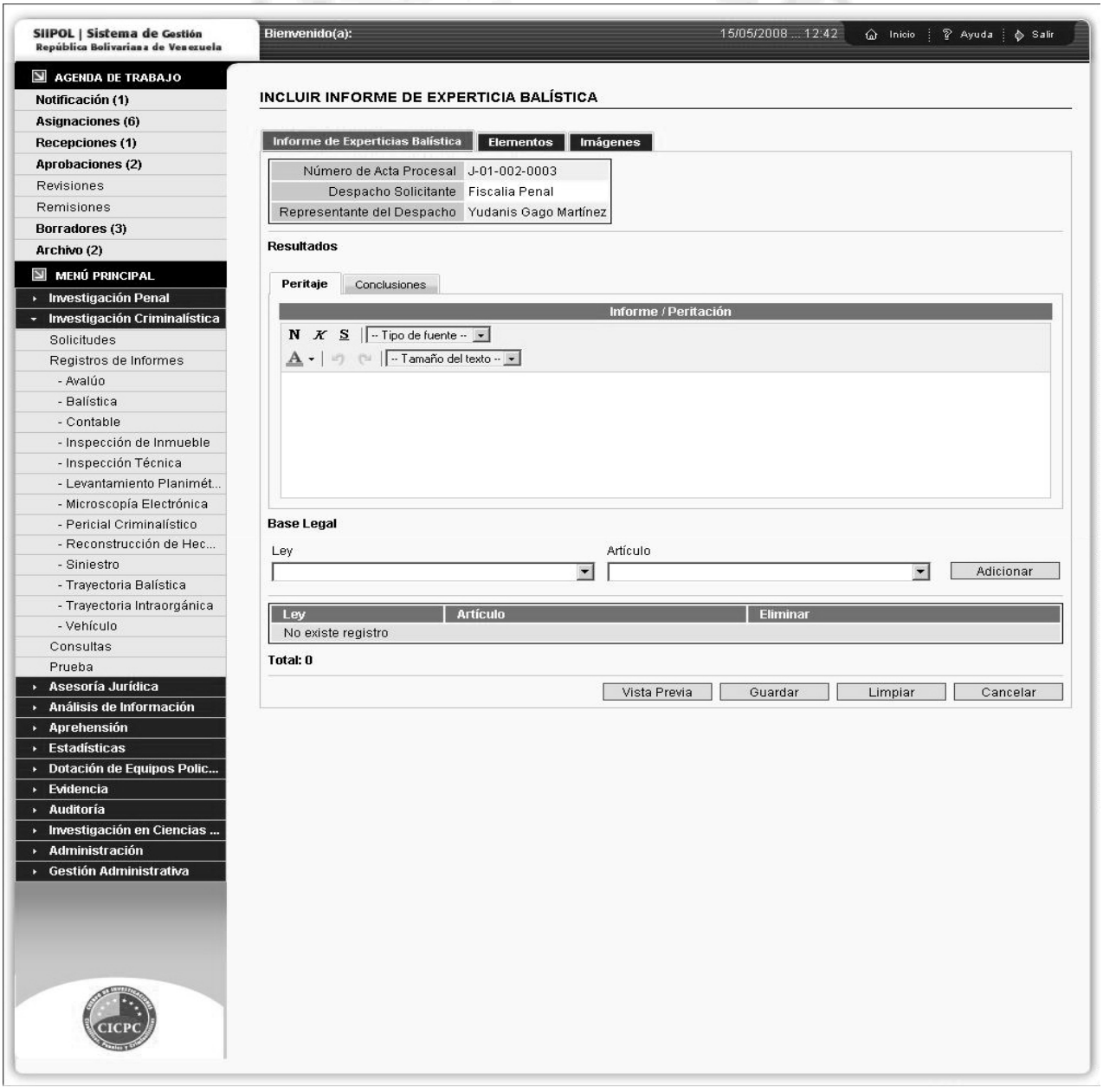

Ilustración 36 Interfaz de usuario Incluir Informe de Experticia Balística.

