FACULTAD 8

Análisis, Diseño e Implementación del submódulo Evidencia del módulo Registro y Control del SIIPOL.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

AUTORES

Amed Vázquez Pérez

Arlan Galvez Alonso

TUTORAS

Ing. Diana García Vicente

Ing. Susel Ruiz Durán

Ciudad de La Habana, 2009

"Año del 50 Aniversario del Triunfo de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaramos que somos los autores de este trabajo y autorizamos a la Facultad 8 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 firmamos la presente a los ____ días del mes de _____ del año 2009.

AUTOR	**AUTOR**
Arlan Galvez Alonso	Amed Vázquez Pérez
TUTOR	**TUTOR**
Ing. Diana García Vicente	Ing. Susel Ruiz Durán

AGRADECIMIENTOS

Agradecer es muy poco, pues habría que escribir más de las páginas que tiene este documento, pero trataremos de hacerlo en este corto espacio.

De Arlan:

…A mi gran familia por apoyarme en todo momento, aún en los más difíciles, pues para mí lo son todo en esta vida… A Tere, por admitirme la comunicación con ellos a lo largo de estos años... A mis segundos hermanos Aldo y Alberto que me han permitido formar parte de sus familias y de sus vidas…A Greisy por los consejos y el apoyo… A Sergio René por la ayuda, los consejos y la fuerza, aún estando lejos… a todas esas personas lindas que han sido parte de mi vida desde que comencé mis estudios en la Universidad…

De Amed:

…A mi familia pues son parte importante en mi vida… a mi novia por el amor y la ayuda brindados…A mis amigos más allegados por la constancia… A mis compañeros de grupo por el tiempo compartido…

De ambos:

A nuestras tutoras Diana y Susel, por haber confiado en nosotros para la realización de este trabajo y por dedicarnos tanto tiempo y esfuerzo.

Al líder indiscutible de nuestra Revolución, el compañero Fidel por darnos la oportunidad de formarnos en esta gran institución que lo es la UCI.

A todos los que de una forma u otra han aportado o quitado su granito de arena en nuestra formación personal y profesional a lo largo de estos años.

DEDICATORIA

A nuestros padres, sentir y latir de nuestros corazones.

RESUMEN

La presente investigación refleja el proceso de desarrollo del submódulo Evidencia del subsistema Registro y Control del SIIPOL, solución integral que se brinda al CICPC como parte de los convenios de colaboración Cuba-Venezuela.

El Departamento de Evidencias ha sido una de las nuevas áreas evaluadas para posibles informatizaciones dentro del CICPC, dentro del cual se realizan un grupo de actividades sobre la evidencia que comienzan desde que es colectada en el sitio del suceso, hasta que se le deja de considerar valiosa para la investigación. En la actualidad en el sistema informático que se utiliza no se cubren estos procesos de vital importancia, por lo que el procesamiento del gran volumen de información generada se realiza de forma manual. Esto influye negativamente en los resultados de los procesos judiciales y la presentación de elementos probatorios en los juicios.

Como solución al problema planteado se desarrolló una aplicación web, basada en lenguaje Java, Oracle 10g como gestor de base de datos y usando tecnología AJAX. Con el uso de la metodología de desarrollo de software RUP como guía para el desarrollo, se efectúa el análisis, diseño e implementación, obteniéndose como resultado una aplicación que cumple con los requisitos obtenidos como resultado del proceso de Ingeniería de Requerimientos aplicado al Departamento de Evidencias. Se valida la propuesta de solución dada mediante un conjunto de pruebas realizadas, obteniéndose los resultados esperados.

ABSTRACT

This research reflects the *Evidencia* submodule, from subsystem *Registro y Control*, development process. This subsystem belongs to integral solution *SIIPOL* who has been developed as part of a collaboration agreement between Cuba and Venezuela.

The Department that manages the evidences (Evidence Department) has been one of the areas evaluated for possible process automation within CICPC. Several activities take place in this department, starting from the evidence collection at the crime scene until it is no longer considered valuable for investigation. The currently used software application does not cover these vital processes, so the huge volume of information is processed manually. This fact negatively affects the outcome of legal proceedings and the presentation of evidence at trial.

As a solution to the problem, a web application based on Java, Oracle 10g as database manager and using AJAX technology was developed. Using the software development methodology RUP as a guide to development, analysis, design and implementation were performed, obtaining an application that covers the requirements generated by applying requirements engineering to the Evidence Department processes. The proposed solution has been validated by a set of tests, yielding the expected results.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Las ***Evidencias*** son aquellos objetos tangibles o intangibles propensos a ser examinados por un experto, para interpretar o aclarar una situación delictiva. Ellas implican la materialidad de los elementos objeto de análisis en la Criminalística para solucionar lo más técnicamente posible interrogantes surgidas en la Investigación Criminal, por lo que permiten, fundamentalmente:

a) La identificación de él o los autores.
b) Las pruebas de la comisión del hecho.
c) La reconstrucción del hecho.
d) Las circunstancias bajo las cuales ocurrió el hecho.

Se definen así con la finalidad de esclarecer o diferenciarlas de otros tipos de elementos que, desde el punto de vista jurídico, se refieren a evidencias circunstanciales o testimoniales. Todas ellas pueden ser usadas en un proceso judicial como pruebas para determinar la culpabilidad o inocencia de una persona, pero si son tratadas de forma incorrecta en los procesos que se le realizan, podrían perder su validez legal y ser descartadas, pudiendo acarrear consigo el retraso de la investigación, el juicio y el caso en general.

En la República Bolivariana de Venezuela no existe información sobre la frecuencia con que se excluyen pruebas por estas razones, ni sobre cómo estos errores prevenibles afectan la actuación de la policía y su acatamiento de la normativa. En tal punto se hace necesario crear un proceso que permita conocer si la evidencia fue tratada correctamente desde su recolección hasta que deje de considerársele como tal.

Por su parte el ***proceso de Cadena de Custodia*** es la fuerza o cualidad probatoria de la evidencia. Este será el encargado de probar (si fuese requerido por el Tribunal) que la evidencia presentada es realmente la misma evidencia recuperada en el sitio del suceso, obtenida del testigo, la víctima o el sospechoso, o adquirida originalmente de alguna forma.

El procedimiento de la Cadena de Custodia no estaba estandarizado o definido para los distintos despachos, delegaciones y subdelegaciones del Cuerpo de Investigaciones Científicas, Penales y Criminalísticas (en lo adelante CICPC) de la República Bolivariana de Venezuela, alguno de ellos lo efectuaban y en estos casos se realizaba de forma poco segura, propensa a falsificaciones, pérdidas y mala calidad de la información de los datos que en él se generaba. El grado de dificultad es elevado, debido a que el 75% de los elementos del proceso tienen un formato de documento manuscrito o planilla (memorandos y documentos de Registro de Cambio de Custodia) repercutiendo en el proceso judicial, haciéndolo más largo y costoso. Muchas de las solicitudes de experticia asociadas la evidencia no se responden debido a que no se recogen las evidencias en el departamento de Resguardo de Evidencia.

La Coordinación Nacional de Criminalística (CNC) es la encargada de planificar, coordinar y dirigir todos los procesos técnico-científicos de las investigaciones. Actualmente existe una situación problemática constituida por la lentitud en el flujo 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.

En las dependencias de la CNC, el proceso de las evidencias es controlado solamente con los libros de entrada, los que no guardaban constancia de la autenticidad de la información registrada por el oficial de guardia.

El nivel de seguridad y veracidad de la información contenida en las Actas de Inspección Técnica y las Investigaciones Penales y Forenses relacionadas con las evidencias es bajo y poco confiable para sustentar un caso judicial, influenciando de forma negativa en los procesos investigativos realizados por los expertos para resolver los casos delictivos.

Luego de firmar el contrato para la realización de la nueva versión del Sistema Integrado de Información Policial, el departamento de Evidencias adscrito al CICPC fue una de las nuevas áreas a informatizar y, tras un largo período de análisis con los especialistas en el área, se determinó una estructura estándar para todas las áreas y departamentos del CICPC que trabajan las evidencias y el proceso de cadena de custodia de las mismas. Se realizaron un conjunto de entrevistas y encuestas de las cuales se extrajeron los requisitos funcionales y no funcionales que debía cumplir el submódulo Evidencia para cumplir con las necesidades del cliente en el nuevo Sistema de Investigación e Información Policial (en lo adelante SIIPOL).

En función de desarrollar un sistema informático que tribute a dicha institución, se propone como **problema científico: ¿Cómo garantizar el cumplimiento de los requisitos funcionales y no funcionales asociados al submódulo Evidencia perteneciente al módulo Registro y Control del SIIPOL?**

Lo que hace que se planteen los siguientes **objetivos:**

General

Desarrollar una herramienta informática que garantice el cumplimiento de los requisitos funcionales y no funcionales asociados al submódulo Evidencia perteneciente al módulo Registro y Control de SIIPOL.

Específicos

- ✓ Analizar los Casos de Uso del submódulo Evidencia del Módulo Registro y Control del SIIPOL.
- ✓ Diseñar e implementar cada una de las capas arquitectónicas de modo que sean capaces de dar respuesta satisfactoriamente a todos los requisitos funcionales y no funcionales de la aplicación.

El objeto de estudio es el funcionamiento de los requisitos funcionales y no funcionales del módulo de Registro y Control perteneciente al SIIPOL, y el **campo de acción** del mismo es el análisis, diseño e implementación del submódulo Evidencia perteneciente al módulo Registro y Control del SIIPOL.

Para dar solución respuesta al problema se plantea la siguiente **idea a defender:**

La aplicación correcta de la metodología de desarrollo de software adecuada y buenas prácticas de diseño orientado a objetos para el análisis, diseño e implementación submódulo Evidencia del módulo Registro y Control, posibilitará la elaboración de un sistema informático que cumpla con los requisitos funcionales y no funcionales definidos para el mismo.

Tareas Investigativas a desarrollar:

- Estudiar el estado del arte.
  - ✓ Revisar la bibliografía referente al tema.
  - ✓ Analizar la información Bibliográfica recopilada.
  - ✓ Realizar un análisis de otras aplicaciones o soluciones similares.

- Elaborar el diseño teórico de la investigación.
  - ✓ Definir la situación problémica, problema, objetivos.
  - ✓ Analizar la Arquitectura propuesta para el SIIPOL.

- Elaborar la propuesta de solución.

- ✓ Realizar el Análisis de la Descripción de los Casos de Uso de forma detallada.
- ✓ Realizar el diseño de las Clases pertinentes, paquetes y componentes necesarios para la solución.
- ✓ Elaborar modelos pertinentes.
- ✓ Implementar las funcionalidades necesarias para dar solución a los requerimientos funcionales y no funcionales de cada caso de uso.

- Estructurar el documento tesis.
  - ✓ Estructurar y organizar los Capítulos, epígrafes.

El documento de tesis está estructurado por capítulos de la siguiente forma:

- ✓ **Capítulo 1. Fundamentación Teórica:** En este capítulo se abordan elementos teóricos relacionados con la investigación como son la metodología a utilizar, herramientas y elementos para el desarrollo sin dejar de mencionar sistemas similares existentes en el campo de acción.

- ✓ **Capítulo 2. Análisis, diseño e implementación de la propuesta de solución:** En este capítulo se hace referencia a los flujos de trabajo de análisis, diseño e implementación, los artefactos que se generaron producto de aplicarle la Ingeniería de Requerimientos al submódulo en cuestión, sin dejar de dar una breve explicación de los temas que se tratan en cada uno de los acápites recogidos dentro de los temas principales.
- ✓ **Capítulo 3. Validación de la solución propuesta:** En este capítulo se pretende, mediante las pruebas que se le aplicaron a la solución final, validar la solución propuesta quedando evidenciado que la solución dada es válida para los usuarios finales.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

1.1. INTRODUCCIÓN

Los primeros pasos de desarrollo del software están destinados, entre otras cosas, para definir el tipo de aplicación a realizar y el ambiente de desarrollo de la misma. Con el presente Capítulo se desea fundamentar teóricamente el desarrollo del submódulo Evidencia del Módulo Registro y Control del SIIPOL, definiéndose en el mismo la fundamentación teórica del tema con análisis de sistemas similares existentes, así como el análisis de las herramientas y metodologías seleccionadas por el cliente para el desarrollo de la aplicación final con una breve introducción a lo al tema de los Sistemas de Gestión de Información Policial.

1.2. SOFTWARE DE GESTIÓN POLICIAL

Un ***Sistema de Gestión de Información*** es un producto capaz de integrar todo un conjunto de actividades que se realizan dentro de una organización o empresa. Se utilizan para automatizar procesos y dar apoyo a otras acciones, con el objetivo de mejorar el tiempo de respuesta a las necesidades de los clientes u otras entidades, para acceder concurrentemente a documentos e información, minimizar el tiempo de acceso a información crucial, así como al análisis de estadísticas y resultados de investigaciones para la toma de decisiones, generar documentos automáticamente y dar solución a la problemática planteada, reducir riesgos de pérdida de documentos de vital importancia para la empresa, proporcionando grandes beneficios en todas las áreas de la misma. Recolectan gran cantidad de información la cual luego puede ser analizada y explotada. Un Sistema de Gestión de información de calidad debe ser de fácil utilización por los usuarios sin necesidad de intervención de un especialista en la rama para el desarrollo de los procesos que controla, con el uso de una interfaz visual de acuerdo a las necesidades del cliente final y la utilización de técnicas de programación avanzada para la implementación de las mismas.

Un ***Sistema de Gestión de Información Policial*** se restringe solamente a los procesos en el área de las entidades policiales, incrementando la satisfacción del cliente y reduciendo los costes legales, protegiendo información sensible referente a la población y del país en general. El éxito de estos sistemas está dado por la confidencialidad y de seguridad, se conoce muy poco de las herramientas y metodologías de desarrollo de este tipo de software; es por ello que la información publicada sobre los mismos es escueta y básica.

Luego de un proceso de búsqueda detallada referente a algunos sistemas similares que pudieran apoyar la elaboración de la propuesta de solución, se encontraron los siguientes:

SISTEMA TERRITORIAL DE EMERGENCIAS Y GESTIÓN POLICIAL (STEGPOL)

Es un Sistema de Información Geográfica 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 hablando, se están cometiendo o se han cometido actos delictivos a nivel territorial, apoyado por sistemas de información en línea desde el lugar de los hechos; lo cual permite la acción rápida y coordinada entre los diferentes actores encargados de la seguridad ciudadana a nivel nacional. Integra 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 están interconectadas entre sí, en especial con diferentes Divisiones o Departamentos de Carabineros, tales como Jefaturas de Zona, Prefecturas, Comisarías, Sub-Comisarías, Tenencias y por último los Retenes.

Vía la Intranet del Estado o vía comunicación en Banda Ancha - Internet se puede llegar a acceder a dicho 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. (1)

SISTEMA DE GESTIÓN PENITENCIARIA (SIGEP)

Proyecto productivo que es desarrollado por la Universidad de las Ciencias Informáticas, tributando al proceso de Informatización y modernización de las Instituciones de la República Bolivariana de Venezuela a raíz de los acuerdos del ALBA. Tiene gran similitud en cuanto a las tecnologías solicitadas por el cliente y el equipo de desarrollo, sus funcionalidades y procesos están dirigidas principalmente a las prisiones venezolanas, pero su arquitectura y herramientas constituyeron el punto de partida para el proceso investigativo previo para determinar cuáles usar en el desarrollo del SIIPOL. Para la elaboración del mismo utilizaron como plataforma de programación Java manejando los frameworks Acegi (actualmente SpringSecurity) para el manejo de la seguridad, Hibernate para la capa de acceso a Datos, Spring para la de negocio y el módulo Spring MVC para la capa de presentación.

Por cuestiones de seguridad los sistemas antes descritos solo hacen alusión a algunos datos escuetos, debido a que las compañías desarrolladoras y las instituciones que las utilizan se reservan sus propiedades y características para evitar ataques al sistema o la integridad de los datos de aquel que lo usa. Por estas y otras razones se decidió realizar un sistema innovador, flexible y eficiente que permitiera reestructurar los procesos que se llevan a cabo por la institución si fuese necesario.

1.3. EL CUERPO DE INVESTIGACIONES CIENTÍFICAS, PENALES Y CRIMINALÍSTICAS

El Cuerpo de Investigaciones Científicas, Penales y Criminalísticas (CICPC) es la Institución que garantiza la eficiencia en la investigación del delito, mediante su determinación científica,

asegurando el ejercicio de la acción penal que conduzca a una administración de justicia. Con la visión de ser la Institución indispensable, por su reconocida capacidad científica y máxima excelencia de sus recursos, con la finalidad de alcanzar el más alto nivel de credibilidad nacional e internacional en la investigación del fenómeno delictivo organizativo y criminalidad violenta; y con el fin de cumplir el decreto que instituye las competencias del CICPC como órgano principal de investigaciones penales al servicio del Estado.

El Sistema Integrado de Información Policial (SIIPOL) es la aplicación informática que actualmente utiliza el CICPC para realizar los procesos internos que desarrollan a diario en las dependencias, delegaciones y subdelegaciones. Está desarrollado sobre una tecnología actualmente obsoleta con servidores de Base de Datos SUN 6500 y base de datos Adabas[1] con lenguaje de programación Natural para el manejo de los datos; el acceso a este sistema actualmente se hace a través de un emulador, como Personal Comunication, o un Telnet. El servidor de la aplicación es con tecnología de Sun con Sistema Operativo Solaris. Utilizan un programa llamado Natural Security que es el que valida la gestión de los usuarios para la seguridad del SIIPOL.

Para mantener actualizado el sistema, el CICPC cuenta con una división de información policial y varias divisiones de análisis y seguimiento de la información; que se encargan de introducir o actualizar la información que es recopilada por las dependencias, y ayudan a las que no tienen conexión directa con el SIIPOL. Además se encarga de tramitar los datos e informaciones que son requeridos por esas dependencias para hacer las averiguaciones y también suministra información a las fiscalías y tribunales. Aun así todos los casos que actualmente se trabajan en el CICPC no están registrados en la aplicación informática.

Además se utilizan datos de otras instituciones como la Base de Datos de SAIME, y la del INTTT, esta información se trae en soportes digitales para actualizar la Base de Datos del SIIPOL una

[1] **Adabas** (**A**daptable **Da**ta**ba**se **S**ystem) fue creada por la empresa alemana Software AG en el año 1969, usa lista invertidas que provee un alto rendimiento así como el acceso a los datos y la integridad en a Base de Datos. Actualmente muy usando en aplicaciones que requieren de gestión de altos volúmenes de datos o en ambientes de alto procesamiento de transacciones analíticas, en sus últimas versiones (Adabas 2006) integra gateways para SOA y SQL.

vez por semana o una vez al mes. El CICPC a su vez envía información para actualizar las Bases de Datos de esas instituciones.

Luego de un estudio preliminar de la situación general que tiene actualmente el CICPC, existen un conjunto de problemáticas presentes en los procesos que desarrollan, que pueden ser resueltos a través de una aplicación informática:

1. Lentitud en la fluidez de la información entre las diferentes áreas de trabajo del CICPC, que deben coordinar su trabajo en la solución de los casos.
2. Falta de información actualizada, oportuna y fiable para los entes de dirección del CICPC, que no permite el conocimiento táctico sobre el curso de las investigaciones de un caso, ni mejorar su contribución al desarrollo de políticas, estrategias y análisis sobre la criminalidad.
3. Limitaciones en la diversidad de información que se requiere para la investigación de los hechos y en la calidad de uso de la que hoy esta almacenada.
4. Imposibilidad para acceder y utilizar información de interés de otras organizaciones como la SAIME, INTTT, CNE, Registro y Notarias, Penitenciarias, entre otras.

Estas y otras problemáticas se pretenden dar solución con la creación de una nueva aplicación informática que cumpla con las expectativas del cliente.

1.3.1. LOS PROCESOS DE LAS EVIDENCIAS

Los elementos que se encuentran en un sitio de un hecho delictivo son colectados por los expertos para su análisis posterior en busca de indicios que permitan esclarecer el delito cometido. Las evidencias, luego de ser colectadas, se registran, empaquetan y son trasladadas a las dependencias responsabilizadas de análisis, siendo resguardadas hasta que sean entregadas en la dependencia.

Estas luego pasan al Departamento de Resguardo de las Evidencias, los que se responsabilizan por su cuidado hasta que esta sea trasladada o entregada a otro funcionario para así evitar que se pierdan, tanto la evidencia como la documentación relacionada a la misma. Este proceso de cambio de custodia entre funcionarios constituye un problema real, debido a que no se lleva el control debido sobre los documentos que este proceso genera, ni todas las dependencias, delegaciones y sub-delegaciones lo realizan de forma estándar. El registro de la llegada de evidencias a algunos departamentos del CICPC no es el más idóneo debido a que solo se controla por los libros de entrada donde se recogen los datos de la evidencia y los del que hizo la entrega, incluyendo la revisión de los documentos que con ella vienen incluidos.

En el proceso de resguardo y Cadena de Custodia de las Evidencias existen grandes fallos debido a que los datos que se recogen en las planillas son potencialmente incorrectos, falsos o ausencia de alguno considerado de importancia, que como consecuencia puede significar la pérdida del elemento clave para la aclaración del hecho o considerarse que el proceso no fue lo suficientemente seguro que se necesitaba para esclarecer el caso. También los documentos que se generan son llenados manualmente, estando a expensas de que se pierdan en el proceso de traslado, movimiento o de análisis de la Evidencia.

En el tiempo en que la Evidencia se encuentra en el proceso de análisis o de experticia, puede ser consumida en los procesos que se le realizan ya que puede ser un elemento de contenido finito o limitado y puede acabarse en un momento determinado; o bien puede ser desechada porque ya no son viables para realizarlos debido a que pueden ser de consistencia orgánica y descomponerse, no quedando constancia de los hechos y las transformaciones realizadas sobre la evidencia, alternado el resultado de las experticias y pudiendo transformar las características de el elemento y los resultados posteriores.

Luego de realizarle las experticias pertinentes a la Evidencia y de cerrado el caso, esta puede ser devuelta a su propietario o lugar de origen.

Estos procesos antes descritos no se encuentran automatizados en el Sistema que actualmente está en uso y no están estandarizados para todas las delegaciones, sub-delegaciones y dependencias del CICPC.

1.4. EL SISTEMA DE INVESTIGACIÓN E INFORMACIÓN POLICIAL

La solución que se propone es un nuevo Sistema de Investigación e Información Policial (SIIPOL), tomando como referencia el Sistema Integral de Información Policial existente, de forma tal que las funciones actuales sean mantenidas, incrementando las posibilidades de uso de la información, y agregando nuevas funciones que el actual sistema no concibe, como es el caso de las Evidencias.

El sistema centralizará en varios módulos o subsistemas la automatización de los procesos que se llevan a cabo en las dependencias que pertenecen al CICPC. La organización y cantidad de subsistemas estará en la medida de las necesidades, por lo que no deben coincidir exactamente con los que existen actualmente en el SIIPOL; en principio los subsistema deberán organizarse por los procesos, y no por áreas de trabajo, es decir que por ejemplo: existirá un único subsistema para la toma de la denuncia, independientemente de que existan varias áreas que lleven a cabo este proceso con sus correspondientes particularidades; y un único subsistema para la sustanciación del expediente investigativo independientemente del tipo de delito investigado y el área que lleve a cabo la investigación.

1.4.1. SUBMÓDULO DE EVIDENCIAS. REQUISITOS

El módulo Registro y Control está compuesto por un total de 55 Casos de Uso distribuidos en los 6 sub-módulos que lo integran de la siguiente forma:

- ✓ 6 Casos de Uso pertenecientes al Submódulo de Auditorias.

- ✓ 16 Casos de Uso pertenecientes al Submódulo de Dotación de Equipos Policiales.
- ✓ 6 Casos de Uso pertenecientes al Submódulo de Evidencias.
- ✓ 8 Casos de Uso pertenecientes al Submódulo de Acceso.
- ✓ 5 Casos de Uso pertenecientes al Submódulo de Seguridad Bancaria.
- ✓ 14 Casos de Uso pertenecientes al Submódulo de Sustancias Químicas.

De los 6 Casos de Usos referentes al Submódulo de Evidencias se extrajeron los siguientes requerimientos como resultado de la aplicación de la ingeniería de requerimientos a los procesos que se desarrollan dentro del departamento.

Requerimientos Funcionales

RF 1. Iniciar cadena de custodia de una Evidencia.
- 1.1. Mostrar los campos que identifican al Elemento a convertir en Evidencia.
- 1.2. Incluir funcionarios presentes en el sitio del Suceso.
- 1.3. Incluir Imágenes relacionadas a la Evidencia.
- 1.4. Incluir los datos específicos de la Evidencia.

RF 2. Consultar Evidencias.

RF 3. Mostrar los datos actualizados de la Cadena de custodia de una Evidencia.

RF 4. Realizar el cambio de cadena de custodia de una Evidencia.
- 4.1. Validar funcionario que entrega la evidencia
- 4.2. Validar Funcionario que recibe la evidencia.
- 4.3. Imprimir o exportar a formato PDF el Control de Cambio de Custodia.

RF 5. Finalizar la Cadena de Custodia de una Evidencia.
- 5.1. Especificar la causa de la finalización de la cadena de la custodia.
- 5.2. Validar los datos correspondientes al funcionario responsable de la finalización.
- 5.3. Imprimir /Exportar a formato PDF el Control de Finalización de Custodia de Evidencia.

RF 6. Mostrar la Cadena de Custodia de una Evidencia.

6.1. Permitir ver los datos Correspondientes al elemento clasificado como Evidencia.

6.2. Imprimir o exportar a formato PDF la Cadena de Custodia de una Evidencia.

RF 7. Listar datos de coincidencias de la consulta.

RF 8. Imprimir o exportar a formato PDF el listado de coincidencias.

El modelo de casos de uso que se presenta a continuación representa los Casos de Usos del submódulo Evidencias perteneciente al módulo Registro y Control.

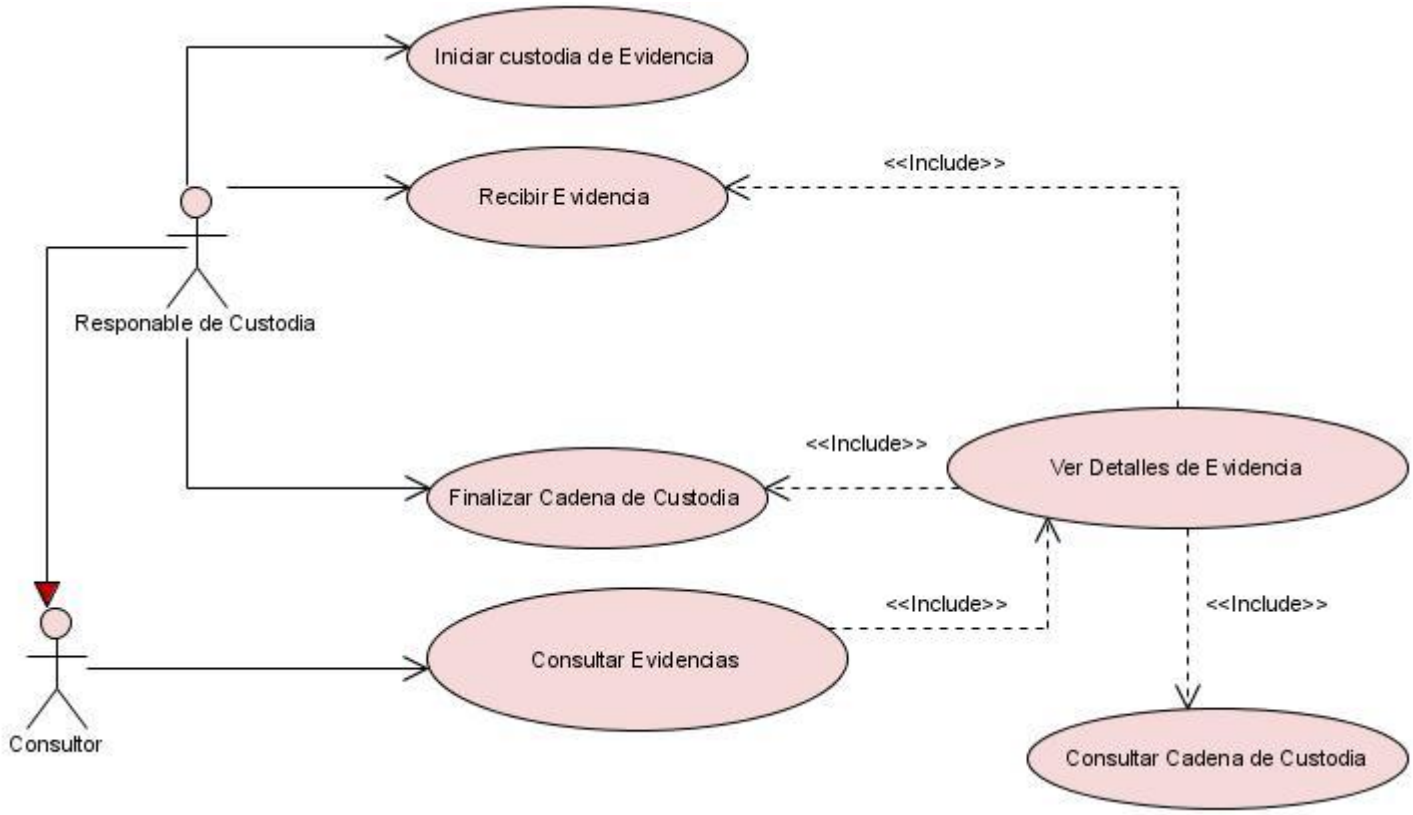


Imagen 1.1: *Modelo de Casos de Uso del Sistema, submódulo Evidencias*.

A continuación se presenta una breve síntesis de los Casos de Uso del submódulo de Evidencias que se tomaron como entrada para la realización del presente trabajo de diploma.

Caso de Uso	**Iniciar Custodia de Evidencia**
Propósito	Transformar un elemento determinado en una Evidencia e iniciar la custodia de la misma.
Actor(es)	Responsable de Custodia de Evidencia
Resumen	El caso de uso se inicia cuando el Responsable de Custodia luego de haber incluido un Objeto, Vehículo, Arma o Arma orgánica en el sistema, accede al sistema con el objetivo de iniciar la custodia de una Evidencia colectada, el sistema muestra y permite especificar los datos correspondientes a la custodia de la Evidencia incluyendo los funcionarios presentes en el sitio del suceso en el momento de la colección de la Evidencia y el conjunto de imágenes que se tomaron del lugar donde fue colectada.
Caso de Uso	**Ver Detalles de Evidencia**
Propósito	Mostrar los datos actualizados de la Cadena de Custodia de una Evidencia.
Actor(es)	Responsable de Custodia de Evidencia, consultor.
Resumen	El caso de uso se inicia cuando el actor accede a la opción que le

	permite ver los detalles de una evidencia. El sistema muestra los datos de la misma y permite consultar la cadena de custodia, Cambiar la Cadena de Custodia o finalizarla en dependencia de los permisos. Permite Imprimir/Exportar los datos mostrados.
Caso de Uso	**Consultar Cadena de Custodia de Evidencias**
Propósito	Mostrar de manera ordenada los datos de la cadena de custodia de una Evidencia.
Actor(es)	Consultor.
Resumen	El caso de uso se inicia cuando el actor en dependencia de los permisos que posea en el sistema, accede a la opción que le permite consultar la cadena de custodia de una Evidencia seleccionada, el sistema muestra todos los datos de los cambios de custodia asociados a la Evidencia en cuestión. Permite Imprimir/Exportar los datos mostrados y ver los datos del elemento clasificado como Evidencia.
Caso de Uso	**Consultar Evidencias**
Propósito	Buscar y listar de manera ordenada un resumen de los datos de las Evidencias coincidentes con uno o varios criterios de búsqueda.
Actor(es)	Consultor.
Resumen	El caso de uso se inicia cuando el actor accede a la opción que le permite consultar Evidencias, el sistema le permite especificar los

	criterios de búsqueda al actor, y lista todas las evidencias que corresponden a los datos incluidos por el actor. El sistema permite acceder a los detalles de las Evidencias listadas e Imprimir/Exportar los datos listados en la consulta.
Caso de Uso	**Finalizar Cadena de Custodia de Evidencia**
Propósito	Finalizar la cadena de custodia de una Evidencia determinada
Actor(es)	Responsable de Custodia de Evidencias.
Resumen	El caso de uso se inicia cuando el actor accede a la opción que le permite finalizar la custodia de una Evidencia determinada. El sistema le muestra y permite seleccionar las opciones que dan acceso a especificar la causa de la finalización de la custodia de la Evidencia, el actor selecciona la causa de finalización deseada y el sistema le permite especificar los datos correspondientes a la finalización de la custodia. El actor incluye los datos correspondientes y el sistema registra la finalización de custodia de la Evidencia mostrando un control de Finalización el cual se puede Imprimir/ Exportar con los datos mostrados.
Caso de Uso	**Recibir Evidencias**
Propósito	Efectuar el cambio de la cadena de custodia de una o varias Evidencias determinadas.
Actor(es)	Responsable de Custodia de Evidencias.

Resumen	El caso de uso se inicia cuando el actor accede a efectuar el cambio de cadena de custodia de una o varias evidencias seleccionadas. El sistema permite consultar y seleccionar las evidencias de las que es propietario y que desea cambiarles la custodia, e incluir los datos referentes al cambio de la cadena de custodia de dichas evidencias seleccionadas, registrándolo una vez que estos son especificados. Permite Imprimir/Exportar el Control de Cambio de Custodia que se genera con los datos del cambio de Custodia de las Evidencias, dando fin al Caso de Uso.

Tabla No. 1: *Descripciones resumidas de los Casos de Uso referentes al submódulo de Evidencias.*

1.5. METODOLOGÍA, LENGUAJES Y HERRAMIENTAS DE DESARROLLO

La industria del software a nivel mundial aplica a sus procesos de desarrollo diversas metodologías, métodos y procedimientos para lograr un software competente y de alta calidad, el cual debe obtener los mejores resultados en todas las esferas. El objetivo es afianzar la calidad del producto y artefactos generados en cada una de las fases o etapas por las que pasa el mismo hasta la entrega final alcanzándose mayor control y transparencia en las mismas.

En los últimos años los procesos de desarrollo de software han proliferado grandemente desarrollándose 2 corrientes principales denominadas **metodologías pesadas** y **métodos ligeros o metodologías ágiles,** la primera intenta lograr sus objetivos mediante la documentación generada y el orden mientras que la segunda por medio de la comunicación entre las personas que intervienen en el proceso de forma directa, trata de mejorar la calidad del proceso.

RATIONAL UNIFIED PROCESS (RUP)

Considerado dentro de las metodologías de desarrollo pesadas, está pensado para ser aplicado a cualquier proyecto de desarrollo por lo que es considerado uno de las más generales de las existentes en la actualidad. Utiliza el Lenguaje Unificado de Modelado (UML, Unified Modeling Language) como lenguaje gráfico para visualizar, especificar, construir y documentar la solución del sistema. RUP se basa principalmente en la experiencia de los líderes en otros proyectos realizados, así como la cohesión del equipo de desarrollo en su totalidad, optimizando la productividad del mismo. Se caracteriza por estar **dirigido por casos de uso** que son los que definen los requerimientos obtenidos con la captura de Requisitos y modelación del negocio a partir de las necesidades identificadas por el usuario, ser **iterativo e incremental** ya que en cada fase de desarrollo el equipo pasa por todos los flujos de trabajo, refinando el trabajo realizado en las iteraciones anteriores y añadiendo nuevos elementos y enriqueciendo el producto final cada vez más.

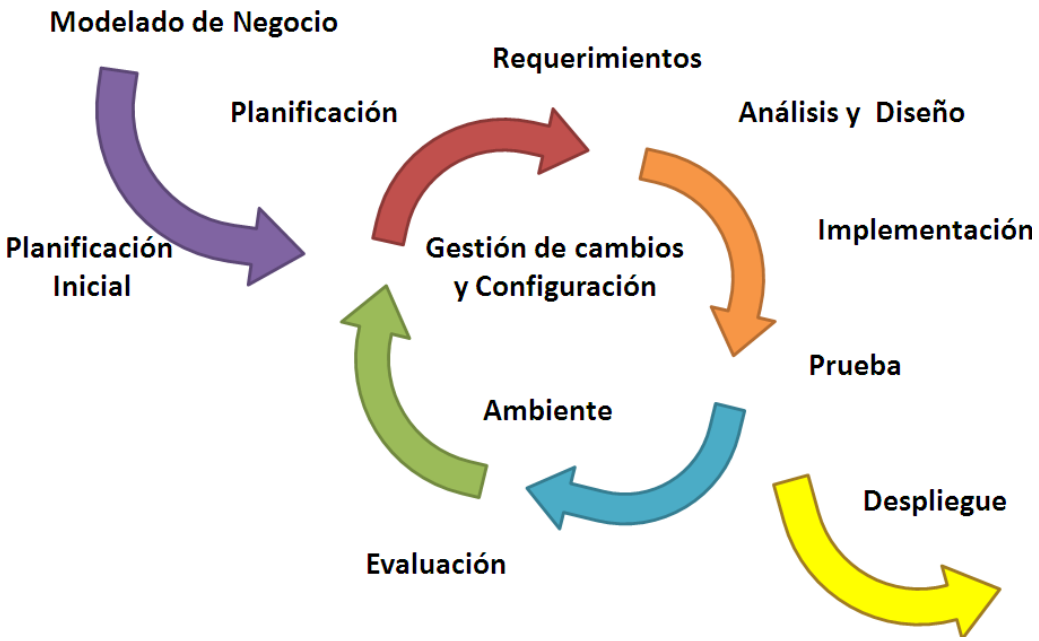


Imagen 1.2: *Representación del proceso ingenieril del software planteado por RUP.*

Otra característica que presenta es que está **centrado en la arquitectura**, en la cual se establecen las pautas iniciales y la vista de lo que será la aplicación final desarrollando los procesos más importantes del negocio o casos de uso arquitectónicamente significativos para establecer lo que será una línea de desarrollo.

RUP se define en 4 fases:

- ✓ **Inicio:** Se describe el negocio y se delimita el alcance del proyecto con la determinación 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. (2)
- ✓ **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.

Dentro de cada una de ellas el equipo de trabajo pasa por cada uno de los flujos de trabajo, inclusive en varias iteraciones para refinar cada vez más el producto final, variables en dependencia del tamaño y envergadura del proyecto.

Los ***flujos de Trabajo*** que este define son los siguientes (ver Imagen 1.3):

- ✓ **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, su ubicación en 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.

Flujos de soporte:

- ✓ **Administración del proyecto**: Involucra actividades con las que se busca construir 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. (2)

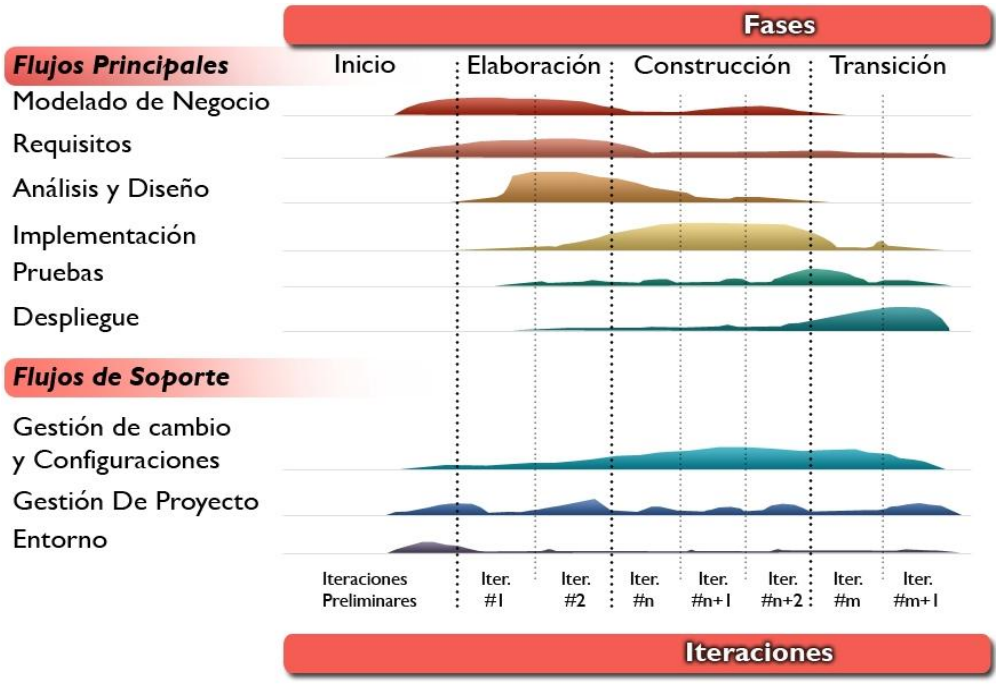


Imagen 1.3: *RUP en 2 Dimensiones.*

RUP está pensado para proyectos y equipos de desarrollo de amplia envergadura en duración y contenido. Además se documentan mediante UML, generando en cada flujo de trabajo e iteración los artefactos de cada una de ellas. Cada uno de los flujos de realiza de forma exhaustiva y eficiente para lograr una perspectiva de lo que realmente se desea hacer en el producto de software.

LENGUAJE DE MODELADO UML

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que permite graficar y especificar todas las fases de desarrollo de un software. Es un estándar del Grupo de

Gestión de Objeto (OMG)[2] y su utilización para el modelado es independiente del lenguaje de implementación, lo que permite graficar, documentar y construir los diseños necesarios y que éstos se puedan implementar en cualquier lenguaje que soporte las posibilidades de UML (como son casi todos los orientados a objeto), dando un apoyo significativo a los mismos. Es un método para el modelado que aporta algunas ventajas con su uso:

- Mayor rigor en la especificación.
- Permite realizar una verificación y validación del modelo realizado.
- Se pueden automatizar determinados procesos y permite generar código a partir de los modelos o aplicar la ingeniería inversa, permitiendo mantener el código y el modelo actualizados. (3)

"UML capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Un sistema se modela como una colección de objetos discretos que interactúan para realizar un trabajo que finalmente beneficia a un usuario externo. La estructura estática define los tipos de objetos importantes para un sistema y para su implementación, así como las relaciones entre ellos. El comportamiento dinámico define la historia de los objetos en el tiempo y la comunicación entre objetos para cumplir sus objetivos. El modelar un sistema desde varios puntos de vista, separados pero relacionados, permite entenderlo para diferentes propósitos." (1)

Permite adaptar elementos al modelado, asignándoles una semántica particular.

[2] **OMG™ (O**bject **M**anagement **G**roup) es una organización internacional de membrecía abierta y un consorcio de la industria de la Computación sin fines lucrativos. OMG Task Forces desarrolla la integración de estándares empresariales para una amplia gama de tecnologías.
Los estándares de modelado de OMG permitirán un poderoso diseño visual, ejecución y mantenimiento de software y procesos.

Selección de las herramientas de desarrollo. Tendencias en la actualidad para el desarrollo de aplicaciones.

Para plantear la solución se escogió realizar una aplicación Web debido a las siguientes características y ventajas que facilitan su uso y que darán al sistema las herramientas que este necesita para funcionar plenamente:

- Permite tener diversos clientes y estaciones de trabajo.
- Sólo se requiere de un navegador Web para acceder al sistema.
- Reducción de recursos en las estaciones de trabajo.
- Instalación y configuración individualizada, facilitando la disminución de los costos por concepto de mantenimiento.
- Acceso descentralizado geográficamente.
- Administración centralizada.
- Información centralizada, permitiendo la realización de copias de seguridad y resguardo de la información de una forma más fácil.
- Facilita el empleo de mecanismos de seguridad para el acceso a la misma.

La solución se basa en las necesidades actuales presentes en el CICPC, que necesita llevar el sistema a cada una de las delegaciones, subdelegaciones y despachos en cada uno de los estados, municipios y parroquias del área geográfica de Venezuela.

¿QUÉ ES UNA HERRAMIENTA DE CÓDIGO ABIERTO?

Código abierto (en inglés ***open source***) es el término con el que se conoce al software distribuido y desarrollado libremente. Se prefiere el uso del término ***Software Libre*** para referirse a programas que se ofrecen con total libertad de modificación, uso y distribución bajo la regla implícita de no modificar dichas libertades hacia el futuro.

La idea detrás del *open source* es simple: cuando los programadores en la red pueden leer, modificar y redistribuir el código fuente de un programa, éste evoluciona, se desarrolla y mejora consecuentemente. Los usuarios lo adaptan a sus necesidades, corrigen sus errores a una velocidad impresionante, mayor a la aplicada en el desarrollo de software convencional o cerrado, dando como resultado la producción de un mejor software. Están apoyados en las grandes Comunidades de Desarrollo dedicadas por completo a mejorar y difundir las facilidades de los programas y herramientas de código libre.

Hay que diferenciar los programas *Open source*, que dan a los usuarios la libertad de mejorarlos, de los programas que simplemente tienen el código fuente disponible, posiblemente con fuertes restricciones sobre el uso de dicho código fuente. Muchas personas consideran que cualquier software que tenga el código fuente disponible es open source, puesto que se puede manipular. Sin embargo, muchos de estos softwares no dan a sus usuarios la libertad de distribuir sus modificaciones, o sea, que restringe el uso comercial y los derechos de los usuarios sobre ellos.

PLATAFORMA DE DESARROLLO

Java Platform Enterprise Edition (J2EE) es un conjunto de tecnologías que reduce significativamente el coste y la complejidad de desarrollo, despliegue y gestión de ambiente,

centradas en un servidor de aplicaciones y una aplicación de cualquier tamaño a desarrollar. Sobre la base de la Plataforma Java, Standard Edition (Java SE), J2EE añade las capacidades que proporcionan una plataforma completa para el desarrollo sin contratiempos, que es a la vez estable, segura y rápida.

LENGUAJE DE PROGRAMACIÓN

Un **lenguaje de programación** es la forma de comunicación por medio de símbolos entre el usuario y la computadora. Está constituido por un grupo de reglas gramaticales, un grupo de símbolos utilizables, un grupo de términos con sentido único (monosémicos) y una regla principal que resume las demás. Actualmente existen lenguajes de programación de bajo, mediano y alto nivel. En el caso de los lenguajes de bajo nivel las instrucciones son muy simples y cercanas al funcionamiento de la maquina como lo son el código maquina y el ensamblador. Los de alto nivel están caracterizados por un alto nivel de abstracción y utilizan un lenguaje más cercano al natural, ejemplo de ellos son: C++, C#, Fortran y Java entre otros. Los lenguajes de programación de nivel medio, como su nombre lo dice, se localizan en un punto medio entre los dos anteriores. Dentro de estos lenguajes podría estar C, que puede acceder a los registros del sistema, trabajar con direcciones de memoria, todas ellas características de lenguajes de bajo nivel y a la vez realizar operaciones de alto nivel. Este último calificativo no es aceptado por todos los especialistas del tema.

Java es un lenguaje de programación de alto nivel y orientado a objetos desarrollado por *Sun Microsystems*[3] a principios de los años 90. Fue influenciado por Objective-C, C++, Smalltalk y Eiffel toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple que éstos a causa de que se han eliminado ciertas características de memoria, liberando a los

[3] Empresa de Informática de Sillycon Valley fabricante de semiconductores y software. Fue constituida en 1982 por el Alman Andres von Bechtolsheim y los norteamericanos Vinod y Koshla. Las siglas SUN derivan de "Stanford University Network"(Red de la Universidad de Stanford).Los softwares mas importantes que desarrolla la SUN actualmente son Java, Solaris y OpenOffice.

programadores de una familia entera de errores (la aritmética de punteros), se ha prescindido por completo los punteros y la recolección automática de basura (*garbage collector*) elimina la necesidad de liberación explícita de memoria (4), o sea, la administración de memoria solicitada dinámicamente de forma manual.

Dicha compañía describe el lenguaje *Java* como "*simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico*".

Java fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. El desarrollo de aplicaciones se apoya en un gran número de clases preexistentes. De algunas de ellas las ha podido hacer el propio usuario, otras pueden ser comerciales, pero siempre hay un número muy importante de clases que forman parte del propio lenguaje (el *API*[4] o *Application Programming Interface* de *Java*).