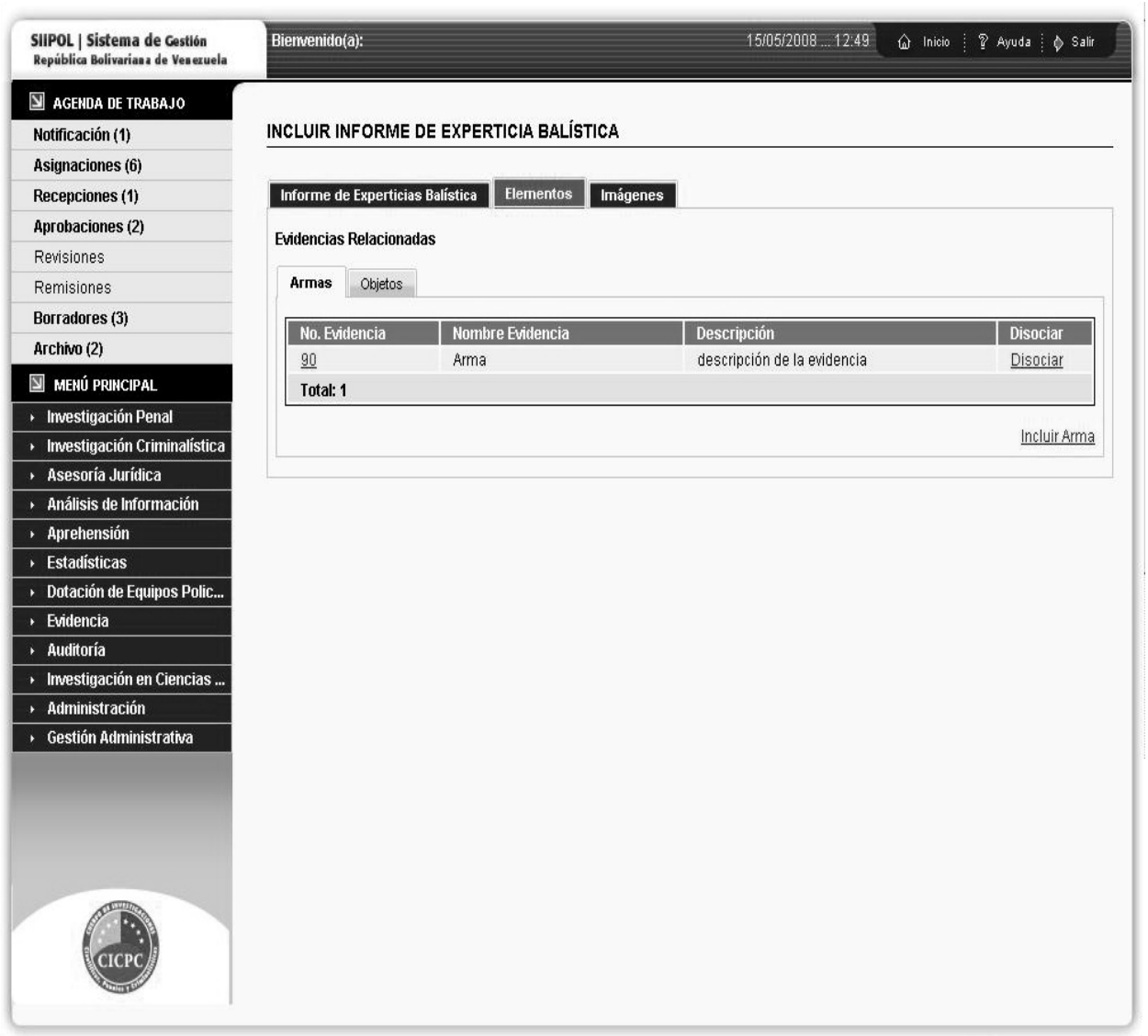

Ilustración 37: Interfaz de usuario Incluir Informe de Experticia Balística (2).

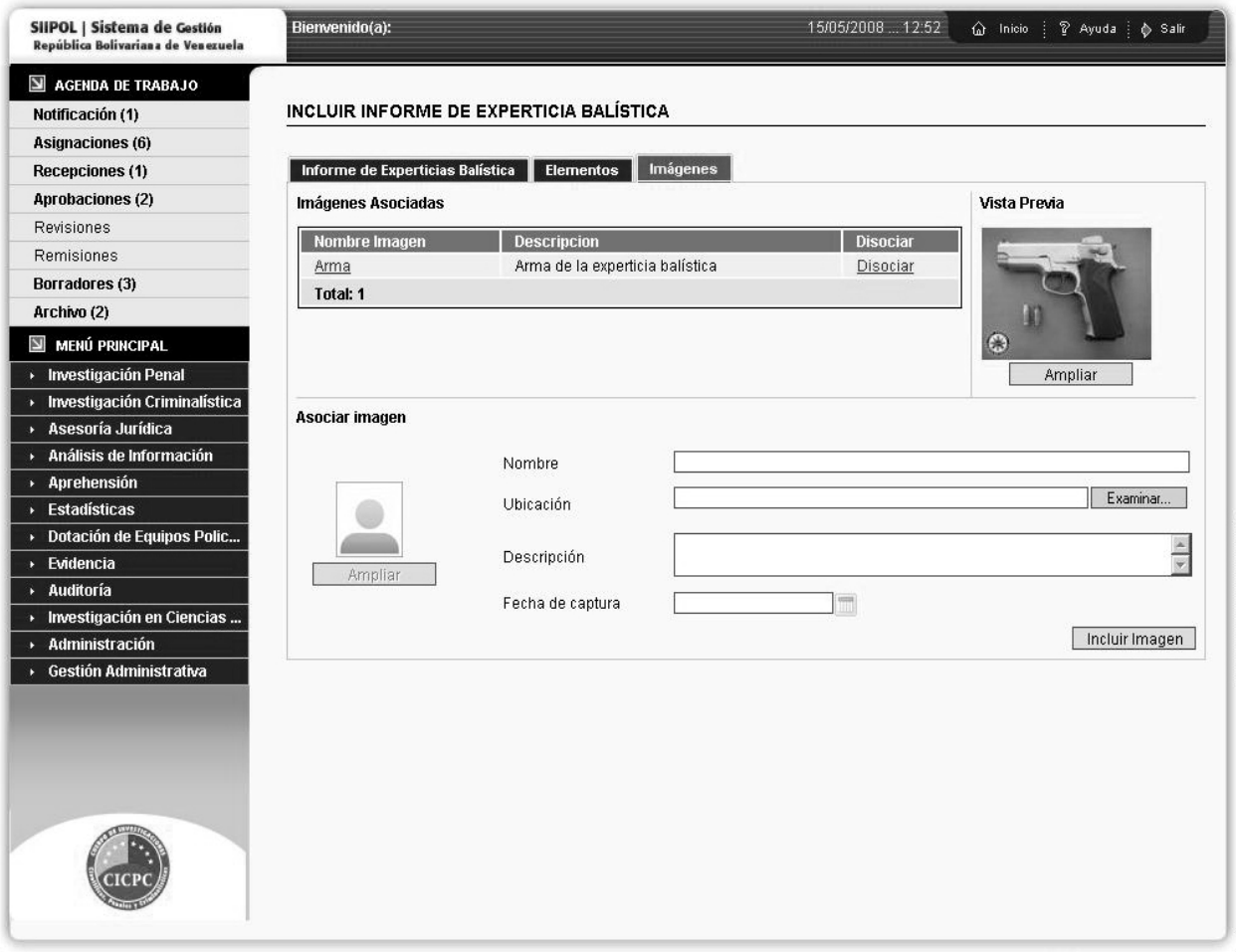

Ilustración 38: Interfaz de usuario Incluir Informe de Experticia Balística (3).