Al ser un lenguaje puro Orientado a Objetos le otorga gran reusabilidad y la independencia de la plataforma permite que los programas escritos en el lenguaje Java puedan ejecutarse igualmente en cualquier tipo de hardware, lo que significa ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como lo expresa el axioma de Java, "write once, run everywhere" (en español: "Escríbelo una vez, ejecútalo en cualquier parte").

En la medida en que es compilado el código fuente java (archivos de extensión .java) se transforma en una especie de código máquina denominado bytecodes[5] (archivos de extensión .class), semejantes a las instrucciones de ensamblador, estos se pueden ejecutar directamente

[4] **A**pplication **P**rogramming **I**nterface (**API)** o **Interfaz de Programación de Aplicaciones** 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.

[5] El **bytecode** es un código intermedio más abstracto que el código máquina. Habitualmente es tratado como un fichero binario que contiene un programa ejecutable similar a un módulo objeto, que es un fichero binario producido por el compilador cuyo contenido es el código objeto o código máquina. Recibe su nombre porque generalmente cada código de operación tiene una longitud de 1 byte, mientras que la longitud del código de las instrucciones varía. Cada instrucción tiene un código de operación entre 0 y 255 seguido de parámetros tales como los registros o las direcciones de memoria.

sobre cualquier máquina sin importar la arquitectura de ésta, porque los bytecodes son un formato intermedio indiferente a la arquitectura, diseñados para transportar el código eficientemente a múltiples plataformas hardware y software las cuales debe tener el intérprete y el sistema de ejecución en tiempo real (run-time) denominada maquina virtual de Java o JVM (java virtual machine).

Java incorpora en el propio lenguaje muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores (threads o hilos[6], ejecución remota, componentes, seguridad, acceso a bases de datos, etc.). Es por ello que muchos expertos opinan que Java es el lenguaje ideal para aprender la informática moderna, porque incorpora todos estos conceptos de un modo estándar, mucho más sencillo y claro que con las citadas extensiones de otros lenguajes. Esto es consecuencia de haber sido diseñado más recientemente y por un único equipo.

ENTORNO DE DESARROLLO INTEGRADO O IDE (INTEGRATED DEVELOPMENT ENVIRONMENT)

Un **IDE** es un entorno de programación creado como un programa de aplicación con un conjunto de herramientas para el programador: un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI. Estos pueden ser aplicaciones independientes o pueden formar parte de aplicaciones existentes, para un lenguaje de programación específico o multilenguaje. Proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación y puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto.

[6] Java soporta sincronización de múltiples hilos de ejecución (*multithreading o programación multihilo*) a nivel de lenguaje, especialmente útiles en la creación de aplicaciones de red distribuidas. De esa forma mientras un hilo se encarga de la comunicación, otro puede interactuar con el usuario mientras otro presenta una animación en pantalla y otro realiza cálculos.

Eclipse es un IDE de código abierto multilenguaje, independiente de la plataforma sobre la que se va a desarrollar. Es una aplicación de cliente enriquecido que emplea *plug-ins* para proporcionar toda su funcionalidad, en contraposición con los entornos monolíticos donde las funcionalidades están todas ya incluidas, sean necesarias o no para el usuario. 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. Eclipse es, en el fondo, solamente una armazón sobre la que se pueden montar otras herramientas de desarrollo. (5)

Comenzó como un proyecto de IBM Canadá. Fue desarrollado por OTI (Object Technology International) para suplantar a VisualAge también desarrollado por este último. En noviembre del 2001, se formó un consorcio para el desarrollo futuro de Eclipse como código abierto. En 2003, la fundación independiente de IBM fue creada.

Eclipse 3.0 (2003) seleccionó las especificaciones de la plataforma Open Services Gateway Initiative (OSGI)[7] como la arquitectura de tiempo de ejecución. En 2006 la fundación Eclipse coordinó sus 10 proyectos de código abierto, incluyendo la Plataforma 3.2, para que fueran liberados el mismo día. Esta liberación simultánea fue conocida como la liberación Callisto en Junio del 2007. La versión consecutiva a Callisto fue Europa, que corresponde a la versión 3.3 de Eclipse y la consecutiva a Europa es Ganymede, que se corresponde a la versión 3.4 de Eclipse, liberada 25 de Junio del 2008.

[7] **OSGI** refiere a **O**pen **S**ervices **G**ateway **I**nitiative, más precisamente el **OSGi14**. Fue creado en Marzo de 1999.Su objetivo es el definir las especificaciones abiertas de software que permita diseñar plataformas compatibles que puedan proporcionar múltiples servicios. Fue pensado principalmente para su aplicación en redes domésticas y por ende en la llamada Domótica o informatización del hogar. Tiene arquitectura definida pero se pensó para su compatibidad con Java y Universal Plug and Play (UPnP).

"Eclipse es una aplicación multiplataforma que dispone de: un editor de texto, resaltado de sintaxis, compilación en tiempo real, pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, wizards para la creación de proyectos, clases, test, componentes, refactorización. Asimismo, a través de "plug-ins" libremente disponibles es posible añadir: Control de versiones con Subversion, vía Subeclipse; Integración con Hibernate, vía Hibernate Tools, con Aptana, para desarrollar con java scripts." (1)

El IDE seleccionado cumple con la definición que da el proyecto Eclipse sobre él: "*una especie de herramienta universal - un IDE abierto y extensible para todo y nada en particular*".

Red Hat Developer Studio es un IDE de desarrollo basado en Eclipse, con herramientas de desarrollo pre-configuradas para plataformas *JBoss* y *Red Hat Enterprise*. Su primera versión fue puesta en manos de los desarrolladores de aplicaciones en código java a mediados del 2007 con su versión 1.0 Beta 1. Incluye JBoss Enterprise Middleware y Red Hat Enterprise Linux. El de desarrollador dispone de herramientas de desarrollo basadas en Eclipse integradas y un entorno disponible completamente en Open Source. La piedra angular del programa para desarrolladores de Red Hat, Developer Studio, está diseñada para maximizar la productividad del desarrollador sobre las soluciones de Red Hat y acelerar el cambio empresarial a una arquitectura Open Source. (6)

Soporta los frameworks de Exadel Studio Pro, RichFaces y Ajax4jsf, JBoss Seam, JSF y Facelets e Hibernate dentro de un potente entorno de desarrollo para aplicaciones SOA[8], AJAX y Java empresariales.

[8] **S**ervice **O**riented **A**rchitecture **(SOA)** o **Arquitectura Orientada a Servicios** define la utilización de servicios para dar soporte a los requisitos del negocio como concepto de arquitectura de software. Permite la creación de sistemas altamente escalables que reflejan el negocio de la organización, a su vez brinda una forma estándar de exposición e invocación de servicios, lo cual facilita la interacción entre diferentes sistemas propios o de terceros.

Developer Studio se caracteriza por:

- Modelo de programación unificado.
- Potentes capacidades AJAX.
- Utilidades Java Platform Enterprise Edition (EE)
- Simplifica el despliegue, la ejecución y la depuración de las aplicaciones J2EE, con la inclusión e integración del servidor de aplicaciones JBoss.
- Tiempo de ejecución integrado con las herramientas de desarrollo, que une el runtime con las herramientas a utilizar. Ello elimina la necesidad de que el desarrollador improvise estructuras y antes de empezar a desarrollar.

Presenta un grupo de prestaciones y es más fácil en cuestiones de configuración que otros como Exadel Studio que muchas de las prestaciones que ya vienen incluidas en Red Hat Developer Studio deben ser incluidas como plugins para que puedan ser usadas por Exadel.

Red Hat Developer Studio fue el IDE de desarrollo seleccionado para la implementación de la aplicación final en la que está incluido el módulo Registro y Control y el submódulo de Evidencias por su fácil utilización y configuración para una rápida implementación de la aplicación final.

HERRAMIENTAS CASE DE MODELADO CON UML

Las **herramientas CASE** (***C****omputer* ***A****ided* ***S****oftware* ***E****ngineering, Ingeniería de Software asistida por computadora)* son diversas aplicaciones destinadas a aumentar la productividad en el desarrollo del software reduciendo el coste de las mismas en términos de tiempo y dinero al mismo tiempo que evitan la aparición de errores permitiendo obtener una idea clara de que es lo que se va a hacer antes de comenzar a programar. Estas herramientas tienen vital importancia en múltiples aspectos del ciclo de vida del desarrollo del software en tareas como realizar un

diseño del proyecto, cálculos de costes, implementación de parte del código con el diseño dado, compilación automática, documentación o detección de errores, entre otras.

La realización de un nuevo software requiere que las tareas sean organizadas y completadas en forma correcta y eficiente. Estas herramientas fueron desarrolladas para automatizar esos procesos y facilitar las tareas de coordinación de los eventos que necesitan ser mejorados.

Existe una gran cantidad de herramientas comerciales para el modelado con UML y estas han venido incrementando drásticamente.

Las herramientas de modelado con UML han venido desarrollándose con el transcurso del tiempo cuya velocidad viene marcada con la creciente complejidad de las aplicaciones a realizar, estas ofrecen cada vez más beneficios para los distintos roles de desarrollo.

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generación código a partir de diagramas y de documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Tiene una amplia gama de características y/o funcionalidades entre las que resaltan las siguientes:

- Soporte de UML versión 2.1.
- Diagramas de Procesos de Negocio - Proceso, Decisión, Actor de negocio, Documento.
- Modelado colaborativo con CVS[9] y Subversion. Interoperabilidad con modelos UML2 (metamodelos UML 2.x para plataforma Eclipse) a través de XMI.

[9] **Concurrent Versions System** (**CVS**) o Sistema de Control de Versiones es una aplicación informática que implementa un sistema de control de versiones: mantiene el registro de todo el trabajo y los cambios en los ficheros, principalmente de código fuente; que forman un proyecto de software y permite que distintos desarrolladores desarrollen aún si se encuentran a grandes distancias. Este se ha popularizado en el mundo del software libre y sus desarrolladores difunden el sistema bajo la licencia GPL.

- Ingeniería inversa - Código a modelo, código a diagrama.
- Ingeniería inversa Java, C++, Esquemas XML, XML, .NET exe/dll, CORBA IDL.
- Generación de código - Modelo a código, diagrama a código.
- Editor de Detalles de Casos de Uso - Entorno todo-en-uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Diagramas EJB[10] - Visualización de sistemas EJB.
- Generación de código y despliegue de EJB´s - Generación de beans para el desarrollo y despliegue de aplicaciones.
- Diagramas de flujo de datos.
- Soporte ORM[11]: Generación de objetos Java desde la base de datos.
- Generación de bases de datos: Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Ingeniería inversa de bases de datos - Desde Sistemas Gestores de Bases de Datos (DBMS) existentes a diagramas de Entidad-Relación.
- Generador de informes para generación de documentación.
- Distribución automática de diagramas - Reorganización de las figuras y conectores de los diagramas UML.
- Importación y exportación de ficheros XML.

[10] Los **Enterprise JavaBeans** (también conocidos por sus siglas **EJB**) son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE de Sun Microsystems .Su especificación detalla cómo los servidores de aplicaciones proveen los EJB, que no son más que objetos desde el lado del servidor.

[11] **ORM (object/relational mapping) Mapeo Objeto-Relacional:** El mapeo de objetos relacionales es la persistencia automática y transparente de objetos en una aplicación Java a tablas en una base de datos relacional, usando metadatos que describen el mapeo entre objetos y la base de datos, ORM, en esencia, trabaja transformando datos de una representación a la otra.

- Integración con Visio - Dibujo de diagramas UML con plantillas (stencils) de MS Visio.
- Editor de figuras.
- Entorno de Desarrollo Inteligente (SDE[12]) para Eclipse.
- SDE para NetBeans.
- SDE para Sun ONE.
- SDE para Oracle JDeveloper.
- SDE para JBuilder.
- SDE para IntelliJ IDEA.
- SDE para WebLogic Workshop. (7)

Esta herramienta es de fácil aprendizaje y muy intuitiva para el desarrollo rápido de herramientas así como para generar código para java.

[12]**Smart Development Environment (SDE) o Entorno de Desarrollo Inteligente de Visual Paradigm** ofrece una integración sin fisuras de esta herramienta de modelado UML con los principales IDEs (Visual Studio, Eclipse, Borland JBuilder, NetBeans, Oracle JDeveloper y otros). Este se añade al IDE para proporcionar un único entorno de desarrollo y modelado. Le ayuda a construir aplicaciones de calidad más rápido, mejor y más barato. Puede dibujar diagramas UML en Eclipse, generar código Java, ingeniería inversa de código Java a diagramas de clase entre otras tantas posibilidades.

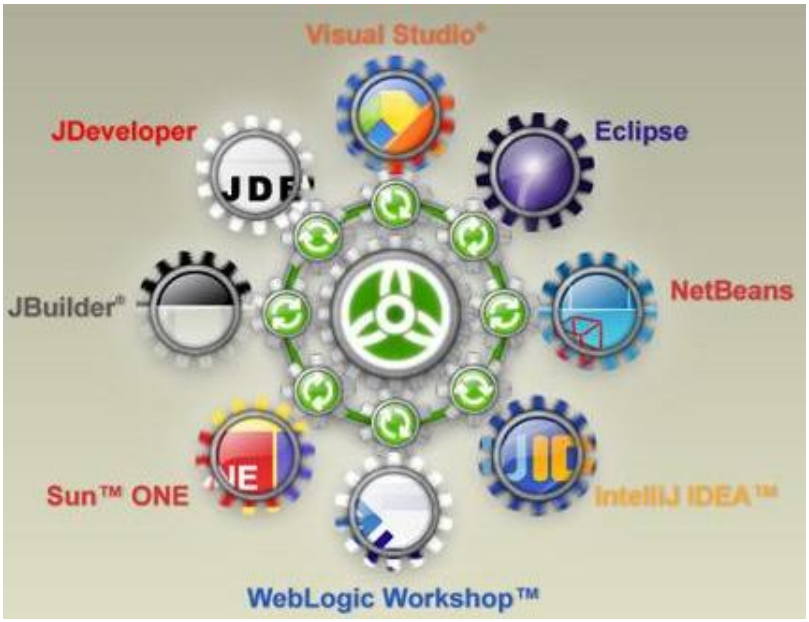


Imagen 1. 4: *Integración del SDE de Visual Paradigm con un grupo de IDEs.*

FRAMEWORKS UTILIZADOS SOPORTADOS POR JAVA

Un **framework** (la traducción aproximada es marco de trabajo) es una estructura de soporte compuesta por componentes genéricos, personalizables, intercambiables y configurables para la organización de un proyecto de software. Son muy utilizados y aceptados, producto de su capacidad de promover la disminución del tiempo de desarrollo del software, la reutilización del código del diseño, el código fuente y el uso de patrones como buenas prácticas de programación. Están relacionados directamente con un dominio específico, o sea, con una familia de problemas relacionados, lo que hace que estos se especialicen en áreas o capas especificas del sistema: presentación, lógica de negocio o acceso a datos. Dicho concepto se emplea en casi todos los ámbitos de desarrollo de software y se utilizan en el desarrollo de aplicaciones múltiples como juegos, aplicaciones médicas, Web, etc.

Capa de presentación

Java Server Faces Framework ó Java Server Faces (**JSF**) es un estándar creado por la SUN y su framework oficial para aplicaciones web basadas en Java, facilitando la construcción de interfaces de usuario (UI) para aplicaciones del lado del servidor con tecnología Java siguiendo el Patrón Modelo- Vista-Controlador (MVC)[13]. Diseñado para ser flexible, posee un modelo de componentes orientado a objetos, conversión de tipos de datos, separación de responsabilidades, desarrolladores de componentes, desarrolladores de lógica de aplicación, montadores de páginas, un poderoso sistema de navegación declarativa, uso de simples clases java como controladores, fácil incorporación de potencialidades AJAX[14]. Posee, además, un conjunto prefabricado y reutilizable de componentes de interfaz de usuario que permiten creación rápida de páginas Web, los que se conectan, mediante Beanes Manejados o de Respaldo, a la lógica de la aplicación y la escritura de eventos generados en el cliente en controladores de eventos en el lado del servidor; con un modelo de programación orientado a eventos. (8)

Esta tecnología incluye:

- Un conjunto de APIs que representan componentes UI y el manejo de su estado, controlando eventos y validaciones de entrada de datos, definición de navegación entre páginas, soporte para internacionalización y accesibilidad.
- Un conjunto de librería de tags personalizables para las JavaServer Pages (JSP) como interfaz entre JSF y la pagina JSP.

[13] El **Patrón Modelo- Vista-Controlador (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 tipos de componentes distintos, desarrollando un diseño que desacople la vista del modelo para mejorar la reusabilidad. De esta forma las modificaciones en las vistas modificarán en menor cuantía la lógica de negocio y el acceso a datos.
[14] del término en inglés *Asincronous Java Script and XML*

JSF aprovecha la tecnología existente, la interfaz de usuario estándar y el nivel conceptual de la Web sin limitar a los desarrolladores a un lenguaje de marcado, protocolo, o dispositivo cliente particular. Las clases de componentes de la interfaz de usuario incluidas en la tecnología JSF encapsulan la funcionalidad del componente, no la presentación específica en el cliente, lo que permite a los componentes UI ser *rendereados* en diversos dispositivos cliente.

Mediante la combinación de las funcionalidades de los componentes UI con los *renders* personalizados y definiendo los atributos de un determinado componente UI, los desarrolladores pueden construir etiquetas personalizadas a un dispositivo cliente en particular.

Como conveniencia, la tecnología JavaServer Faces proporciona un tag de bibliotecas para ser *renderizadas* por los clientes HTML permitiendo a los desarrolladores usar dicha tecnología en las aplicaciones a desarrollar.

Su arquitectura define claramente mediante el patrón MVC la separación de la capa de presentación de la capa de lógica de negocio facilitando la conexión de cada capa de la aplicación, ello permite que cada desarrollador se preocupe por la parte que le corresponde y luego integrar todas las partes en un todo, como un rompecabezas.

Posee un grupo de bondades factibles para el desarrollador como son:

- Modelo de trabajo basado en etiquetado y XML como componentes de Interfaz de Usuario (UI).
- Arquitectura basada en el patrón MVC.
- Asocia (de forma modular) cada componente gráfico con los datos (beanes de respaldo o backing beans).
- Incluye la capa de control, definida de forma declarativa en archivos XML. Lo que implica control de eventos y errores.
- Validación en cliente y en servidor.
- Control de mensajes y roles.

- Es muy flexible, con capacidad de crear componentes personalizados y *renders* para señalizarlos.
- Es más fácil de manejar que otros frameworks como Struts.

Al igual que Struts[15], 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 de sus deficiencias. Además, algunos IDEs como Eclipse y Netbeans brindan facilidades que soportan el trabajo con el mismo. A diferencia de Struts, JSF es parte de Java Enterprise Edition, todos los servidores de aplicaciones, por tanto, incluyen JSF.

Ajax4JSF es una librería Open Source que se integra totalmente en la arquitectura de JSF y extiende la funcionalidad de sus etiquetas dotándolas con tecnología AJAX[16] de forma limpia y sin añadir código Java Script. Mediante este framework podemos variar el ciclo de vida de una petición JSF, recargar determinados componentes de la página sin necesidad de recargarla por completo, realizar peticiones al servidor automáticas, control de cualquier evento de usuario, etc. En definitiva Ajax4jsf permite dotar a nuestra aplicación JSF de contenido mucho más profesional con muy poco esfuerzo.

Ajax4JSF es una extensión del estándar JSF que se integra a este con gran facilidad, evita escribir código *Java Script* al brindar una amplia gama de componentes, permite agregar capacidades AJAX 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.*

[15] ***Struts*** es un *framework* de la capa de presentación que implementa el patrón MVC en Java 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.

Capa de lógica de negocio

Spring es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java. Spring no obliga a usar un modelo de programación en particular; se ha popularizado en la comunidad de programadores en Java al considerársele una alternativa y sustituta del modelo de Enterprise JavaBean (EJB). Por su diseño el framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria.

Spring incluye una amplia gama de funcionalidades que ayudan al desarrollador a crear aplicaciones de una forma segura y con un reducido margen para la ocurrencia de errores, además de que permite una fácil integración con otros frameworks.

Algunas características son las siguientes:

- Es considerado el framework ligero más completo que facilita una configuración y escritura de objetos muy centralizada y automatizada. No es muy agresivo a la arquitectura del proyecto, permite el montaje de complejos sistemas de componentes (POJOs) de forma coherente y transparente. El framework brinda agilidad y mejora grandemente las capacidades de prueba de la aplicación. Es compatible con ambientes de desarrollo J2SE y J2EE.

- Contiene una capa común de abstracción para el manejo de las transacciones que permite conectar los administradores de las mismas, haciendo que sea más fácil para demarcar las operaciones sin tratar con bajo nivel de cuestiones. Estrategias genéricas para **JTA** [17]y conexiones simples de JDBC están incluidas.

[17] JTA (del inglés **Java Transaction API** - **API para transacciones en Java**) forma parte de **Java Enterprise Edition APIs**, establece una serie de Interfaces java entre el manejador de transacciones y las partes involucradas en el sistema de transacciones distribuidas: el servidor de aplicaciones, el manejador de recursos y las aplicaciones transaccionales. Fue desarrollado por la Sun Microsystems y es una especificación construida bajo el Proceso de comunidad Java JSR 907. (20)

- Contiene una capa de abstracción de JDBC que ofrece una significativa jerarquía de excepciones, lo cual simplifica el tratamiento de errores y reduce enormemente la cantidad de código a escribir.
- Brinda soporte a varios frameworks de presentación, entre ellos Java Server Faces (JSF)
- Acomoda varias tecnologías de vista como por ejemplo JSP, Velocity, Tiles, iText, POI, entre otras.
- Brinda soporte a la programación orientada a aspectos (AOP) y complementa la Programación Orientada a Objetos (POO).

En la Imagen a continuación se muestra la arquitectura por capas de Spring.

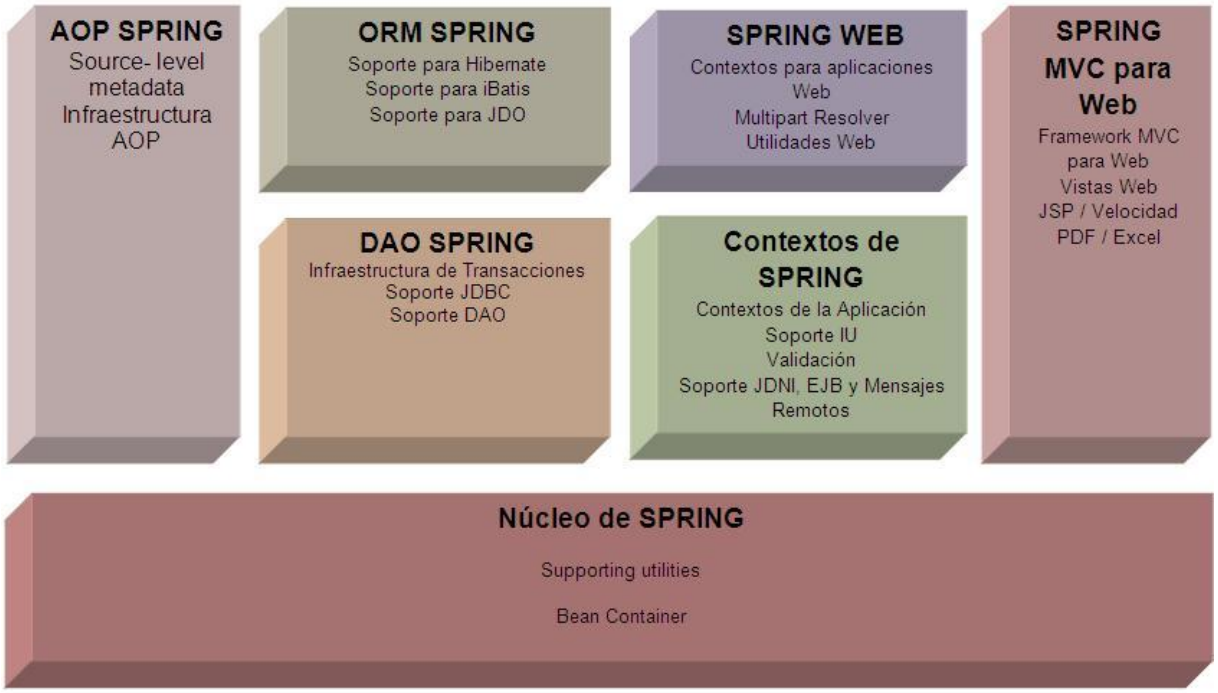


Imagen 1. 5: *Arquitectura por capas de Spring.*

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

Seguridad

El **Framework Acegi Security System** cubre todas las facetas de la seguridad de una aplicación empresarial, concebido para integrarse con Spring, fácil de extender, no necesita de contenedores para su uso y es de código abierto.