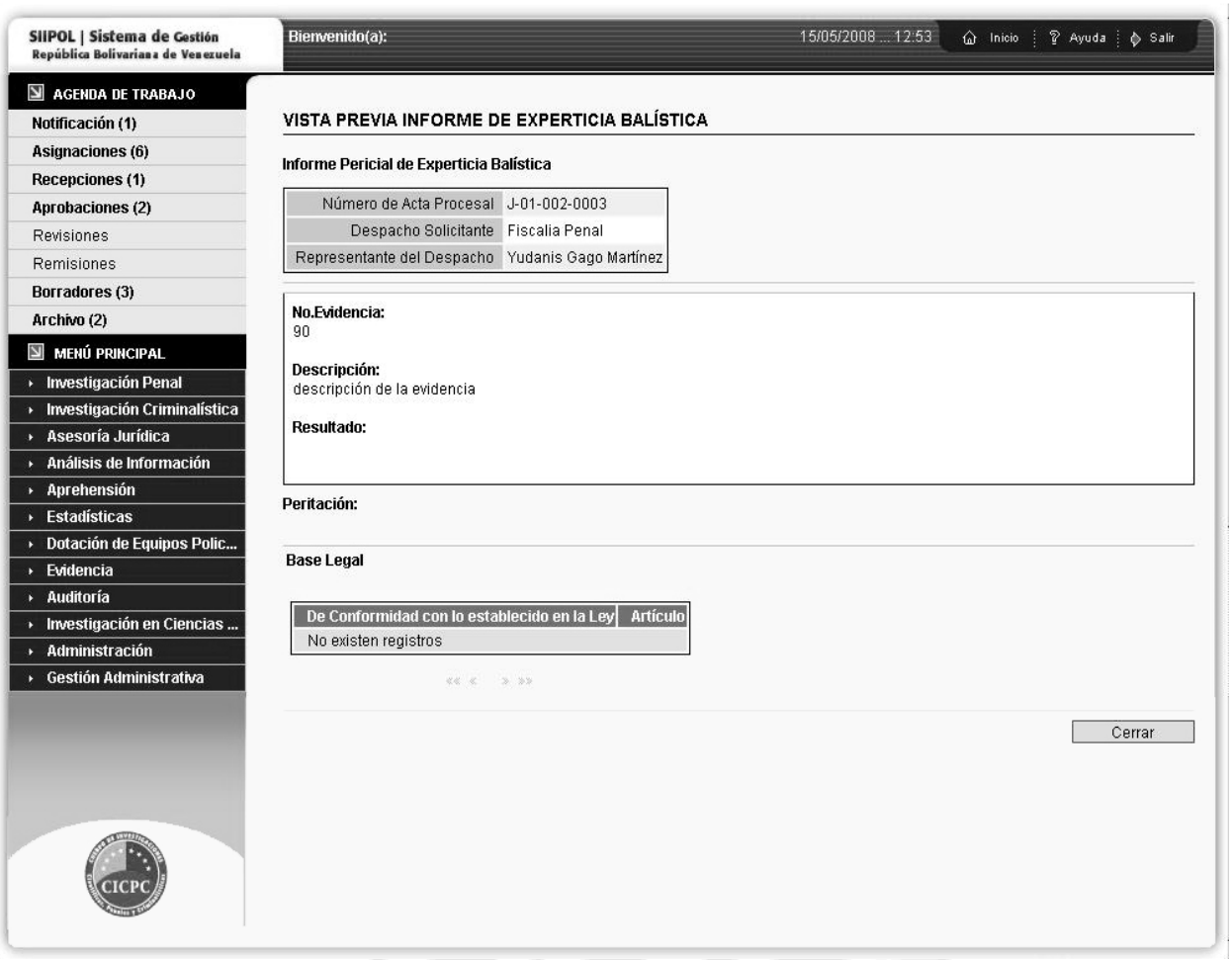

Ilustración 39: Interfaz de usuario Ver Informe de Experticia Balística.

Ilustración 40: Interfaz de Usuario Ver Informe de Microscopia Electrónica.