La versión estable ACEGI 1.0.0 fue liberada en mayo 2006, desde entonces ha sido utilizada en gran cantidad de proyectos, ha tenido más de 70000 descargas y cientos de contribuciones de la comunidad de programadores y de software libre para mejorar sus prestaciones y mejorarla cada vez más.

Dentro de las principales funcionalidades y aplicaciones del framework de seguridad Acegi se tienen:

- Está bien documentado.
- Brinda rápidos resultados.
- Reutiliza la experiencia en Spring: utiliza los contextos de aplicación de Spring para todas las configuraciones lo cual centraliza y facilita el trabajo de los desarrolladores con conocimientos de Spring.
- Asegura instancias de objetos de dominio: en muchas aplicaciones es conveniente definir listas de control de acceso (ACLs) para cada instancia de un objeto de dominio, por lo que se provee de un amplio paquete de ACL.

- El sistema de seguridad puede operar dentro de una misma aplicación web utilizando los filtros de siempre. No hay necesidad de hacer cambios especiales o desplegar las bibliotecas al Servlet o contenedor EJB.
- Completo (pero opcional) contenedor para la integración: La recogida de credenciales y la autorización de su capacidad de Servlet o contenedor EJB puede ser completamente utilizada, incluida a través de "contenedor de adaptadores". Actualmente soporta Catalina (Tomcat), Jetty, JBoss y Resina, con nuevos contenedores fácilmente añadibles.
- No es necesario agregar código de seguridad en los objetos de negocio de Spring.
- No solamente protege los métodos de ser invocados sino que también trata con los objetos retornados.
- Asegura las peticiones HTTP.

Compatibilidad con los métodos de seguridad de HttpServletRequest. (9)

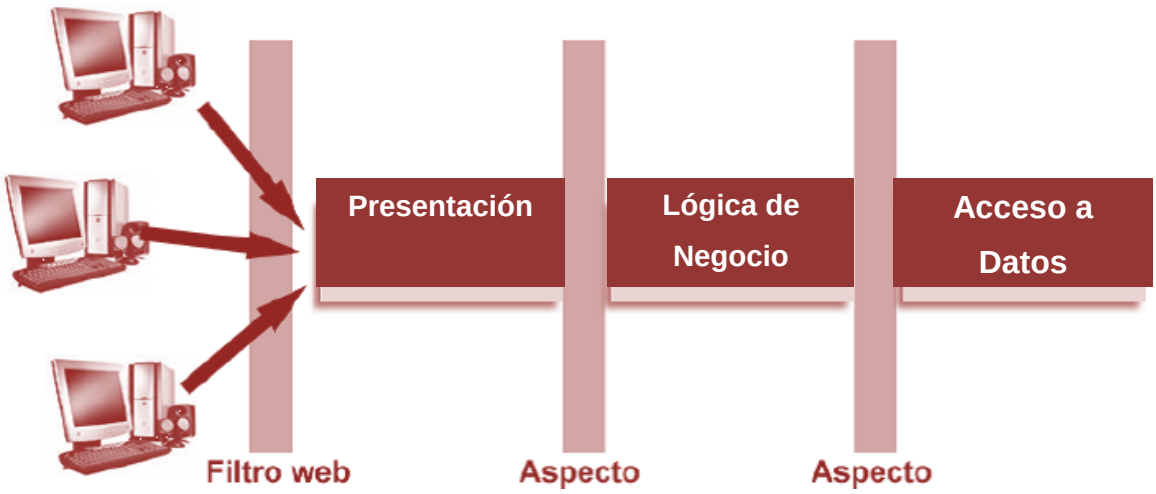


Imagen 1. 6: *Manejo de la Seguridad mediante Acegi***.**

No se especifica mucho más sobre este framework debido a que existen muy pocos puntos de contacto entre el trabajo del desarrollador y el aseguramiento de la aplicación, lo que no es su ni responsabilidad, por lo que no es objetivo en el desarrollo de esta investigación.

Capa de Acceso a Datos

La forma en que Java maneja las conexiones a una Base de datos es abriendo una conexión mediante la Interfaz de Programación de Aplicaciones (API) de JDBC, y la consulta SQL (Lenguaje de Consultas Estructurado en su acrónimo inglés); debido a que son objetos que se van a hacer persistentes en una base de datos o en ficheros a través del tiempo lo que provoca que este proceso sea altamente repetitivo y por lo tanto muy propenso a errores.

Una aplicación con su modelo de dominio orientado a objetos no interactúa directamente con las columnas y tablas; por lo que se hizo uso de una forma muy elegante y novedosa de realizar las acciones sobre la base de datos a través de entidades encargadas de esta función, haciendo uso del Mapeo Relacional de Objetos (ORM).

Hibernate es un Framework que se le integra al Eclipse y utiliza Mapeo Objeto-Relacional (ORM) para la relación entre el modelo objetual y el modelo relacional. Permite una fácil integración con cualquier otro Framework de Java y para su uso no requiere de muchas reglas específicas o patrones, lo que evita que se hagan grandes cambios en la arquitectura de la aplicación. Resuelve un gran problema de portabilidad dado el hecho que es muy configurable mediante el uso de XML, permitiendo configurar, entre otras cosas el dialecto del gestor de base de datos con el que se trabaja. Automatiza de forma eficiente el trabajo con operaciones altamente repetitivas como crear, actualizar, consultar y eliminar. Permite trabajar con objetos persistentes haciendo uso de polimorfismo y herencias y relaciones y la mayoría de los tipos de datos que soporta Java. Es una solución no intrusiva, que no obliga a seguir determinadas reglas o patrones de diseño cuando se desarrolla la capa de lógica de negocio o las clases del dominio. Las clases

persistentes no requieren implementar la interfaz serializable, aunque sí deben cumplir con la característica de tener un constructor sin parámetros. (1)

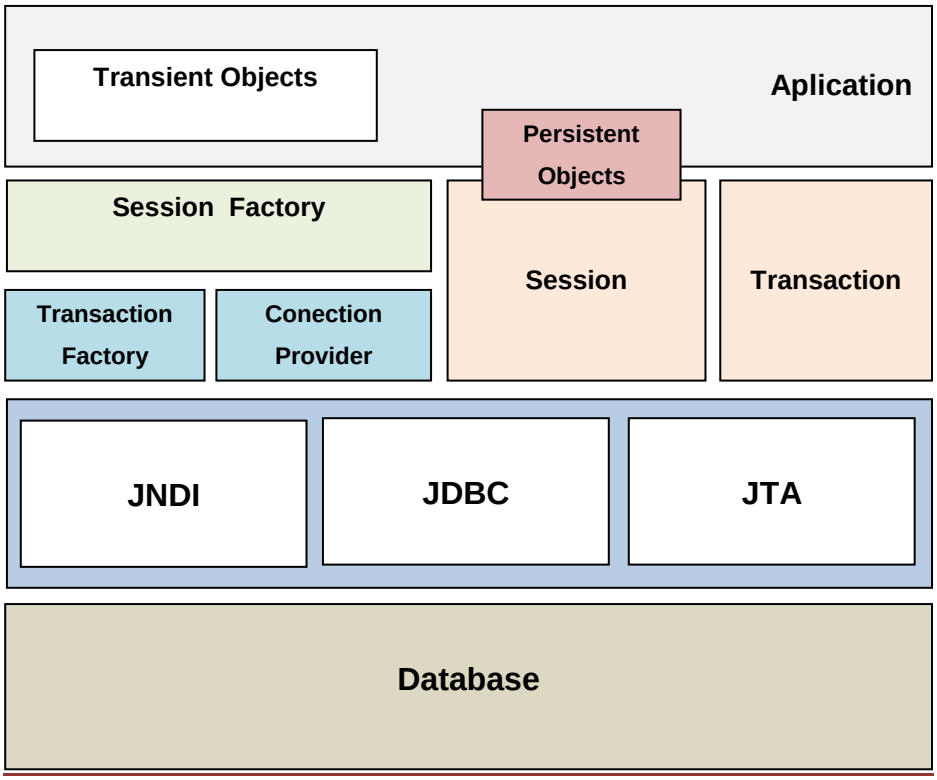


Imagen 1. 7: *Arquitectura por capas del Framework Hibernate*

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, por norma general, sólo se use en empresas muy grandes y/o multinacionales.

Para desarrollar en *Oracle* se utiliza PL/SQL, un lenguaje de 5ta Generación, muy potente para tratar y gestionar la base de datos, y muy versátil.

1.6. ARQUITECTURA TÉCNICA

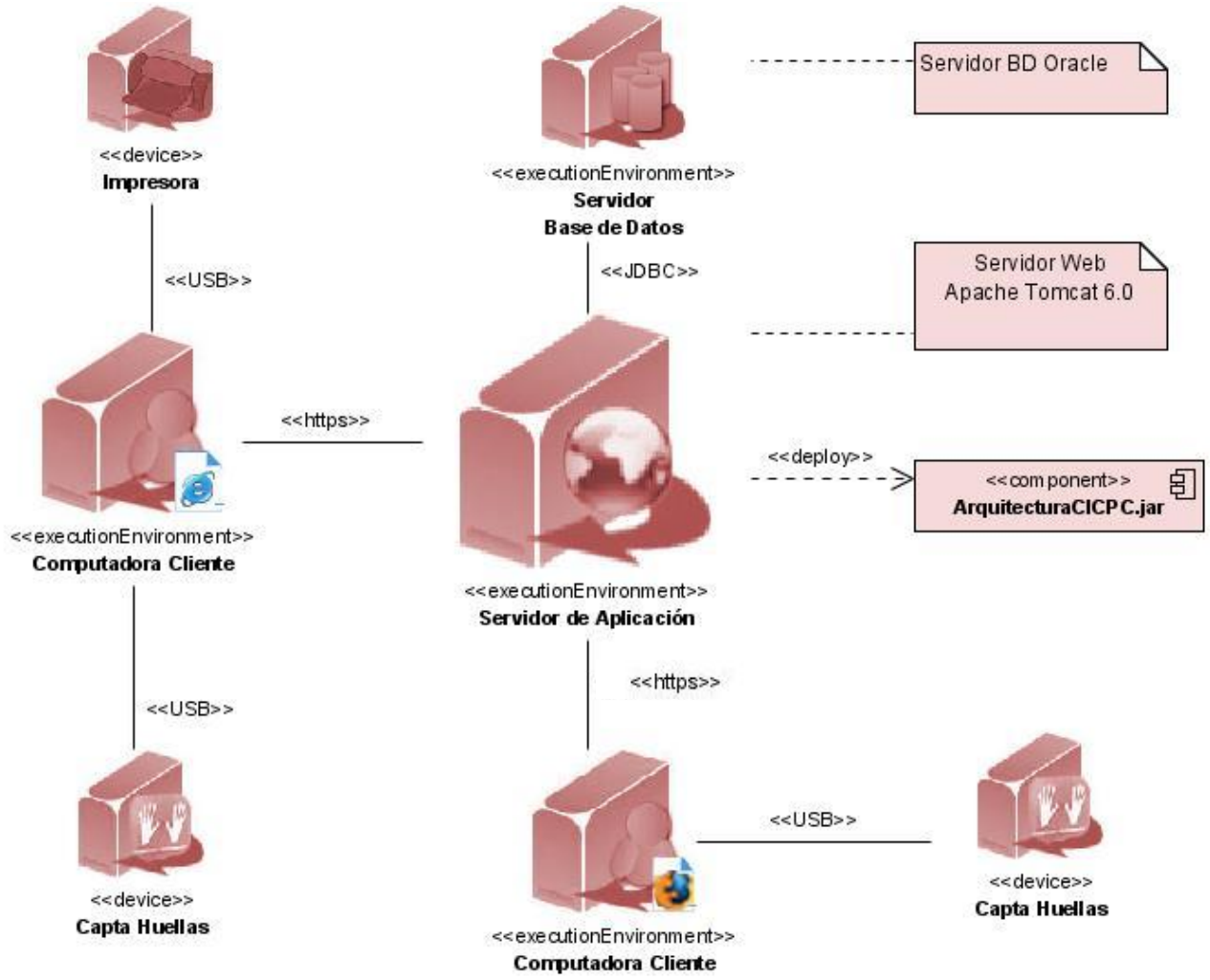


Imagen 1. 8: *Vista de despliegue de la propuesta de solución final.*

Con la utilización de el lenguaje java para el desarrollo, los frameworks definidos y Visual Paradigm como herramienta case para el desarrollo de los artefactos que regirán el ciclo de desarrollo guiado por RUP; se elaborará la propuesta de solución, especificada a alto nivel mediante una aplicación con una arquitectura cliente- servidor para una aplicación Web con comunicación mediante protocolo HTTPs, con un servidor de aplicaciones donde compartirán recursos la base de datos Oracle y el servidor web Apache Tomcat. En dicho servidor se publicará la aplicación para su despliegue final (Imagen 1.8). No solo se utiliza el patrón MVC sino también del patrón N capas de 3 niveles que se utilizan en la aplicación para la organización del código fuente; permitiendo escalabilidad, modularidad y facilidades de desarrollo y mantenimiento de la misma.

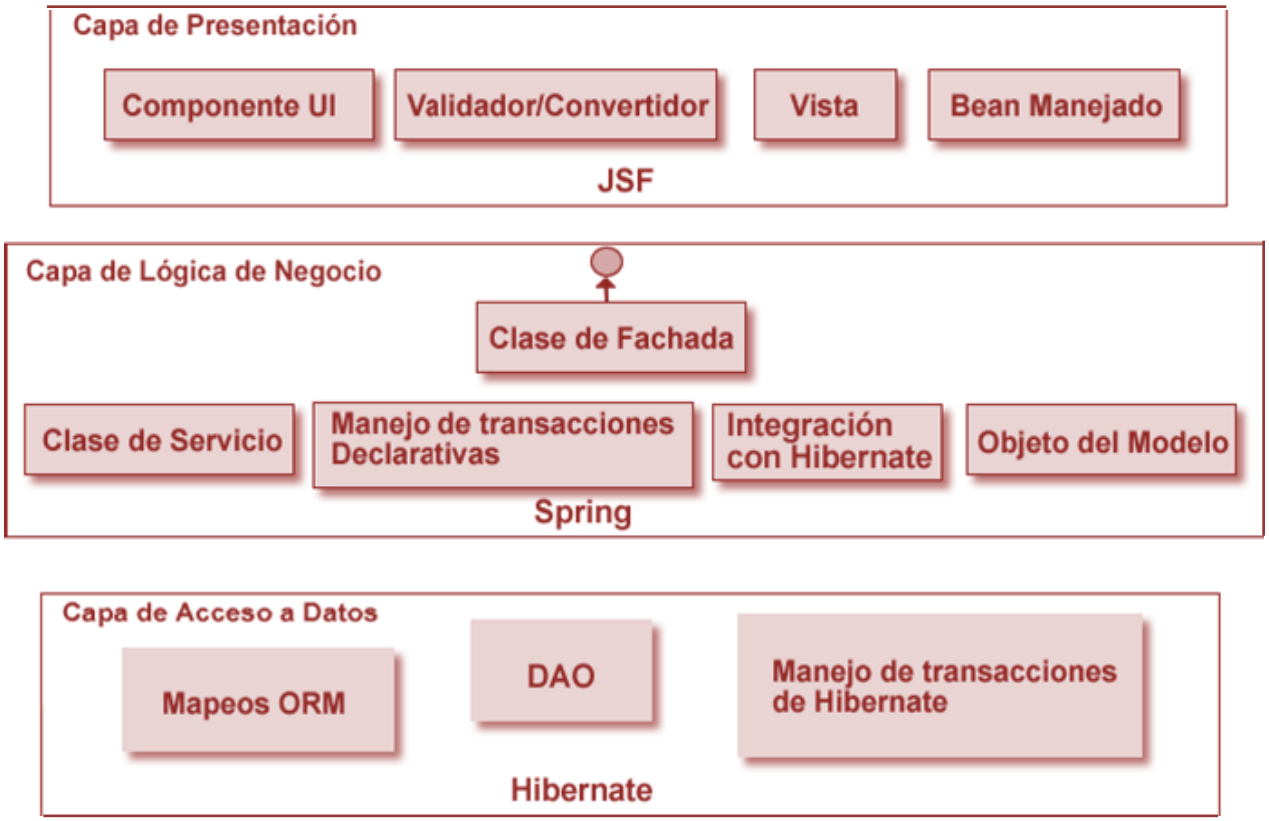


Imagen 1. 9: *Modelo n-capas de 3 niveles de la solución.*

1.7. CONCLUSIONES

En este capítulo se profundizó en el tema del Sistema de Gestión e Información Policial, se realizó un estudio de un conjunto de aplicaciones similares que se consideraron de importancia para el diseño de la solución de software. Se fundamentó también la metodología, tecnologías y estándares utilizados para la elaboración de la propuesta de solución del producto de software. Con estos elementos definidos, se puede comenzar con el Flujo de Trabajo Análisis y Diseño, específicamente en el Análisis referente al módulo Registro y Control del SIIPOL.

Con el uso de las tecnologías y herramientas de punta seleccionadas para el desarrollo del producto final, la decisión de realizar una aplicación Web soportada por el lenguaje de programación Java y usando J2EE como plataforma , con la utilización de Eclipse y Visual Paradigm como entorno de trabajo y los frameworks para cada una de las capas definidas por la arquitectura, se facilitará el desarrollo rápido y eficiente de un software de calidad, competente, guiado y orientado por RUP como metodología para el desarrollo y que cumplirá con los requisitos definidos con el cliente.

CAPÍTULO 2. ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE LA PROPUESTA DE SOLUCIÓN.

2.1. INTRODUCCIÓN

En el presente capítulo se tratan temas relacionados al análisis, diseño e implementación de la propuesta de solución, para lo cual se comenzará con una breve referencia de los principales argumentos relacionados con los flujos de trabajo mencionados y los artefactos que se generan producto de la Ingeniería de Requerimientos aplicada al submódulo. Además se explicarán elementos propios del sistema, que se adaptaron a la arquitectura definida por el equipo de Arquitectos principales del proyecto, algunos de los patrones de diseño utilizados así como los estándares de codificación que se tuvieron en cuenta para la implementación de la propuesta final.

El Flujo de Trabajo Análisis y Diseño.

El Flujo de Trabajo de Análisis y Diseño tiene como objetivos primordiales para el desarrollo:

- La transformación de los requerimientos en un diseño de cómo va a ser implementado el sistema
- Evolucionar hacia una arquitectura del software robusta.
- Adaptar el diseño para que éste coincida con el ambiente de implementación, diseñando el sistema con un enfoque hacia el rendimiento.

En la fase de Inicio, la disciplina de Análisis y Diseño se preocupa por establecer si la visión del sistema es factible, y en determinar las tecnologías potenciales para la solución de software (dentro de la actividad ***Realizar la Síntesis de la Arquitectura***). Si se considera que pocos

riesgos se asocian al desarrollo (porque el dominio se entiende muy bien, el sistema no es novedoso o cualquier otra razón parecida) entonces éste aspecto se omite del flujo de trabajo.

En la parte inicial de la fase de Elaboración se enfoca el esfuerzo en crear una arquitectura inicial del sistema (en la actividad ***Definir la Arquitectura Candidata***) la cual provea de un punto inicial para todo el trabajo de análisis. Si la arquitectura ya existe (porque fue creada en iteraciones anteriores o en proyectos anteriores) el trabajo se enfoca en cambios para refinar la arquitectura (actividad ***Refinar la Arquitectura***) y en analizar el comportamiento y crear un conjunto inicial de elementos los cuales proveen un comportamiento apropiado (en la actividad ***Análisis de Comportamiento***).

Después de que los elementos iniciales son identificados, se refinan en iteraciones futuras. La actividad ***Diseñar Componentes*** produce un conjunto de componentes los cuales proveen un comportamiento adecuado para satisfacer los requerimientos del sistema. Si el sistema incluye una base de datos, entonces la actividad de ***Diseñar la Base de Datos*** se ejecuta en paralelo. El resultado es un conjunto inicial de componentes los cuales en un futuro son refinados dentro de la Implementación.

El análisis en pocas palabras

El objetivo de este flujo de trabajo es traducir los requisitos a una especificación que describe cómo implementar el sistema. El análisis consiste en obtener una visión del sistema que se preocupa de ver QUÉ hace, de modo que sólo se interesa por los requisitos funcionales.

El modelo de análisis ayuda a refinar los requisitos y permite razonar sobre los aspectos internos del sistema. Este proporciona una estructura centrada en el mantenimiento en aspectos tales como la flexibilidad ante los cambios y la reutilización, también se utiliza como entrada en las actividades de diseño e implementación. El modelo de análisis se considera una primera aproximación al modelo de diseño, aunque es un modelo por sí mismo. Mediante su realización se puede utilizar el resultado para planificar el trabajo del diseño y le implementación.

El análisis proporciona una visión general de del sistema que puede ser más difícil de obtener mediante el estudio de los resultados del diseño e implementación, debido a que contienen demasiados detalles. Este provee una visión general, proporciona una vista conceptual, precisa y unificadora del sistema a implementar de forma genérica, o sea, aplicable a varios diseños. Este puede o no mantenerse durante el ciclo de vida del sistema a realizar.

Las **clases de análisis** representan una abstracción de una o varias clases y/o subsistemas de diseño del sistema, estas se centran en el tratamiento de los requisitos funcionales, definiendo responsabilidades a un nivel menos formal, al igual que las relaciones que se establecen entre ellas que son de forma más conceptual que sus contrapartidas de diseño. Tienen atributos y entre ellas se establecen relaciones de asociación, agregación / composición, generalización / especialización y tipos asociativos.

Las clases de análisis están dadas por 3 estereotipos básicos: clase de interfaz, control y entidad. Cada estereotipo estandarizado en UML define una semántica específica para identificar y describir dichas clases además de ayudar a los desarrolladores a distinguir el ámbito de cada una de las clases.

Las clases Interfaz se utilizan para modelar la interacción entre el sistema y los actores, que a menudo envuelve recibir y presentar información y peticiones de los usuarios y sistemas externos y para ellos también. Modelan las partes del sistema que dependen de sus actores y representan generalmente la abstracción de ventanas, formularios, interfaces de comunicaciones, impresoras, sensores, entre otras.

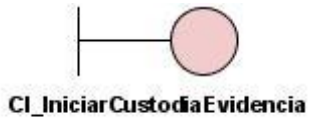


Imagen 2.1: *Representación UML de una clase interfaz.*

Las clases Entidad se utilizan para modelar la información que tiene una larga vida y generalmente es persistente[18], estas modelan la información y el comportamiento relacionado con algún concepto o fenómeno como lo son objetos o sucesos del mundo real, por ejemplo: una persona, un objeto, un auto, una planilla, entre otros. Ellas reflejan la información de modo que benefician a los desarrolladores al diseñar e implementar el sistema, significando el soporte para la persistencia. Estas clases muestran generalmente una estructura de datos lógica y contribuyen a comprender de qué información depende el sistema.

Imagen 2.2: *Representación UML de una clase Entidad.*

Las Clases Control representan la coordinación, secuencia, las transacciones y el control de objetos, usadas con frecuencia para encapsular la lógica del negocio interno de un caso de uso, que no puede asociársele ninguna información concreta y de larga duración almacenada por el sistema (una clase entidad), utilizada también para representar derivaciones, cálculos complejos y modelar los aspectos dinámicos del sistema, ya que permiten coordinar las acciones y flujos principales, delegando el trabajo a otras clases(ya sean entidades o interfaces).

Imagen 2.3: *Representación UML de una clase Control.*

[18] Un objeto se dice persistente cuando es almacenado en un archivo u otro medio permanente. Un programa puede grabar objetos persistentes y luego recuperarlos en un tiempo posterior para su utilización.

2.2. MODELO DE ANÁLISIS

El **Modelo de análisis** contiene las clases del análisis y sus objetos organizados en paquetes que colaboran.

De los temas antes tratados referentes a los procesos de Evidencia, los requisitos y los Casos de Uso antes descritos se evidencia la utilización de 2 entidades principales que son la Evidencia que es donde se almacena todos los datos referentes al elemento clasificado como tal, los datos del colector, el funcionario fotógrafo, los funcionarios presentes en el sitio del suceso, las imágenes relacionadas con la evidencia, la fecha en que fue colectada, entre otras y la Custodia de Evidencia que se almacenan datos como la fecha de inicio de la cadena, el responsable, y la evidencia a la que se le inició la custodia.

A continuación se muestra el diagrama de clases de Análisis correspondiente al submódulo de Evidencias.

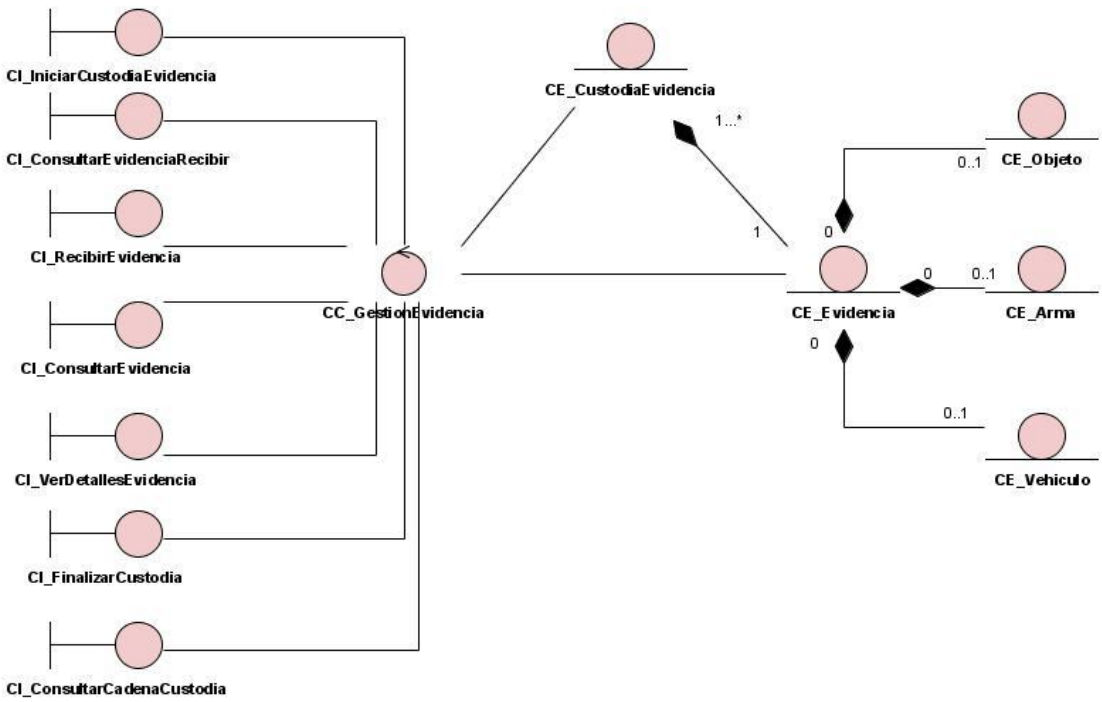


Imagen 2.4: *Diagrama de clases de análisis referentes al submódulo de Evidencias.*

La **Realización de casos de uso del análisis** describe cómo se llevan a cabo y se ejecuta un caso de uso determinado en término de las clases del análisis y de sus objetos en interacción.

Para tener una mayor visión se muestran los Diagramas de Colaboración referentes a los Casos de Usos más importantes.

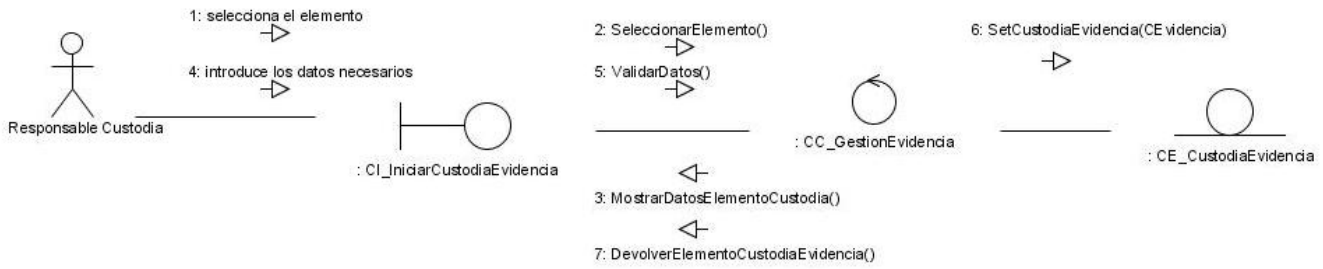


Imagen 2.5: *Diagrama de Colaboración Caso de Uso Iniciar Custodia de Evidencia.*

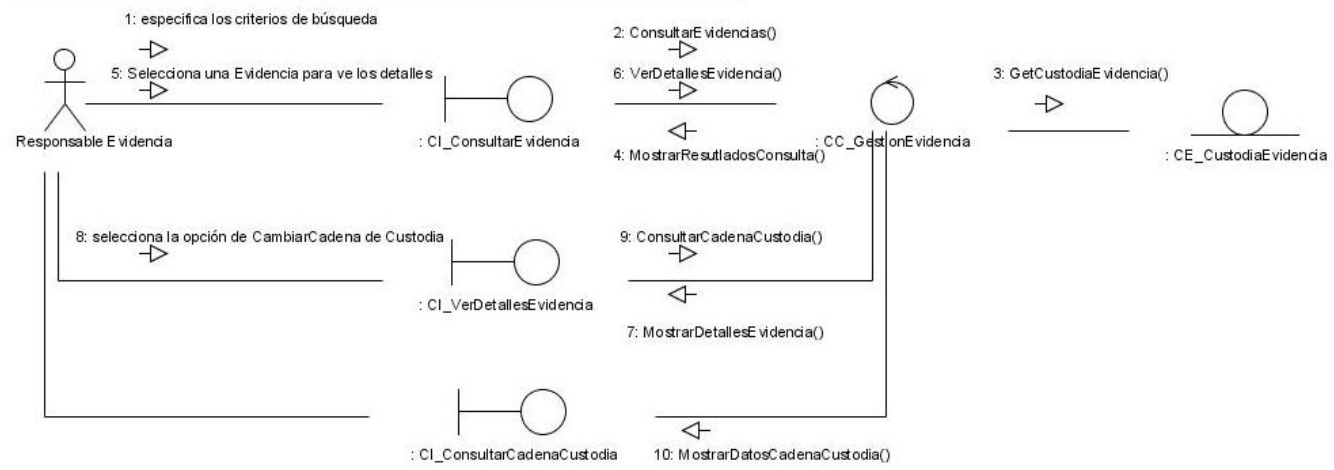


Imagen 2.6: *Diagrama de Colaboración Caso de Uso Consultar Cadena de Custodias de una Evidencia.*

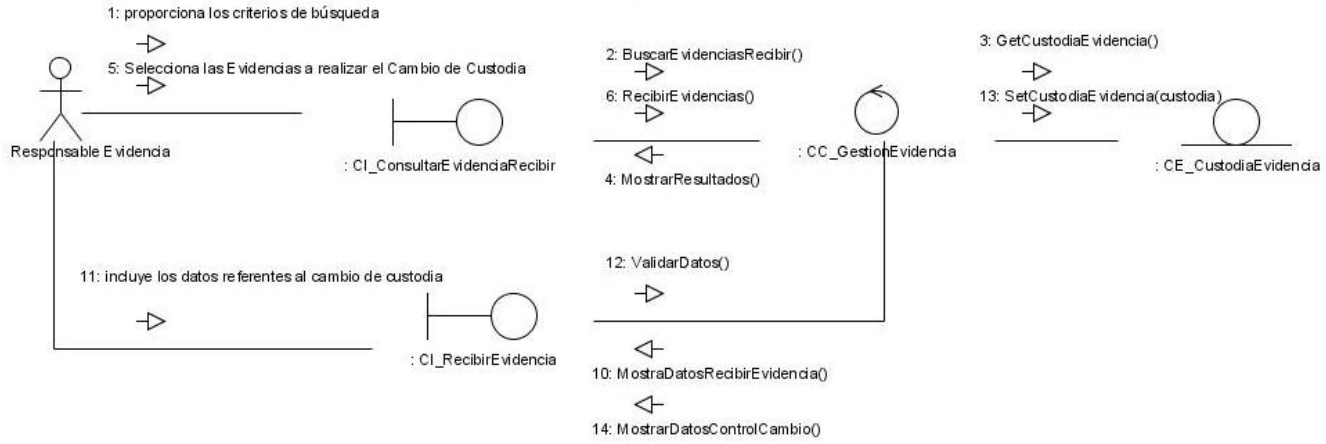


Imagen 2.7: *Diagrama de Colaboración Caso de Uso Recibir Evidencias.*

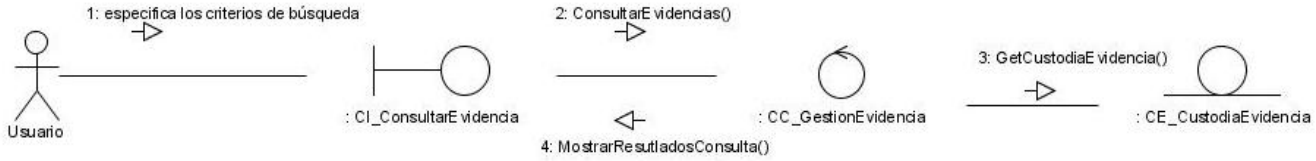


Imagen 2.8: *Diagrama de Colaboración Caso de Uso Consultar Evidencias.*

Se obtuvo como resultado del Análisis que se necesitan al menos 7 clases Interfaces necesarias para la realización de los Casos de Uso del submódulo, una clase Controladora para la parte del negocio referente al submódulo y 2 clases Entidades principales que son la Custodia de Evidencia y la Evidencia como tal. Este modelo de análisis constituye la entrada para el modelo del diseño, pero ello no significa que no aumenten la cantidad de componentes a crear en éste ya que la trazabilidad entre un Modelo de Análisis y un Modelo de Diseño es de 1: n[19] respecto a los componentes que se generan de uno con respecto al otro.

[19] Se refiere a la relación uno: muchos que se especifica de esta forma.

El diseño en pocas palabras

El diseño es el centro de atención al final de la fase de elaboración y el comienzo de las iteraciones de construcción, ello contribuye a una arquitectura estable y sólida, y crear un plano del modelo de implementación. Durante la fase de construcción, cuando la arquitectura es estable y los requisitos están bien entendidos, el centro de atención se desplaza a la implementación.

El diseño es el refinamiento del análisis, teniendo en cuenta los requisitos no funcionales y otras restricciones, además definir CÓMO cumple el sistema los objetivos trazados. Cuando el modelo de diseño es muy preciso, la implementación puede generarse mediante un generador automático de código, teniendo en cuenta para el desarrollo del flujo patrones de diseño[20] para dar cumplimiento a cada una de las funcionalidades definidas. En esta etapa se tienen en cuenta principalmente la arquitectura definida por el equipo de arquitectos del software, el uso de patrones de diseño y el uso de frameworks como buenas prácticas y facilitar el proceso de desarrollo.

El propósito del diseño es especificar una solución que trabaje y pueda ser fácilmente convertida en código fuente y construir una arquitectura simple y fácilmente extensible.

MODELO DE DISEÑO

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en como los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar.

[20] Soluciones que ya existen a los principales problemas que han surgido en el desarrollo de software.

Sirve de abstracción de la implementación y es utilizada como entrada fundamental de las actividades de implementación, por lo que es natural para guardar y mantenerlo a través del ciclo de vida completo del software.

Antes de pasar a la presentación de los diagramas de clases de diseño utilizando estereotipos Web para cada uno de los Casos de Uso, se debe detallar primeramente algunos puntos referentes a la arquitectura definida para el desarrollo de la aplicación final.

Teniendo en cuenta las características del sistema que se desea implementar, que tiene que ser un sistema Web, se ha decidido adoptar una arquitectura cliente-servidor con vista a la instauración de este tipo de software y las características que debe tener las cuales son muy afines con este tipo de aplicaciones distribuidas.

La arquitectura cliente-servidor está definida por ser un sistema distribuido donde los clientes solicitan los servicios y el servidor los provee. Presenta las siguientes características:

- ✓ El cliente no necesita conocer la lógica del servidor ni su ubicación física o geográfica, solo su interfaz externa la cual debe ser única y bien definida.
- ✓ Se abstrae tanto al hardware como el software que ambos soportan facilitando el uso de las aplicaciones así como el acceso global de estas.
- ✓ El cliente y el servidor pueden actuar como una sola entidad o como entidades separadas, realizando diferentes tareas y actividades concurrentemente.
- ✓ Los cambios en el servidor no reportan ningún o pocos cambios en el cliente.

Con el uso de este tipo de arquitectura de obtienen un grupo de ventajas como son:

- **Centralización del control:** los accesos, recursos y la integridad de los datos son controlados por el servidor de forma que la aplicación no pueda ser dañada por datos no

válidos o por alguien no autorizado a su uso; también facilita el mantenimiento y actualización de los datos.

- **Escalabilidad:** Se puede aumentar la capacidad de trabajo del cliente y el servidor por separado al igual que la cantidad de elementos en la red, tanto clientes como servidores.
- **Fácil mantenimiento:** Los cambios a nivel del servidor de cualquier tipo, ya sea reparaciones, actualizaciones, reemplazo o traslado del mismo no afectarán o lo harán de forma mínima[21] a los clientes.

La seguridad en las transacciones, sencillez de la interfaz, y la facilidad de empleo de las mismas, hace que sea tan utilizado en la actualidad.

Se definió para la aplicación final por parte del equipo de Arquitectura, una arquitectura de n-capas de tres niveles, la que ofrece mayor modularidad, simplificación en el desarrollo, aumento de la mantenibilidad y mayor escalabilidad para el producto, definiendo un grupo de clases las que influyen en el desarrollo como son la clase BaseBean en la capa de presentación que contiene un conjunto de funcionalidades afines para los beanes de respaldo de las páginas como lo establece JSF, por lo que todos los beanes deben heredar de dicha clase.

En los demás paquetes del subsistema también se tienen un grupo de clases que tienen un grupo de funcionalidades comunes para las diferentes clases dentro de cada paquete. Una de las clases que se relaciona con el paquete se servicio tienen relación con la clase Carga la cual se encarga de inicializar los objetos transaccionales definidos por Hibernate con solo los datos necesarios (una propiedad de Hibernate, en el contexto dicho objeto fue declarado *lazy-initilization*) y eliminar el proxy a estos objetos ya que si el objeto esta en este estado no tiene cargado los valores que se necesitan y provocan un error en la aplicación si se piden esos valores.

[21] Esta independencia de los cambios en cualquiera de las partes involucradas también se conoce como encapsulación.

En el caso de la capa de acceso a Datos, los objetos responsabilizados deben heredar de DaoGenerico y de DaoGenericoImpl que poseen un grupo de funcionalidades comunes para dichos elementos al igual que las entidades del dominio heredarán de la clase EntidadPersistenteBase.

2.2.1. DIAGRAMA DE PAQUETES

En UML, un **diagrama de paquetes** muestra como un sistema está dividido en agrupaciones lógicas, mostrando las dependencias entre esas agrupaciones. Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema.

Los Paquetes están normalmente organizados para maximizar la coherencia interna dentro de cada paquete y minimizar el acoplamiento externo entre los paquetes*. (Ver Imagen 2.9)*

2.2.2. DIAGRAMA DE CLASES DEL DISEÑO

El **Diagrama de Clases de Diseño** es la captura la estructura lógica del sistema - las clases y elementos que constituyen el modelo -. Es un modelo estático, describiendo lo que existe y qué atributos y comportamiento tiene, más que cómo se hace algo. Los diagramas de Clases son los más útiles para ilustrar las relaciones entre las clases e interfaces. Las generalizaciones, las agregaciones y las asociaciones son todas valiosas para reflejar la herencia, la composición o el uso y las conexiones respectivamente.

¿Qué son los patrones de diseño?

Los **patrones de diseño** son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Algunos patrones utilizados en el Diseño

Para realizar el diseño del submódulo se tuvo en cuenta un grupo de patrones de diseño definidos por el equipo de la arquitectura como lo es el patrón **Modelo Vista Controlador (MVC)** que es un patrón de arquitectura de software que separa los datos de la aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos:

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 todo derivar nuevos datos; por ejemplo, no permitiendo valores nulos en los valores de validación como la credencial del funcionario.
Vista: Este presenta el modelo en un formato adecuado para interactuar, 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.

Ello permite que los cambios en la vista no afecten el resto de la aplicación, permitiendo mostrar distintas vistas sin afectar el modelo. Ejemplo de la implementación de este patrón se encuentra en sí en los frameworks utilizados como JSF que permiten implementar este patrón donde se tiene que el modelo son las clases de domino utilizadas como la clase Evidencia o CustodiaEvidencia, la vista las páginas .jsp (ejemplo: iniciarCustodiaEvidencia.jsp) creadas y el controlador los beanes manejados o de respaldo como lo es IngresarEvidenciaNoActaProcesalManejado.

Otro de los patrones utilizado en el diseño es ***Inyección de Dependencias (Dependency Injection)***, este se basa en que el objeto define sus dependencias (o sea, los objetos con los que trabaja) solo a través de argumentos del constructor, argumentos de un método de factory o propiedades que se registran en el objeto después de su construcción. Por tanto es el trabajo del contenedor el inyectar estas dependencias cuando se crea el bean. Esto es fundamentalmente lo inverso (de aquí la aplicación de la Inversión de Control) de la técnica tradicional del bean resolviendo sus propias dependencias usando construcción directa de las clases o algún tipo de patrón de creación o localización, mejorando considerablemente el rendimiento de la aplicación.

Se hace evidente con el uso que el código se vuelve mucho más claro cuando se aplica el principio DI, alcanzando un grado más alto de desacoplamiento debido a que los beans no buscan sus dependencias de forma activa ni conoce su posición ni tipo concreto. Este patrón es muy utilizado por el framework de negocio Spring.

El patrón de diseño ***fachada o facade*** es un patrón de Estructura que se utilizó para proveer de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas. Con un conjunto de prestaciones, este permite reducir la dependencia de código externo en los trabajos internos de implementación, ya que la mayoría del código lo usa facade, permitiendo así más flexibilidad en el desarrollo de sistemas, desacoplando la implementación de la interfaz que se presenta. Ejemplo de ello es la interfaz EvidenciaFacade la cual posee los métodos necesarios para realizar las funcionalidades relacionadas con las Evidencias necesarias por los beanes manejados, pero sin saber la implementación interna de la misma. Este está muy relacionado al Service Locator ya que este es el que se encarga a diferencia de la inyección de dependencia, que el objeto tenga que ser el responsable de buscar su dependencia en el contenedor. La implementación de dicha fachada es la EvidenciaFacadeImpl en la cual se tendrá acceso a todas las clases de servicios del subsistema para así delegar la funcionalidad a cada uno de los servicios, propiciando el aumento de la cohesión del sistema y la escalabilidad del código de la aplicación.

Con el uso del patrón **Bajo acoplamiento** se propició tener las clases ligadas entre sí lo menor posible, de forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión en el resto de ellas, potenciando la reutilización y disminuyendo la dependencia.

Para la abstracción en el acceso a datos se utilizó el patrón **DAO (Data Access Object o Objetos de Acceso a Datos)** el cual se encarga de abstraer y encapsular todos los accesos a la fuente de datos, ocultando completamente los detalles de implementación del acceso a datos a la capa superior mediante una interfaz que es la que se utiliza. Esta interfaz permite que la implementación se abstraiga a los esquemas de almacenamiento de los datos persistentes y conexión a la Base de Datos.

Para un mejor entendimiento de los diagramas de clases de diseño desarrollados, se excluyeron las relaciones con las clases BaseBean, DaoGenerico, Carga, EntidadPersistenteBase y otras clases que no fueron usadas pero no desarrolladas por el equipo de trabajo, permitiendo ganar en claridad y sencillez en los diagramas presentados. También se crearon 2 diagramas por cada caso de uso para facilitar la comprensión, uno para la Capa de Presentación y otro donde se especifican las relaciones de los beanes manejados con las otras capas de la aplicación.

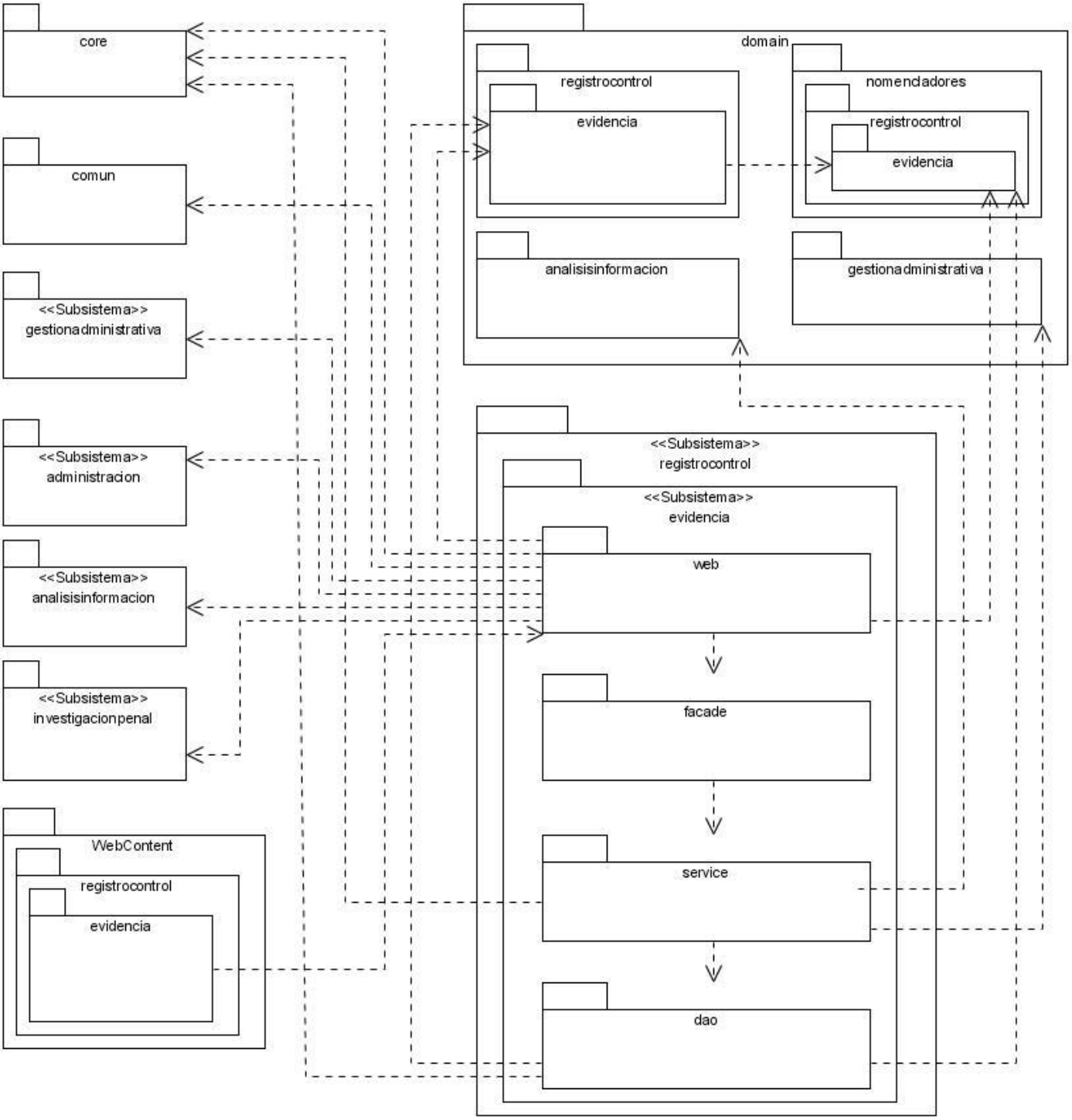


Imagen 2.9: *Diagrama de relación de paquetes del sub-sistema Evidencia con otros paquetes presentes en la aplicación general.*