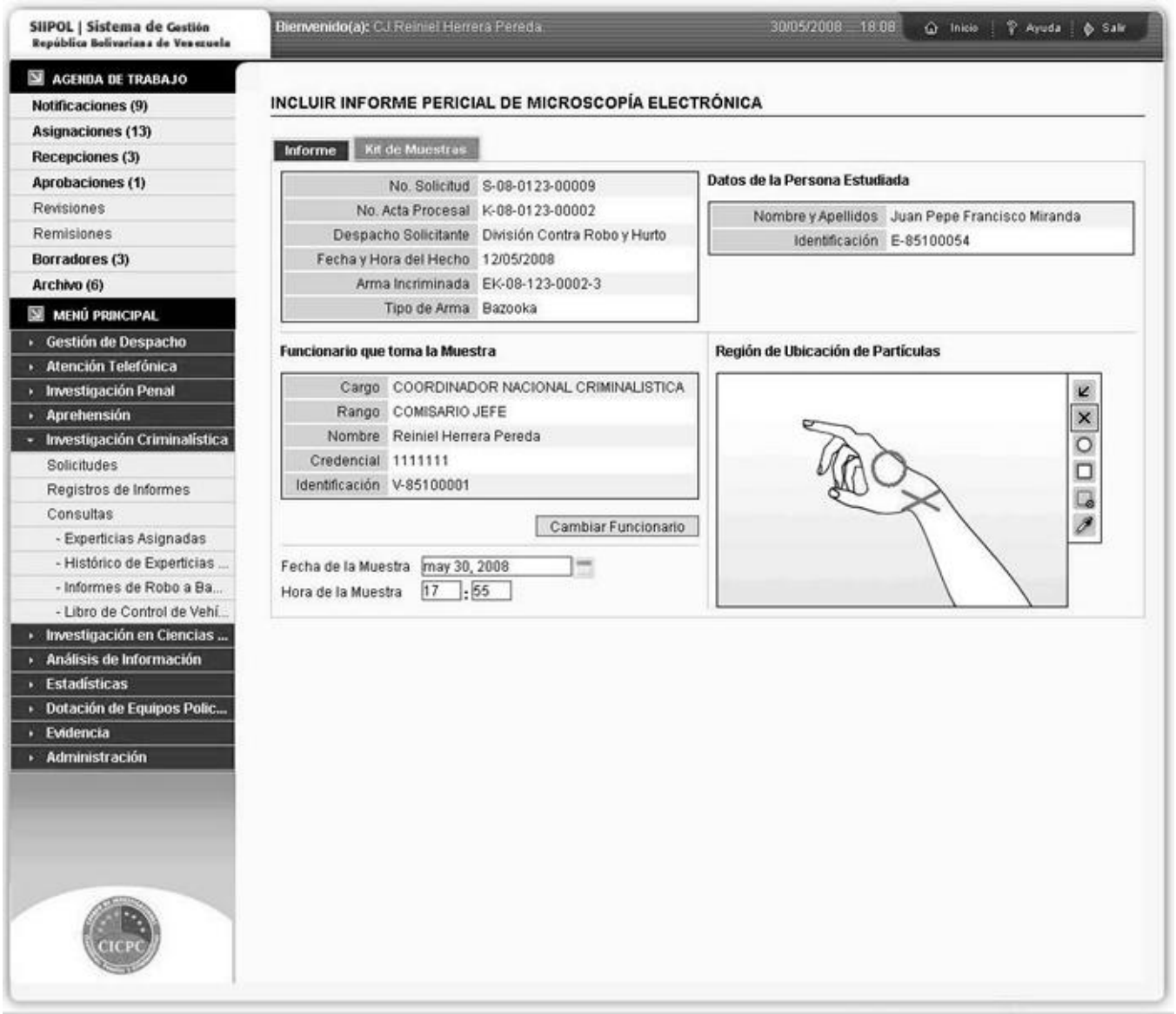

Ilustración 41: Interfaz de Usuario Incluir Informe de Microscopia Electrónica (1).