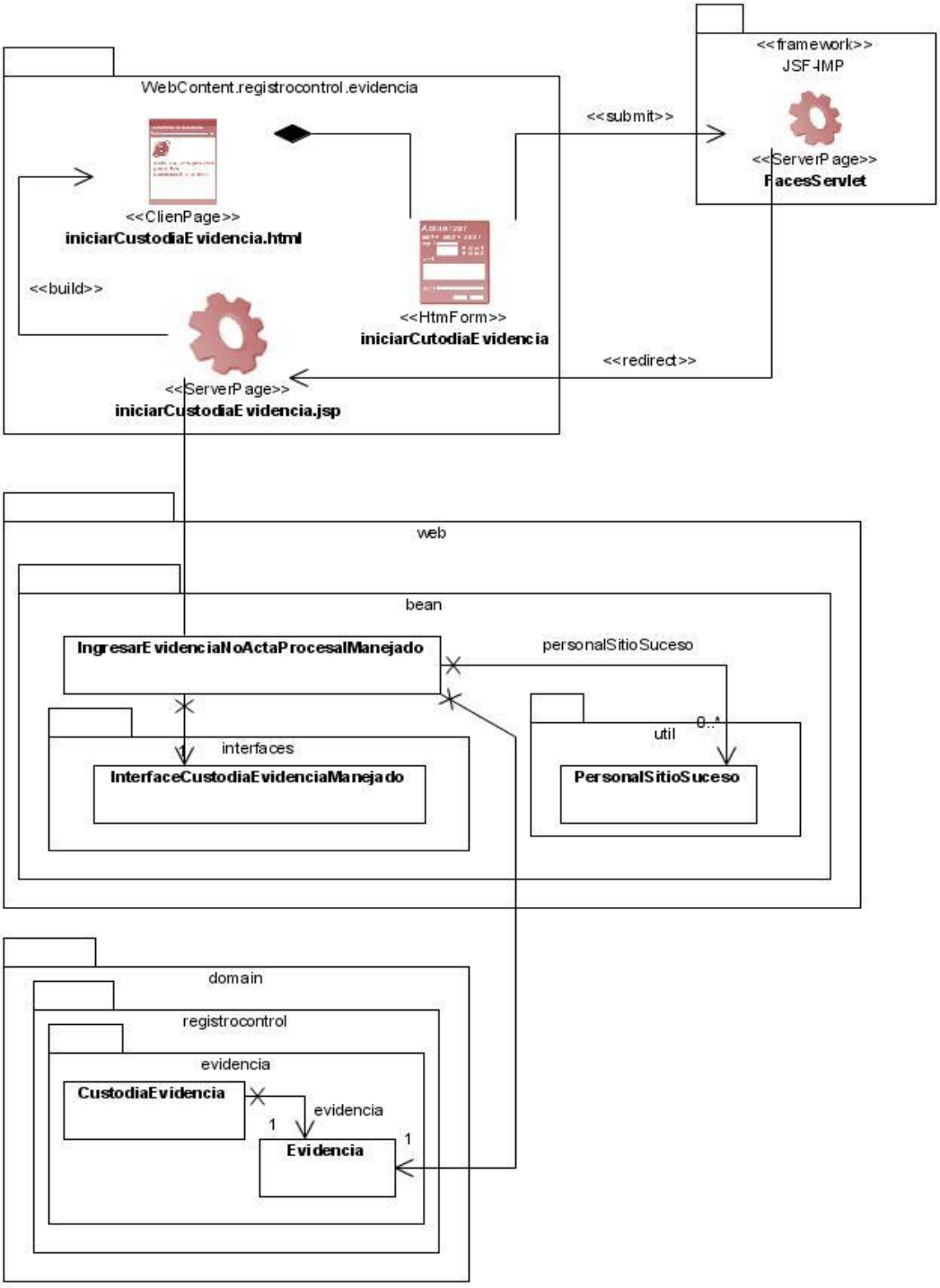


Imagen 2.10: *Diagrama de Clases de Diseño. Caso de Uso Iniciar Cadena de Custodia. Capa de Presentación.*

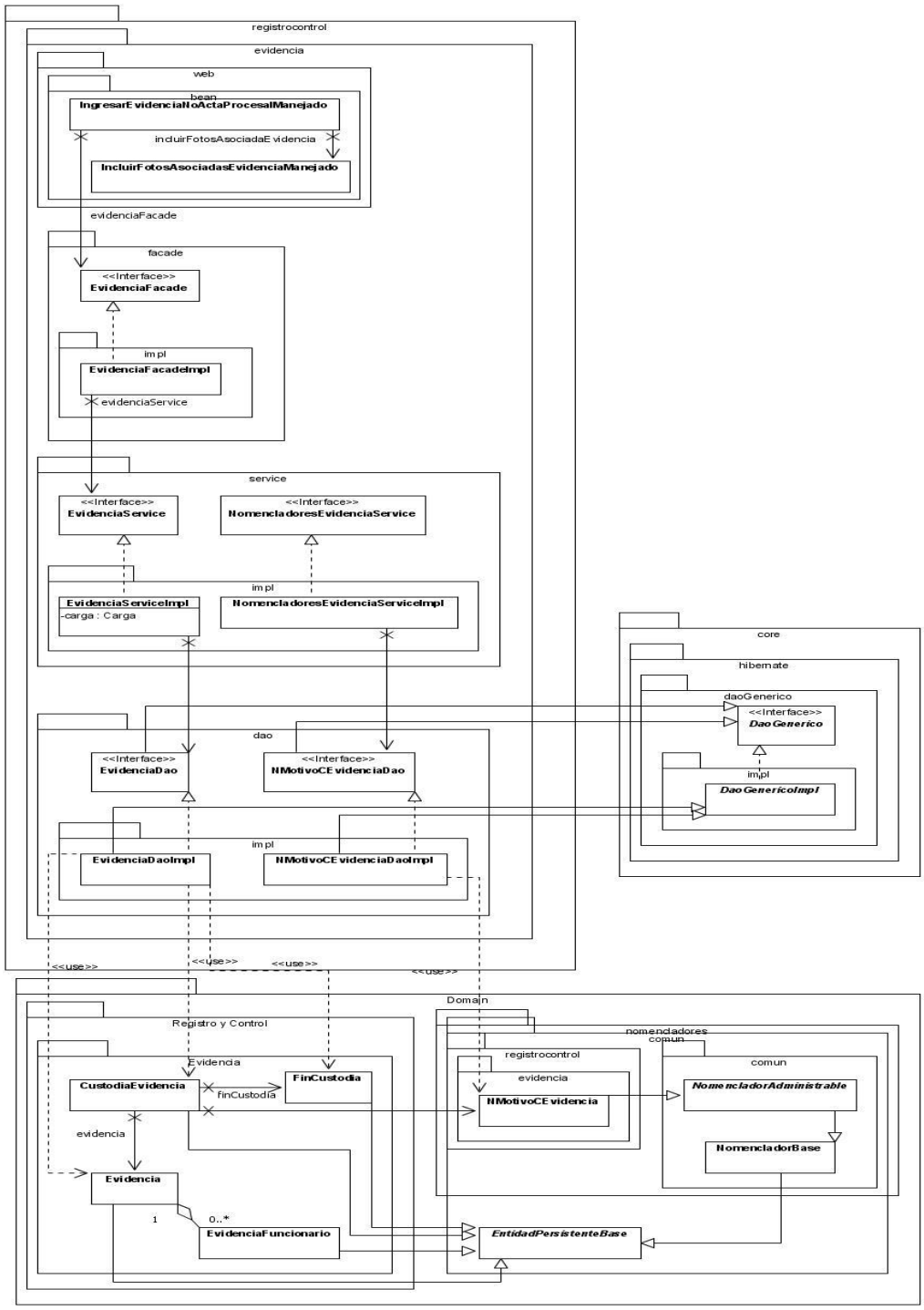


Imagen 2.11: *Diagrama de Clases de Diseño. Caso de Uso Iniciar Cadena de Custodia. Capas presentación, negocio y acceso a datos.*

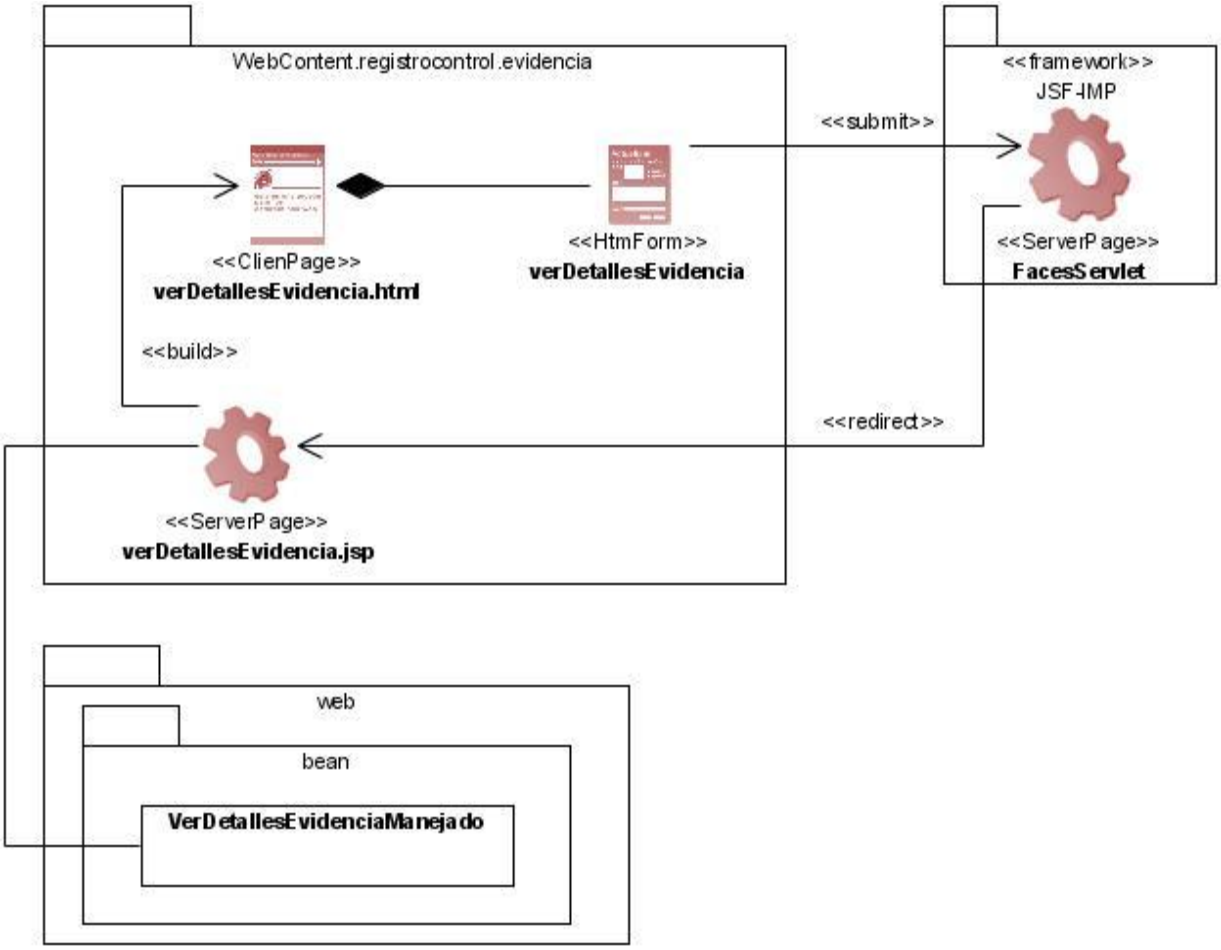


Imagen 2.12: *Diagrama de Clases de Diseño. Caso de Uso Ver detalles de Evidencia. Capa de Presentación.*

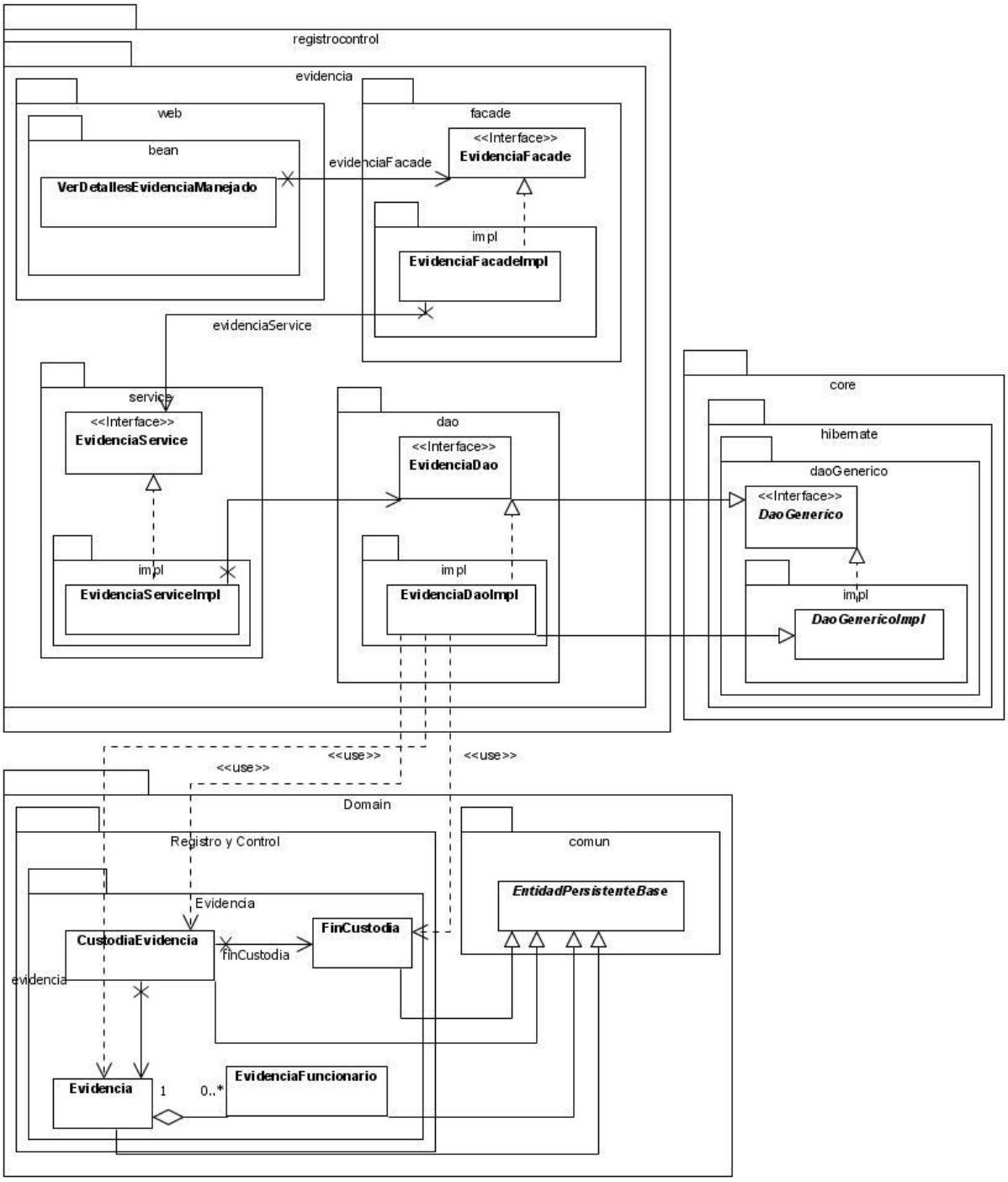


Imagen 2.13: *Diagrama de Clases de Diseño. Caso de Uso Ver detalles de Evidencia. Capas presentación, negocio y acceso a datos.*

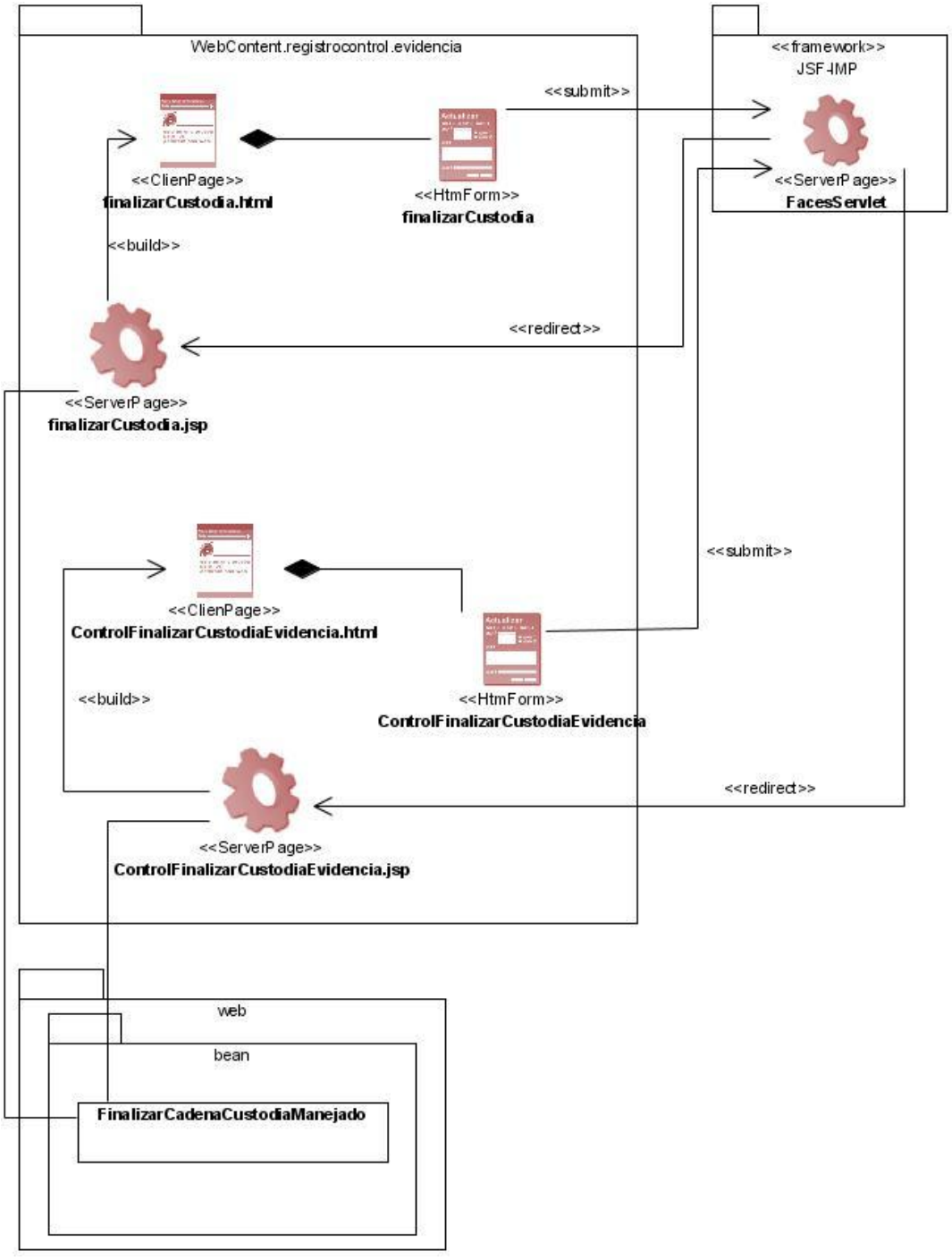


Imagen 2.14: *Diagrama de Clases de Diseño. Caso de Uso Finalizar Cadena de Custodia. Capa de Presentación.*

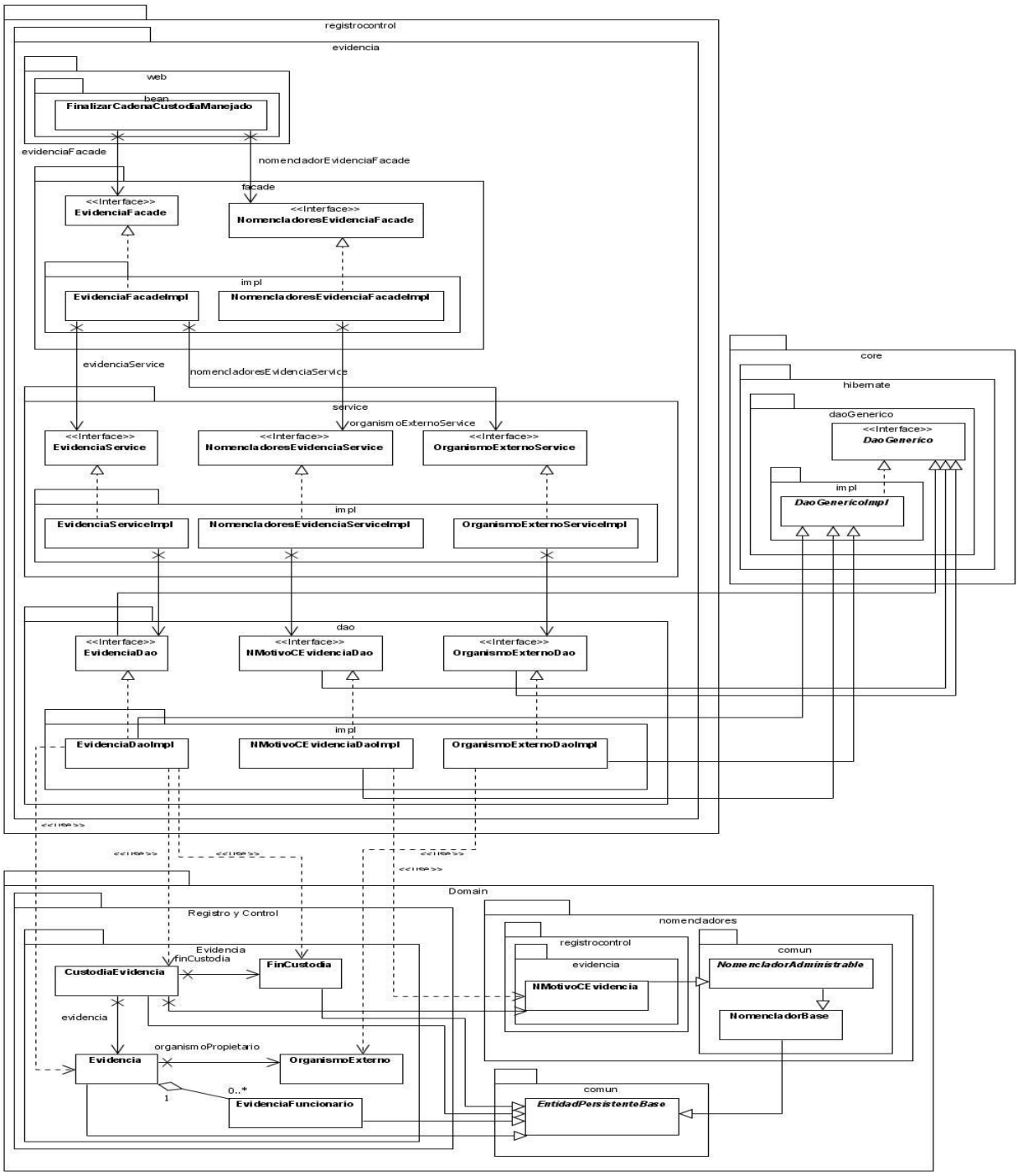


Imagen 2.15: *Diagrama de Clases de Diseño. Caso de Uso Finalizar Cadena de Custodia. Capas presentación, negocio y acceso a datos.*