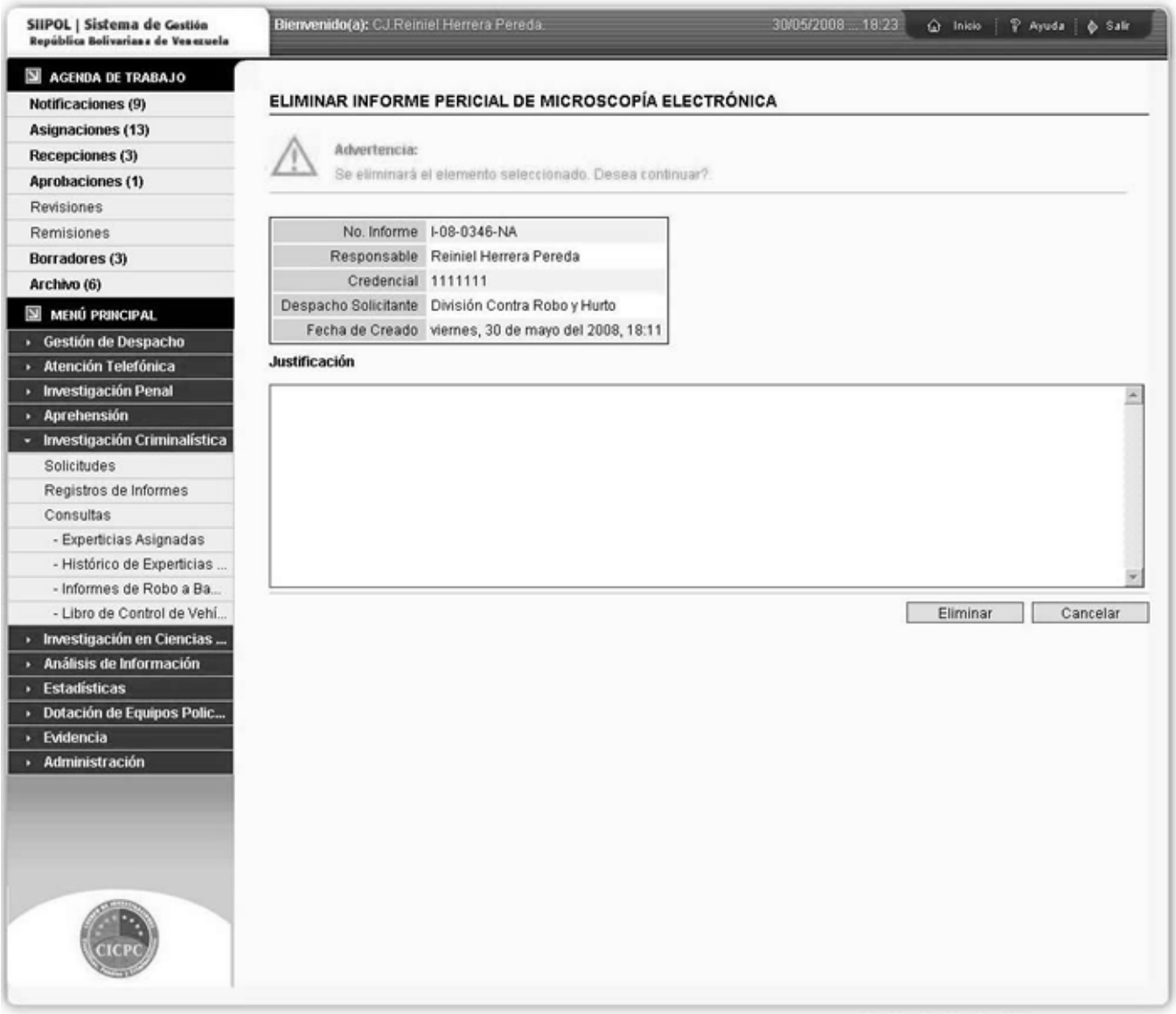

Ilustración 42: Interfaz de Usuario Eliminar Informe de Microscopia Electrónica

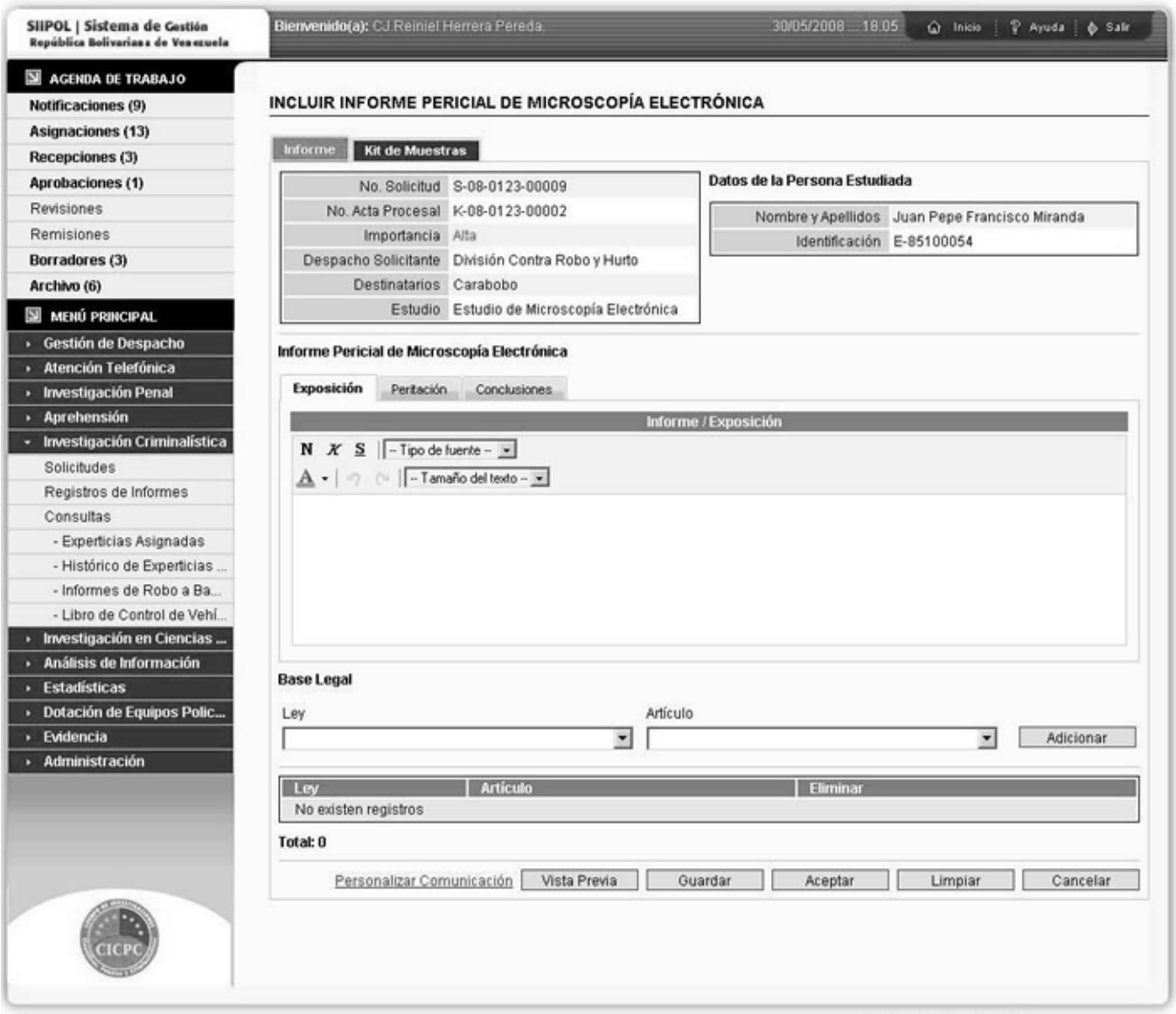

Ilustración 43: Interfaz de Usuario Incluir Informe de Microscopía Electrónica (2).

3.4 Conclusiones

En este capítulo se describieron las especificidades de las técnicas de programación que se utilizaron, de las provistas por los frameworks, después de haber expuesto una muestra de los artefactos generados en la implementación del módulo, además se dio una pequeña muestra de las interfaces de usuario. Una vez terminado el flujo de trabajo los casos de uso pasan al equipo de calidad interna del proyecto, para validar el módulo como parte del subsistema y del sistema en general, el trabajo de pruebas realizado se va del alcance de este documento.

Conclusiones

El resultado principal del trabajo actual es un módulo de software funcional integrado al SIIPOL, con toda la documentación generada durante su diseño e implementación, esta es la prueba más fehaciente del cumplimiento de los objetivos a grandes rasgos.

Posterior a su implementación, el módulo ha pasado por diferentes iteraciones del flujo de trabajo de Pruebas de RUP y se han corregido una serie de errores existentes, dándose el visto bueno por parte del equipo de trabajo de control de la calidad, que ha validado el sistema, cuya primera versión está lista para pasar las pruebas de aceptación; conjunto de acciones que permiten a una representación del cliente dar el sí al sistema propuesto.

Cada uno de los objetivos se fueron cumpliendo en el tiempo dedicado al trabajo en el módulo, y a la tarea de sentar las bases teóricas para tener un dominio de la totalidad del contexto de desarrollo se le prestó amplia atención, por constituir la base a partir de la cual se diseñó e implementó el resultado de la ingeniería de requerimientos. El modelo de sistema, criticable como cualquier otro trabajo dentro del proceso de construcción de software, fue estudiado detalladamente, de manera iterativa, y por cada Caso de Uso, el resultado fue una retroalimentación que permitió corregir detalles pasados por alto en el trabajo que antecede al que se describe en este documento. Por otra parte se logró una familiarización con los procesos que se automatizaron y los conceptos que se manejaron por parte de los desarrolladores.