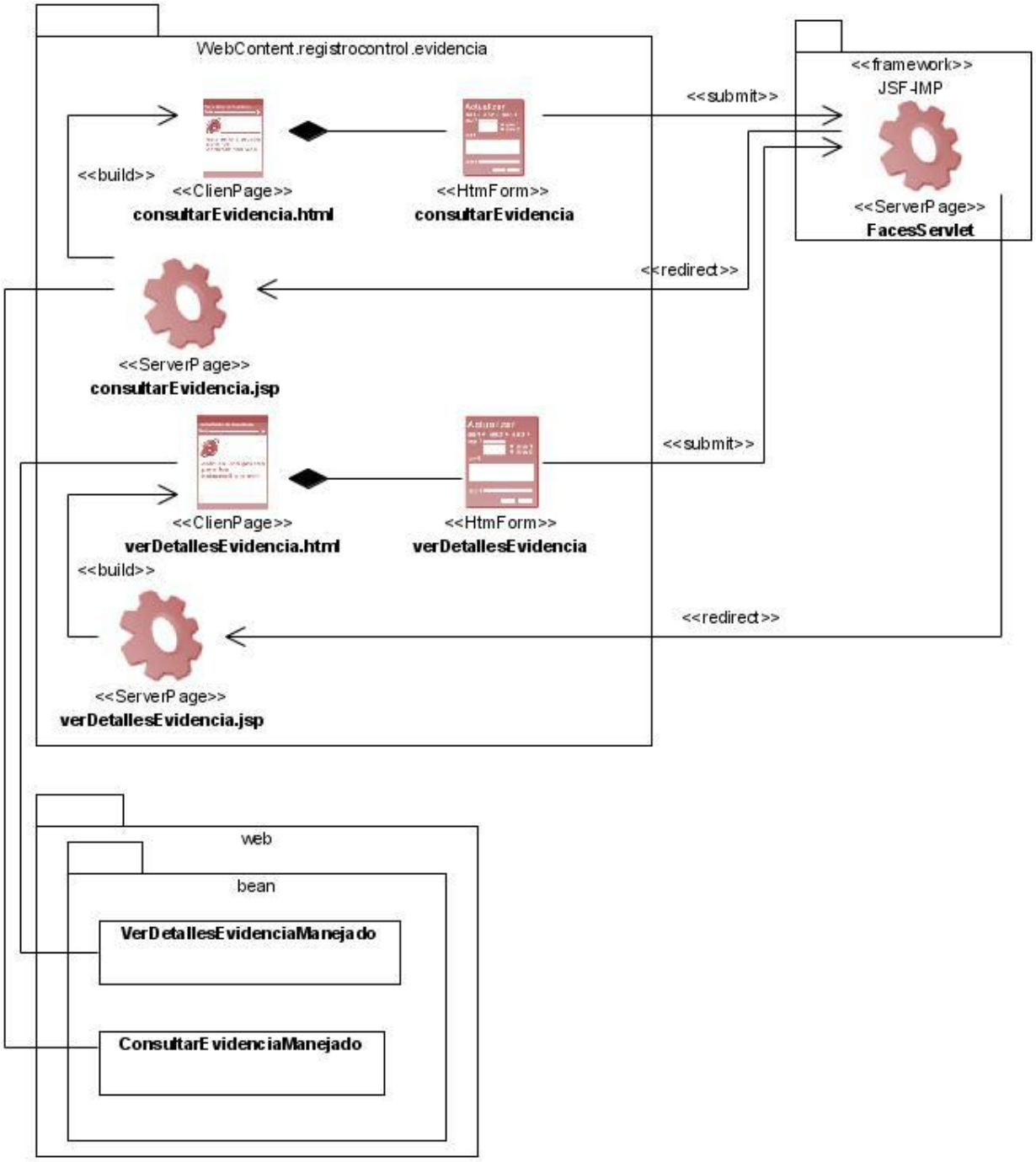


Imagen 2.16: *Diagrama de Clases de Diseño. Caso de Uso Consultar Evidencia. Capa de Presentación.*

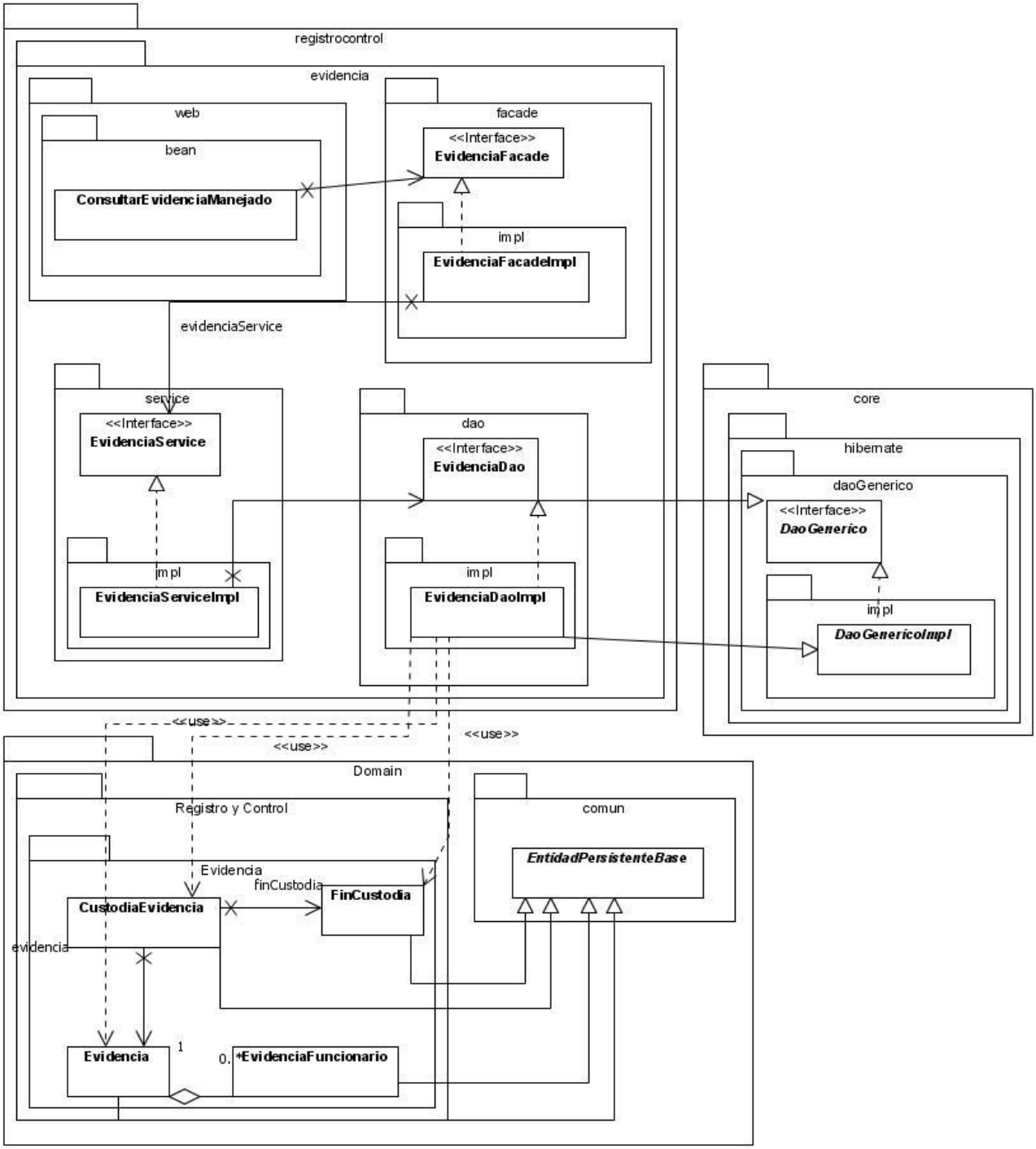


Imagen 2.17: *Diagrama de Clases de Diseño. Caso de Uso Consultar Evidencia. Capas presentación, negocio y acceso a datos.*

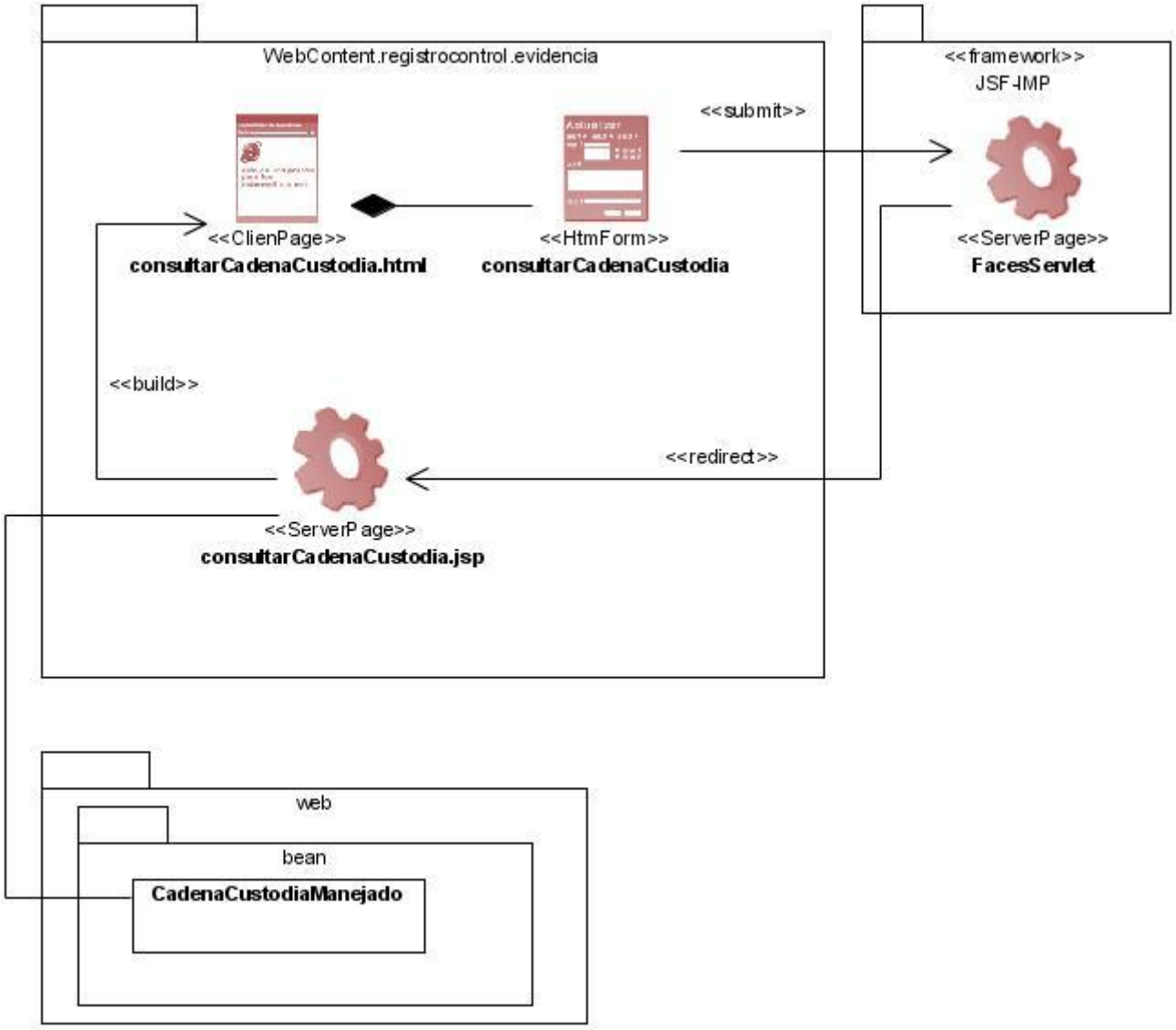


Imagen 2.18: *Diagrama de Clases de Diseño. Caso de Uso Consultar Cadena de Custodia. Capa de Presentación.*

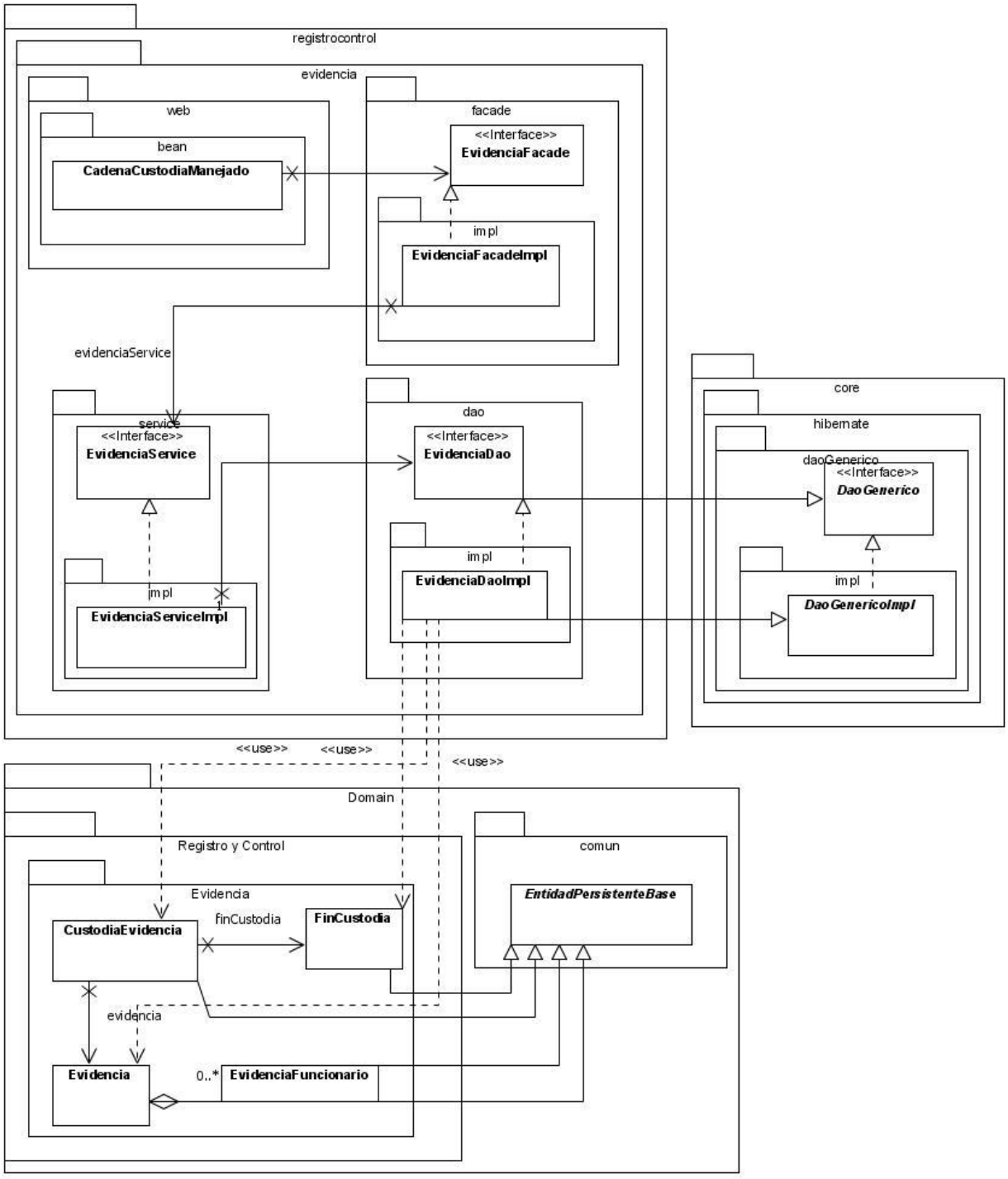


Imagen 2.19: *Diagrama de Clases de Diseño. Caso de Uso Consultar Cadena de Custodia. Capas presentación, negocio y acceso a datos.*

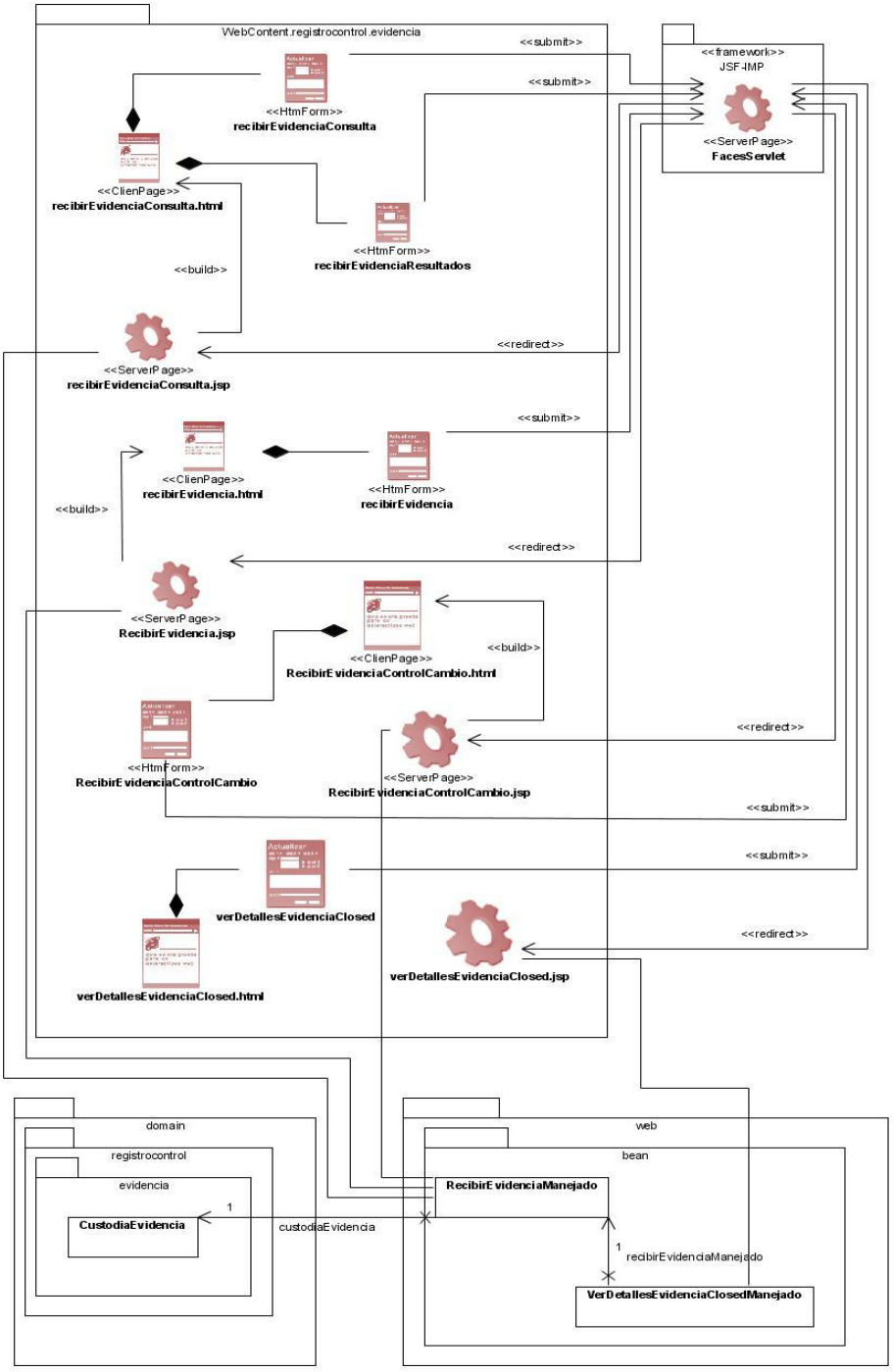


Imagen 2.20: *Diagrama de Clases de Diseño. Caso de Uso Recibir Evidencia. Capa de Presentación.*

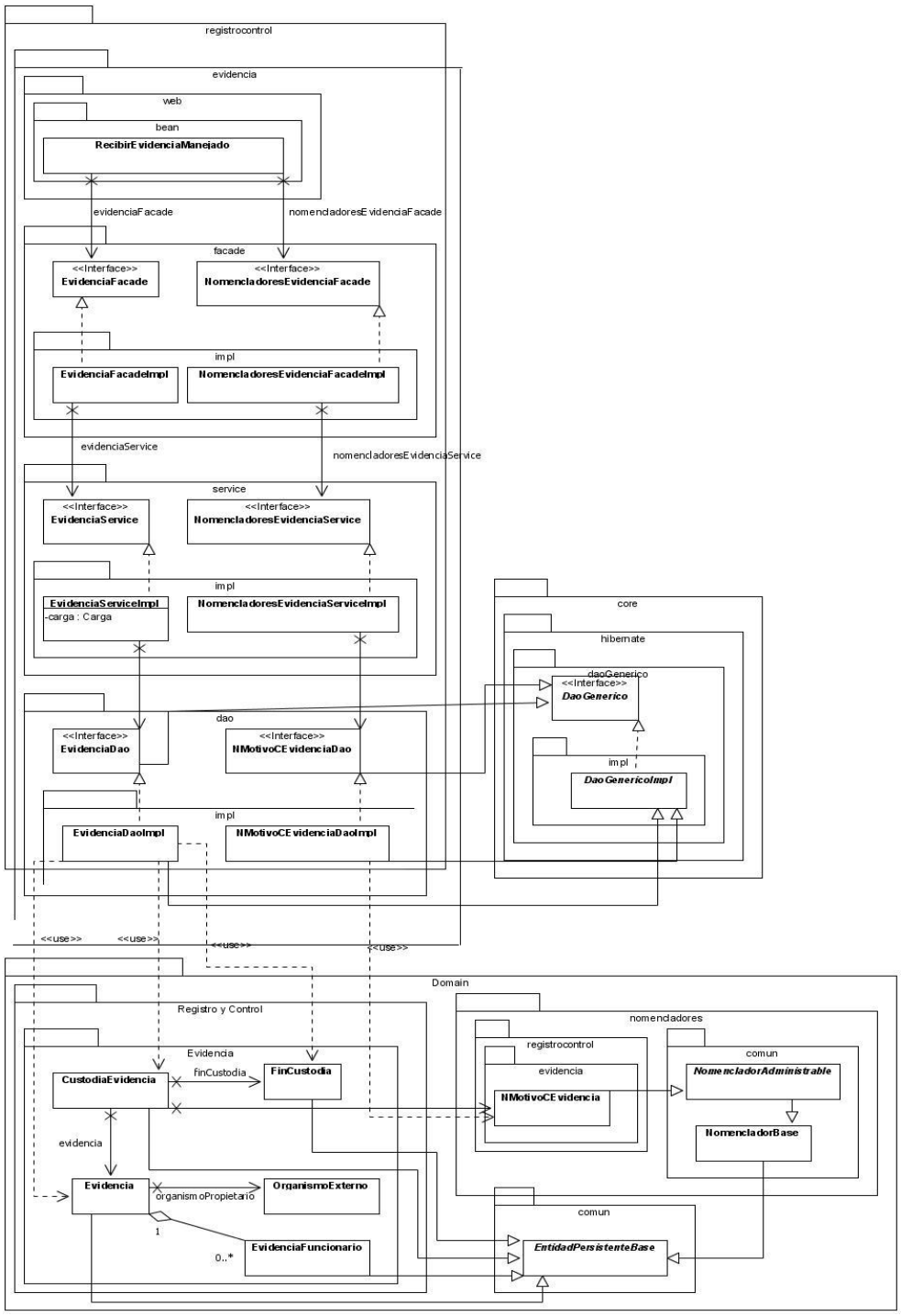


Imagen 2.21: *Diagrama de Clases de Diseño. Caso de Uso Recibir Evidencia. Capas presentación, negocio y acceso a datos.*

A continuación se describen textualmente algunas de las clases más relevantes desarrolladas en el proceso de diseño.

Nombre	**IngresarEvidenciaNoActaProcesalManejado**
Tipo de clase	Bean de respaldo para iniciar la custodia de un elemento calificado como evidencia.