A partir de este conocimiento adquirido, fue una tarea más que nada de dedicación y esmero diseñar y codificar el conjunto de clases que dan cumplimiento a los requisitos funcionales y no funcionales asociados a los Casos de Uso del módulo Experticias Criminalísticas. Otra tarea muy importante fue la integración al sistema, que se logró llevando a cabo buenas prácticas de diseño e implementación, díganse patrones, convenciones de código y documentación del mismo, llevando a correcto término todos los objetivos propuestos como cumplimiento de las actividades planificadas.

Por todos los resultados obtenidos, una parte de los cuales ha quedado expuesto en este trabajo y el resto se puede consultar en la documentación del proyecto, se concluye que los objetivos del trabajo: Analizar, Diseñar e Implementar un módulo que cumpliera los requisitos relacionados con las experticias criminalísticas del SIIPOL, han sido cumplidos en su cabalidad.

Recomendaciones

- Realizar una refactorización de la implementación de los casos de uso del sistema para optimizar el rendimiento de la aplicación de cara al cliente.
- Consultar para mayor comprensión del tema tratado la documentación del proyecto.
- Consultar a los diseñadores y programadores a la hora de la creación de medios visuales, para la documentación de apoyo al usuario, debido al alto nivel de familiarización con el sistema que obtuvieron en el proceso de desarrollo.
- Capacitar nuevos programadores para las próximas etapas de desarrollo del sistema, apoyándose en la experiencia de los actuales.
- Tener en cuenta el resultado de la implementación de módulo y del sistema en general a la hora de realizar proyectos de iguales características, siempre cumpliendo con las políticas de confidencialidad de la información tratada.
- Sugerir un proyecto similar para la aplicación de las Ciencias Criminalísticas en Cuba.

Referencias Bibliográficas

1. **Rodriguez Baryolo, Yunexis y Monagas Reyes, Miguel Angel.** *Trabajo de Diploma: Ingeniería de Requerimientos del proceso de Criminalística del CICPC.* Ciudad de la Habana : UCI, 2007.

2. Dmapas. *Dmapas.* [En línea] 2008. [Citado el: 10 de enero de 2008.] http://dmapas.cl/productos_stegpol.htm.

3. **Kruchten, Philippe.** *Rational Unified Process, An Introduction, Third Edition.* 2003.

4. **Penadés, Patricio Letelier y Mª Carmen.** *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* España : Universidad Politécnica de Valencia, 2006.

5. LaWebDelProgramador. [En línea] [Citado el: 6 de Marzo de 2008.] http://www.lawebdelprogramador.com.

6. **James Rumbaugh, Ivar Jacobson, Grady Booch.** *El lenguaje Unificado de Modelado. Manual de Referencia.* s.l. : Addison Wesley.

7. Montejava.es. [En línea] [Citado el: 20 de Enero de 2008.] http://www.montejava.es/articulo15.asp.

8. Course. [En línea] [Citado el: 20 de Febrero de 2008.] http://www.course.com.

9. Sitio Web oficial Visual-Paradigm. *Sitio Web oficial Visual-Paradigm.* [En línea] 2008. [Citado el: 12 de enero de 2008.] http://www.visual-paradigm.com/product/vpuml/.

10. java.sun.com. [En línea] [Citado el: 21 de Febrero de 2008.] http://java.sun.com.

11. SlideShare.net. *SlideShare.net.* [En línea] [Citado el: 20 de Enero de 2008.] http://www.slideshare.net.

12. **Mann, Kito.** *JavaServer Faces in Action.* s.l. : Manning, 2005.

13. jboss.org. *jboss.org.* [En línea] [Citado el: 25 de Enero de 2008.] http://www.jboss.org.

14. **Walls, Craig.** *Spring in Action.* s.l. : Manning, 2005.

15. **Red Hat Inc.** SeamFramework.org. *SeamFramework.org.* [En línea] 2008. [Citado el: 15 de febrero de 2008.] http://www.seamframework.org/.

16. mygnet.net. *mygnet.net.* [En línea] [Citado el: 25 de Enero de 2008.] http://www.mygnet.net.

17. Ibatis. [En línea] [Citado el: 10 de Enero de 2008.] http://ibatis.apache.org/.

18. **González, Héctor Suárez.** *Manual Hibernate.* s.l. : Java Hispano, 2003.

19. emagister.com. *emagister.com.* [En línea] [Citado el: 20 de Enero de 2008.] http://www.emagister.com.

20. msdn. *msdn.* [En línea] [Citado el: 4 de Febrero de 2008.] msdn.microsoft.com.

21. Lycos. [En línea] [Citado el: 21 de Enero de 2007.] http://usuarios.lycos.es/neossoftware/articles/j2se/spring.php.

22. Adictos al trabajo. *Adictos al trabajo.* [En línea] 9 de Abril de 2007. [Citado el: 10 de diciembre de 2007.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=Ajax4Jsf.

23. *Conferencias de ISW.* **UCI, departamento de Ingeniería de Software.** 2007-2008.

24. Programación en Castellano. [En línea] [Citado el: 25 de Enero de 2007.] http://www.programacion.net.

Bibliografía

Adictos al trabajo. *Adictos al trabajo.* [En línea] 9 de Abril de 2007. [Citado el: 10 de diciembre de 2007.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=Ajax4Jsf.

Advanced Development Methods, Inc (ADM) . SCRUM. [En línea] 2008. [Citado el: 11 de 2 de 2008.] http://www.controlchaos.com/.

CHRISTIAN BAUER, GAVIN KING. *Java Persistence with Hibernate.* s.l. : Manning Publications Co., 2007.

Conferencias de ISW. **UCI, departamento de Ingeniería de Software.** 2007-2008.

Course. [En línea] [Citado el: 20 de Febrero de 2008.] http://www.course.com.

David Geary, Cay Horstmann. *Core JavaServer™ Faces, Second Edition.* s.l. : Prentice Hall, 2007.

Dmapas. *Dmapas.* [En línea] 2008. [Citado el: 10 de enero de 2008.] http://dmapas.cl/productos_stegpol.htm.

Eclipse. Eclipse. *Eclipse.* [En línea] 2008. [Citado el: 11 de febrero de 2008.] http://www.eclipse.org/.

emagister.com. *emagister.com.* [En línea] [Citado el: 20 de Enero de 2008.] http://www.emagister.com.

González, Héctor Suárez. *Manual Hibernate.* s.l. : Java Hispano, 2003.

Iafis Argentina. *Iafis Argentina.* [En línea] [Citado el: 10 de Abril de 2008.] http://www.iafis.com.ar/html/afis.html.

Ibatis. [En línea] [Citado el: 10 de Enero de 2008.] http://ibatis.apache.org/.

Jalon, Javier Garcia de. *Aprenda Java como si estuviera en primero.* San Sebastian : Escuela Superior de Ingenieros Industriales, 2000.

James Rumbaugh, Ivar Jacobson, Grady Booch. *El lenguaje Unificado de Modelado. Manual de Referencia.* s.l. : Addison Wesley.

java.sun.com. [En línea] [Citado el: 21 de Febrero de 2008.] http://java.sun.com.

Javahispano.org. [En línea] [Citado el: 16 de Enero de 2008.] http://www.javahispano.org/contenidos/es/comparativa_j2ee___net/.

jboss.org. *jboss.org.* [En línea] [Citado el: 25 de Enero de 2008.] http://www.jboss.org.

Kruchten, Philippe. *Rational Unified Process, An Introduction, Third Edition.* 2003.

LaWebDelProgramador. [En línea] [Citado el: 6 de Marzo de 2008.] http://www.lawebdelprogramador.com.

Lycos. [En línea] [Citado el: 21 de Enero de 2007.] http://usuarios.lycos.es/neossoftware/articles/j2se/spring.php.

Mann, Kito. *JavaServer Faces in Action.* s.l. : Manning, 2005.

María A. Mendoza Sanchez. Ing. Informático, Microsoft Certified Professional ,Analísta y Desarrolladora - TeamSoft. *Metodologías De Desarrollo De Software.* Peru : s.n., 2004.

Molpeceres, Alberto. Procesos de desarrollo: RUP, XP y FDD. *Willi.net.* [En línea] 2002. [Citado el: 10 de enero de 2008.] http://www.willydev.net/descargas/articulos/general/cualxpfddrup.PDF.

Monografias.com. [En línea] [Citado el: 15 de Enero de 2008.] http://www.monografias.com/trabajos24/arquitectura-cliente-servidor/arquitectura-cliente-servidor.shtml.

Montejava.es. [En línea] [Citado el: 20 de Enero de 2008.] http://www.montejava.es/articulo15.asp.

msdn. *msdn.* [En línea] [Citado el: 4 de Febrero de 2008.] msdn.microsoft.com.

mygnet.net. *mygnet.net.* [En línea] [Citado el: 25 de Enero de 2008.] http://www.mygnet.net.

NetBeans. NetBeans. *NetBeans.* [En línea] 2008. [Citado el: 12 de febrero de 2008.] http://www.netbeans.org.

Penadés, Patricio Letelier y Mª Carmen. *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* España : Universidad Politécnica de Valencia, 2006.

Proactiva. [En línea] [Citado el: 5 de Febrero de 2008.] http://www.proactiva-calidad.com.

Programación en Castellano. [En línea] [Citado el: 25 de Enero de 2007.] http://www.programacion.net.

Red Hat Inc. SeamFramework.org. *SeamFramework.org.* [En línea] 2008. [Citado el: 15 de febrero de 2008.] http://www.seamframework.org/.

Rodriguez Baryolo, Yunexis y Monagas Reyes, Miguel Angel. *Trabajo de Diploma: Ingeniería de Requerimientos del proceso de Criminalística del CICPC.* Ciudad de la Habana : UCI, 2007.

Scribd. [En línea] [Citado el: 20 de Enero de 2008.] http://www.scribd.com.

Scrum Alliance, Inc. ScrumAlliance. [En línea] [Citado el: 11 de febrero de 2008.] http://www.scrumalliance.org/.

Sitio Web oficial Visual-Paradigm. *Sitio Web oficial Visual-Paradigm.* [En línea] 2008. [Citado el: 12 de enero de 2008.] http://www.visual-paradigm.com/product/vpuml/.

SlideShare.net. *SlideShare.net.* [En línea] [Citado el: 20 de Enero de 2008.] http://www.slideshare.net.

UCI, Cátedra Historia de la Informática. *Conferencia de Historia de la Informática: Historia de los lenguajes de programación.* Ciudad de la Habana : s.n., 2008.

UCI, departamento de Programación. *Programación 3: Conferencia 1.* 2008.

WALLS, CRAIG. *Spring in Action, Second Edition.* s.l. : Manning Publications Co., 2008.

Walls, Craig. *Spring in Action.* s.l. : Manning, 2005.

Wikipedia.org. *Wikipedia.org.* [En línea] [Citado el: 18 de Enero de 2008.] http://es.wikipedia.org.

Anexos

Glosario

A continuación, en orden alfabético, se muestra el significado de algunos términos usados en este documento que pueden dificultar la compresión del mismo:

ACID: En bases de datos ACID es la propiedad de realizar transacciones seguras. Es un acrónimo de Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad en español.

AFIS: Sistema Automático de Identificación de Huellas Dactilares, sistema informático compuesto de Hardware y Software integrados que permite la captura, consulta y comparación automática de huellas dactilares agrupadas por fichas decadactilares, o en forma de rastro latente (parte degradada de huella levantada en la escena de un crimen) (22).

Agraviado: Toda persona que sufre una lesión jurídica.

AJAX: Es una técnica de desarrollo para crear aplicaciones web, mediante la cual un grupo de acciones se ejecutan en navegador de los usuarios y mantiene comunicación asíncrona con el servidor en segundo plano.

Ajax4JSF: Es un framework libre y Open Source que adiciona capacidades a las páginas JSF, a través de componentes implementados.

Apache TomCat: es un servidor web con soporte de servlet y JSP. Incluye el compilador Jasper, que compila las páginas JSP convirtiéndolas en servlet.

API: es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

ApplicationContext: Clase del framework Spring que se encarga de gestionar el manejo de beans (23).

Autopsia: Significa estudio de la muerte, a través de la apertura de todas las cavidades del organismo y el examen detallado de todos los órganos.

Avalúo: Valor estimado de un bien.

Bean: En el lenguaje Java es un componente software reutilizable que evita programar los distintos componentes uno a uno. Existen con la finalidad de ahorrar tiempo al programar.

BeanFactory: En el framework Spring para Java, es una clase que implementa la familia de patrones de diseño Factory, que aplica la inversión del control para separar las dependencias entre los beans del código de la aplicación.

Bytecode: Es un código intermedio más abstracto que el código máquina. Habitualmente es tratado como un fichero binario que contiene un programa ejecutable.

Cédula: Medio de identificación de un individuo venezolano.

CICPC: Cuerpo de Investigaciones Científicas Penales y Criminalísticas de Venezuela, institución gubernamental encargada de investigar hechos delictivos.

CLR: Common Language Runtime, es el núcleo de la plataforma .NET. Es el motor encargado de gestionar la ejecución de las aplicaciones para ella desarrolladas y a las que ofrece numerosos servicios que simplifican su desarrollo y favorecen su fiabilidad y seguridad.

CNC: Coordinación Nacional de Criminalística: coordinación del CICPC encargada de los asuntos de investigaciones criminalísticas.

DAO: En español Objeto de Acceso a Datos, es un Patrón de diseño de clases en ingeniería de software que permite a quien lo aplique suministrar una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos.

Dependency Injection: En español inyección de dependencia, es una manera de lograr la inversión de control, utilizada por Spring para manipular las dependencias entre los objetos.

Enterprise Java Bean: Es una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE.

Evidencia: Objeto involucrado en una falta disciplinaria o investigación penal, que aporta información sobre la forma o motivo en que ocurrió el hecho.

Faces Servlet: Es el Servlet para las componentes visuales del framework JSF.

Factory: Familia de patrones cuya implementación tiene la responsabilidad de crear instancias de objetos de otras clases. Tienen además la responsabilidad y el conocimiento necesario para encapsular la forma en que se crean determinados tipos de objetos en una aplicación (24).

Framework: Es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas

y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Herramienta CASE: Aplicación informática destinada a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero, se utiliza para la modelación del sistema.

Hibernate: Framework de capa de persistencia para el lenguaje Java.

HQL: Lenguaje de consultas del framework Hibernate similar al SQL, pero que se refiere a clases y objetos no a tablas de la base de datos.

HttpServlet: Clase de Java que hereda de la clase GenericSevlet que presenta los métodos necesarios para manejar las peticiones de una aplicación web en el servidor.

iBatis: Framework de capa de persistencia para el lenguaje Java (17).

IBIS: Sistema Integrado de Identificación Balística es una herramienta utilizada en la CNC para almacenar, en una base de datos, las características microscópicas de las evidencias relacionadas con proyectiles, armas de fuego, lugar de los hechos y otros (1).

IDE: Entorno Integrado de Desarrollo, es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse exclusivamente para un lenguaje de programación o bien para varios (6)

J2EE: Es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java con arquitectura de N niveles distribuida.

JBoss Seam: Es un framework que integra la capa de presentación con la capa de negocios y persistencia

JDBC: Es el acrónimo de Java Database Connectivity, un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice (15).

JRE: es el acrónimo de Java Runtime Environment (entorno en tiempo de ejecución Java) y se corresponde con un conjunto de utilidades que permite la ejecución de programas Java sobre todas las plataformas soportadas.

JSF: framework de la capa de presentación para Java.

LoC: Inversión del control, técnica de programación utilizada para manejar las dependencias entre objetos.

MVC: Es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones web.

NetBeans: Entorno Integrado de Desarrollo para crear aplicaciones en lenguaje Java.

Open Source: Representa el software de dominio público, esto significa sin licencia, cuyo código fuente está disponible y se le permite usar y modificar.

ORM: 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.

Page Controller: Patrón de diseño para aplicaciones web, que promueve el uso de una clase para manejar las peticiones de una página en el contexto del patrón MVC.

Plataforma Win32: Es una versión del conjunto de librerías del sistema operativo Windows.

Proxy: Patrón de diseño que su implementación utiliza un objeto como intermediario para acceder a un objeto, permitiendo controlar el acceso a él (17).

Rich-**Faces**: Conjunto de librerías para el framework JSF.

Service Locator: Patrón de diseño que provee un punto de acceso centralizado a determinados servicios

Servlet: Es un objeto que se ejecuta en un servidor o contenedor JEE, fue especialmente diseñado para ofrecer contenido dinámico desde un servidor web

SIGEP: Sistema de Gestión de Penitenciaria.

SIIPOL: Sistema de Investigación e Información Policial.

Siniestros: Daños o perjuicios a las personas o bienes asegurados que se encuentran cubiertos por una póliza de seguros.

Spring: Framework de la capa de lógica de negocio para Java.

SQL-**MAP**: Framework de la capa de persistencia de Java.

STEGPOL: Es un Sistema de Información Geográfica con tecnología de punta chileno.

Struts: Es un framework de la capa de presentación que implementa el patrón MVC en Java

Testigo: Persona que tiene conocimiento de un hecho, por haber estado relacionado al mismo directa o indirectamente.

Validadores: En el framework JSF son utilizados para verificar la validez de los datos enviados al servidor.