Atributo	**Tipo**
seguridadFacade	SeguridadFacade
evidenciaFacade	EvidenciaFacade
analisisInformacion	AnalisisInformacionFacade
nomencladorFacade	NomencladorFacade
administracionFacade	AdministracionFacade
incluirFotosAsociadasEvidenciaManejado	IncluirFotosAsociadasEvidenciaManejado
comunicacion	Comunicacion
expediente	Expediente
actaProcesalExterna	ActaProcesalExterna

numeroDiligencia	String
funcionarioRegistrado	Funcionario
nuevaEvidencia	Evidencia
fechaColecto	Date
horaColecto	Date
direccionColecta	Direccion
listaEstado	List<EstadoVenezolano>
elemento	Elemento
funcionarioFotografo	Funcionario
credencialFotografo	String
interfaceCustodiaEvidenciaManejado	InterfaceCustodiaEvidenciaManejado
reRender	String
yaValidado	boolean
tipoFuncionario	String
tipoFuncionario	String

credencial	String
seleccionadoEsInterno	boolean
funcionarioInterno	Funcionario
validadoInterno	boolean
mostrarTabla	boolean
dataTableX	UIData
listadoFuncionarios	List<PersonalSitioSuceso>
funcionariosPresentes	List<EvidenciaFuncionario>
credencialFuncionarioExterno	String
organismo	String
selectItemOrganismo	List<SelectItem>
organismoExterno	EnteExterno
letraCedula	String
numeroCedula	String
primerNombre	String

segundoNombre	String
primerApellido	String
segundoApellido	String

Para cada responsabilidad:	
Nombre	verTipoFuncionario(ActionEvent e)
Descripción	Método para el cambio de funcionario a incluir en la página
Nombre	incluirFuncionario(ActionEvent e)
Descripción	Incluye un funcionario presente en el sitio del suceso a la evidencia
Nombre	eliminarFuncionario(ActionEvent e)
Descripción	Metodo para disociar un funcionario que se asoció a la evidencia
Nombre	isInicializarIncluidos()
Descripción	Método para inicializar lo elementos para asocias fotos a la evidencia
Nombre	iniciarCustodia(ActionEvent e)
Descripción	Método que permite iniciar la custodia de una evidencia

Nombre	cancelar(ActionEvent e)
Descripción	Método para cancelar la inicialización
Nombre	validarCredencialFuncionarioFotografo(ActionEvent e)
Descripción	método para validar la credencial del funcionario fotógrafo
Nombre	validarFuncionarioResente(ActionEvent e)
Descripción	Método para validar los funcionarios presentes en el sitio del suceso
Nombre	getFuncionarioRegistrado()
Descripción	Método que permite obtener el funcionario logueado en la aplicación

Tabla No. 2: *Descripción textual clase IngresarEvidenciaNoActaProcesalManejado, desarrollada en el proceso de diseño del submódulo Evidencias.*

Nombre	**EvidenciaFacadeImpl**
Tipo de clase	Fachada para el submódulo Evidencia

Atributo	**Tipo**
evidenciaService	EvidenciaService
seguridadFacade	SeguridadFacade
investigacionPenalFacade	InvestigacionPenalFacade
investigacionForenseFacade	InvestigacionForenseFacade
analisisInformacionFacade	AnalisisInformacionFacade
gestionAdministrativaFacade	GestionAdministrativaFacade
evidenciaCarga	Carga
organismoExternoService	OrganismoExternoService

Para cada responsabilidad:	
Nombre	autenticarResponsable(Integer idEvidencia, String pass)
Descripción	Busca el responsable de una evidencia de ID dado y autentica su usuario y contraseña de SIIPOL

Nombre	numeroEvidencia(Comunicacion comunicacion, Expediente exp, ActaProcesalExterna actaProcesalExterna)
Descripción	Crea el numero de la Evidencia
Nombre	cambiarCustodiaEvidencia(Integer motivo, Date FechaEntrega, Integer idEvidencia, Funcionario solicitante, Comunicacion comunicacion, String Observaciones)
Descripción	Cambia la custodia de la Evidencia
Nombre	finalizarCadenaCustodia(Funcionario responsable, Date fechaFinalizada, Evidencia evidencia, String lugar, Integer causa, String justificacion, Comunicacion comunicacion)
Descripción	Finaliza la Cadena de custodia de una Evidencia
Nombre	guardarOModificar(Evidencia evidencia)
Descripción	Guarda una Evidencia
Nombre	obtenerCustodiaEvidencia(Integer idEvidencia)
Descripción	Obtiene la ultima custodia de una Evidencia dado su id
Nombre	filtrarEvidencias(List<Elemento> listaElementos)
Descripción	Filtra una Lista de Elementos y devuelve de ellos los que sean Evidencias en una Lista.

Nombre	obtenerPorEjemplo(Evidencia evidencia)
Descripción	Obtiene todos las que concuerden con un ejemplo.
Nombre	consultarCadenaCustodia(Integer idEvidencia)
Descripción	Obtiene todas la cadena de custodia de una Evidencia dado su Id
Nombre	obtenerPorId(Integer id)
Descripción	Obtiene una evidencias dado su id
Nombre	iniciarCustodiaEvidencia(Evidencia evidencia)
Descripción	Registra la cadena de custodia de una Evidencia
Nombre	consultarEvidencias(String noActaProcesal, String noEvidencia, String noActaProcExt, String pnombre, String snombre, String papellido, String sapellido, String horaColeccionInicial, String horaColeccionFinal, Date fechaInicio, Date fechaFin, String tipoOrden, String modoOreden)
Descripción	Devuelve un listado de custodias de evidencias dados parámetros opcionales de consulta.
Nombre	obtenerPrimeraCustodia(Integer idEvidencia)
Descripción	Obtiene la primera custodia dado el id de la evidencia
Nombre	obtenerPorIdUI(Integer id)

Descripción	Obtiene la evidencias dado su id
Nombre	obtenerCustodiaEvidenciaUI(Integer idEvidencia)
Descripción	Obtiene la primera cadena de custodia dado el id de la evidencia
Nombre	obtenerEvidenciaPorActaProcesal(String numero)
Descripción	Obtiene una lista de evidencias asociadas a un número de acta procesal
Nombre	propietarioEvidencia(Integer idEvidencia)
Descripción	Busca a la persona propietaria de una evidencia finalizada por entrega al propietario
Nombre	obtenerEvidenciasFuncionario(String credencial, Date fechaI, Date fechaF)
Descripción	Lista todas las ultimas custodias de evidencias de las que un funcionario es responsable y que se inicio su custodia en un rango de fechas dado
Nombre	perteneceElementoEvidencia(Integer idElemento)
Descripción	Verifica si un elemento dado está asociado a una evidencia
Nombre	incluirOrganismoExterno(OrganismoExterno org)
Descripción	Incluye un nuevo organismo propietario de una evidencia
Nombre	obtenerOrganismoExternoPorRif(String rif)

Descripción	Obtiene el organismo externo dado su número de Rif
Nombre	estaFinalizadaEvidencia(Integer idEvidencia)
Descripción	Retorna 0 si la evidencia no está finalizada y si está finalizada, el Número de causa de finalización
Nombre	bloquearEvidencias(List<Evidencia> listaEvidencias)
Descripción	Permite bloquear una lista de evidencias
Nombre	cambiarCustodiaEvidencia(Elemento elemento, Funcionario funcionario)
Descripción	cambia la custodia de la(s) evidencia(s) que contengan al elemento dado a un funcionario dado
Nombre	evidenciaCustodiadaPor(Integer idfuncionario, Integer idEvidencia)
Descripción	comprueba que una evidencia determinada este siendo custodiada en ese momento por un funcionario determinado
Nombre	elementoEsEvidenciaCustodiadaPor(Integer idElemento, Integer idFuncionario)
Descripción	Verifica que el elemento pasado es evidencia y que el funcionario pasado es el responsable de la custodia
Nombre	finalizarEvidenciaDARFA(List<Elemento> listaObjetos, Funcionario funcionario, String lugar)

Descripción	Finaliza todas las evidencias que contengan los objetos especificados y que en ese momento estén siendo custodiadas por el funcionario especificado
Nombre	obtenerCustodiaEvidenciaSinInicializar(Integer idEvidencia)
Descripción	Obtiene la ultima custodia de una evidencia dado el id de la evidencia.
Nombre	buscarEvidenciaPorElemento(Integer idElemento)
Descripción	Obtiene la evidencias dado el id del elemento
Nombre	finalizarEvidenciaPorOrdenFiscal(Elemento elemento, Funcionario funcionario)
Descripción	Finaliza la custodia de la Evidencia por Orden Fiscal

Tabla No. 3: *Descripción textual clase EvidenciaFacadeImpl, desarrollada en el proceso de diseño del submódulo Evidencias.*

2.2.3. REALIZACIONES DE CASOS DE USO

Los diagramas de contrato entre paquetes que se presentan a continuación representan la realización de los Casos de Uso Significativos del sub- módulo de Evidencias.

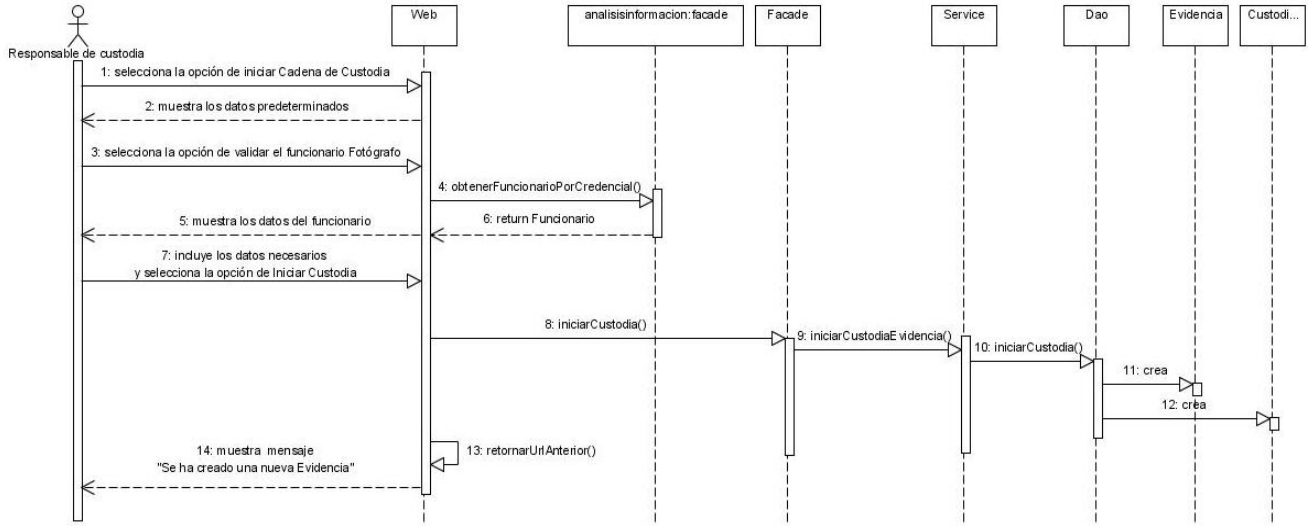


Imagen 2.22: *Diagrama de contrato entre paquetes Caso de Uso Iniciar Custodia de Evidencia.*

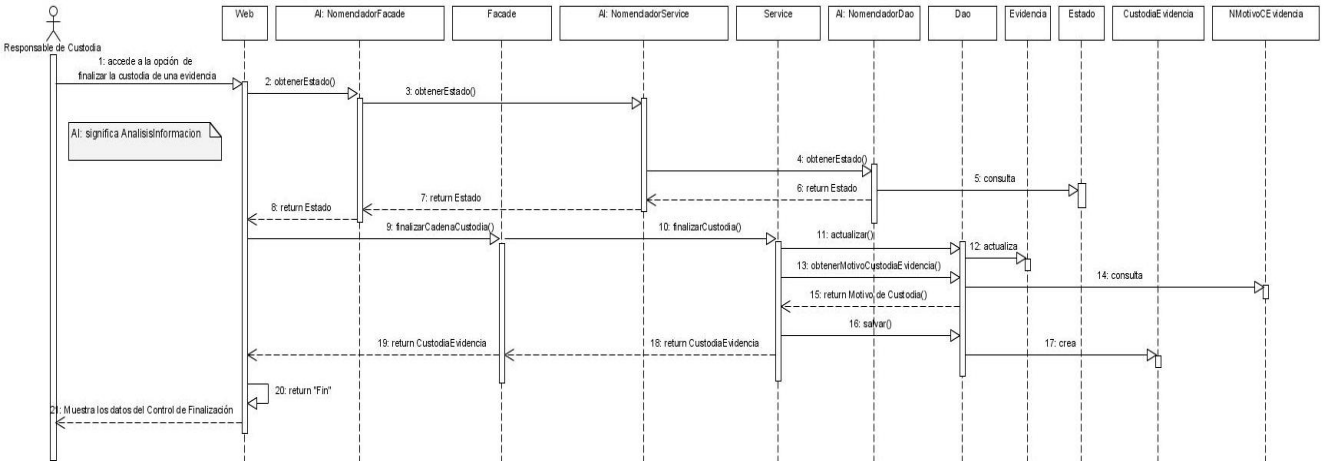


Imagen 2.23: *Diagrama de contrato entre paquetes Caso de Uso Finalizar Custodia de Evidencia.*

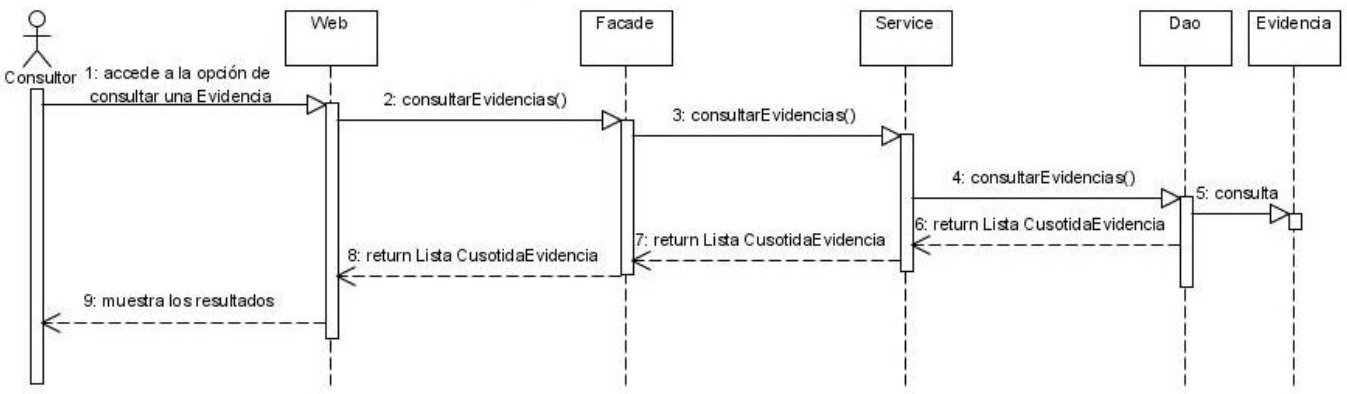


Imagen 2.24: *Diagrama de contrato entre paquetes Caso de Uso Consultar Evidencias.*

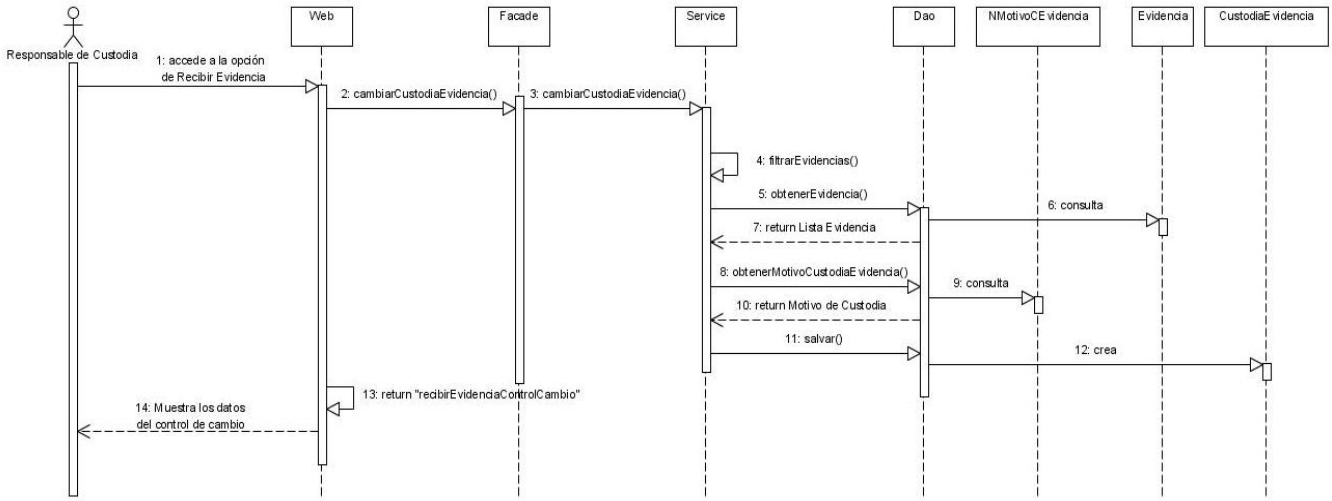


Imagen 2.25: *Diagrama de contrato entre paquetes Caso de Uso Recibir Evidencia.*

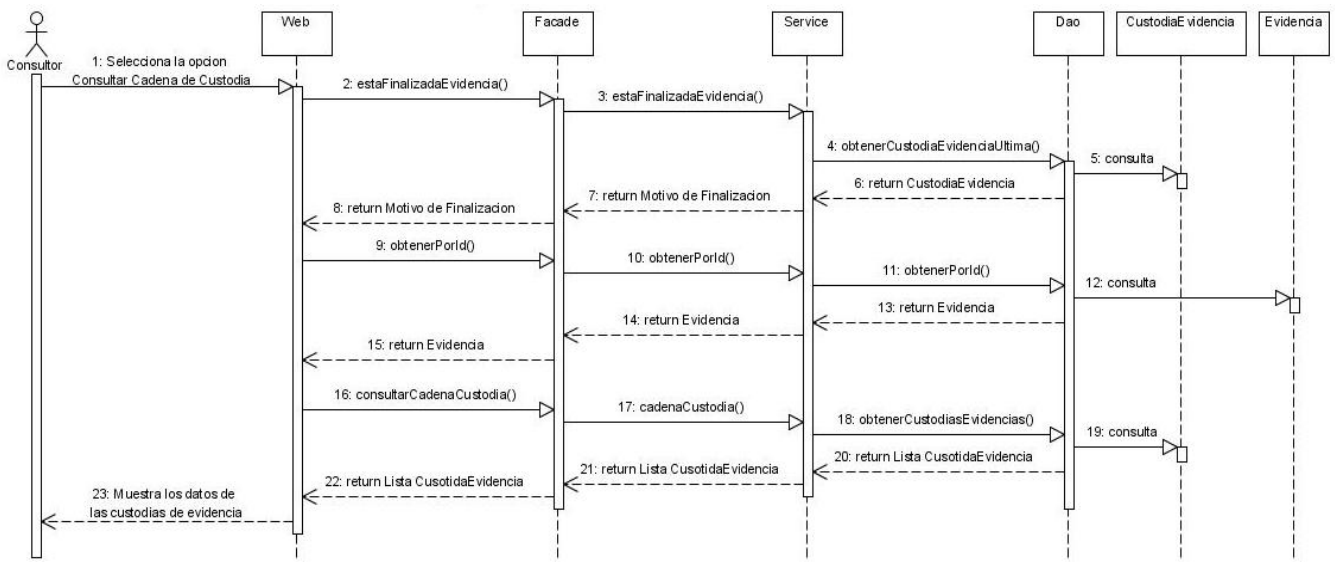


Imagen 2.26: *Diagrama de contrato entre paquetes Caso de Uso Consultar Cadena de Custodia de Evidencia.*

2.3. MODELO DE DATOS

Un **modelo de datos** es un conjunto de conceptos, reglas y convenciones que permiten describir y manipularlos datos de la parcela de un cierto mundo real que deseamos almacenar en la base de datos.

Son un conjunto de herramientas conceptuales para describir la representación de la información en términos de datos. Los modelos de datos comprenden aspectos relacionados con: estructuras y tipos de datos, operaciones y restricciones.

Es básicamente una “descripción” de una base de datos así como de los métodos para almacenar y recuperar información de esos contenedores. Son abstracciones que permiten la implementación de un sistema eficiente de base de datos; por lo general se refieren a algoritmos, y conceptos matemáticos. Por lo general permite describir las estructuras de la base de datos (el tipo de los datos que incluye la base y la forma en que se relacionan), las restricciones de integridad (las condiciones que los datos deben cumplir para reflejar correctamente la realidad

deseada) y las operaciones de manipulación de los datos (agregado, borrado, modificación y recuperación de los datos de la base).

2.3.1. DIAGRAMA DE CLASES PERSISTENTES

El diagrama que se presenta continuación representa la relación de las clases persistentes Evidencia y la CustodiaEvidencia con las demás clases del dominio, las cuales se encuentran dentro del paquete cicpc/domain, en sus respectivos subsistemas, se excluyeron los nomencladores que se relacionan con las diferentes clases persistentes para lograr en claridad en el diagrama.

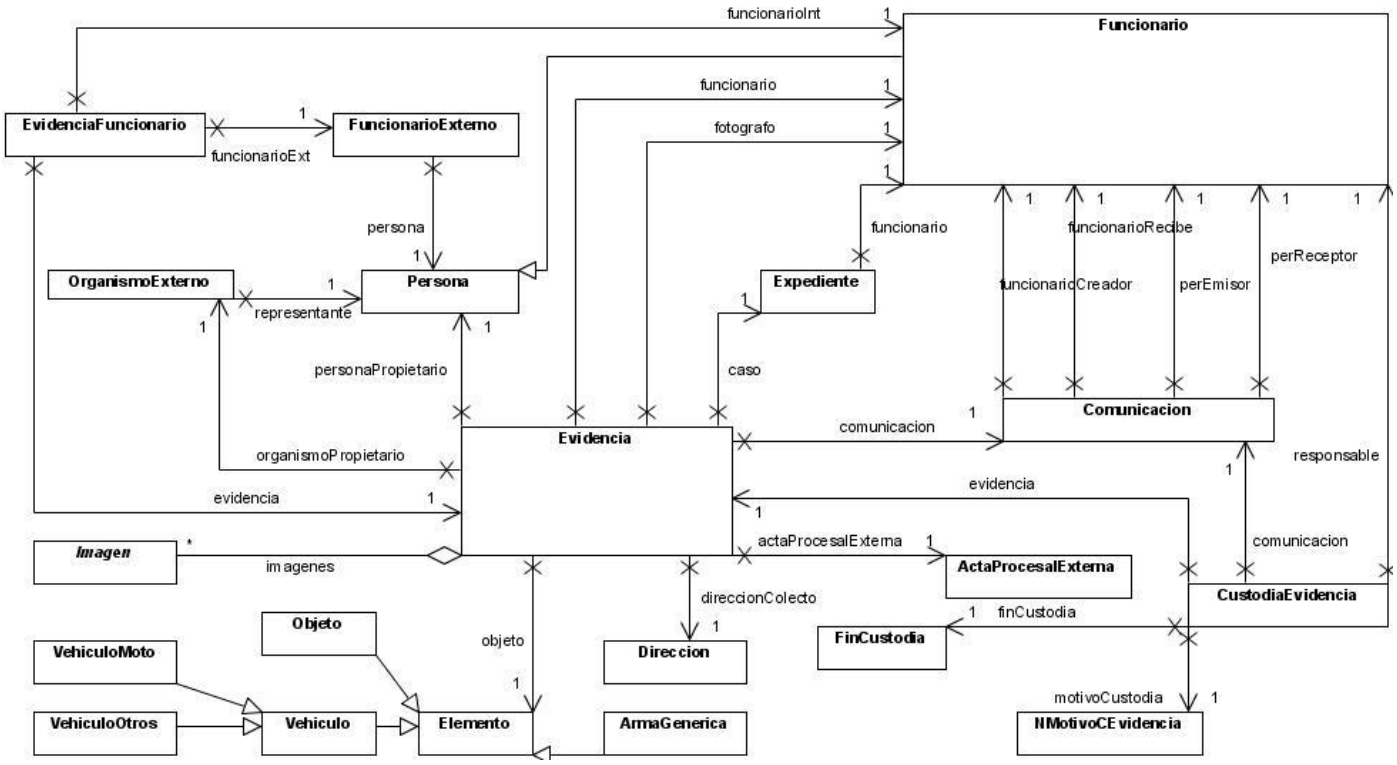


Imagen 2.27: *Diagrama de clases persistentes del Submódulo Evidencias.*

2.3.2. DIAGRAMA DE TABLAS DEL MODELO RELACIONAL

El Modelo Relacional es un modelo en el que la base de datos es vista como una colección de relaciones, una relación puede ser vista como una tabla con sus filas o tuplas y columnas.

El modelo se compone de tres partes:

1. Estructura de datos: básicamente se compone de relaciones.
2. Manipulación de datos: un conjunto de operadores para recuperar, derivar o modificar los datos almacenados.
3. Integridad de datos: una colección de reglas que definen la consistencia de la base de datos

A continuación se muestra el diagrama de tablas del modelo relacional de evidencia con los demás elementos. Se muestra por partes debido al tamaño y proporción del diagrama ya que este presenta todas las tablas que se relacionan con las principales tablas determinadas en el diagrama de clases persistentes del submódulo las que son DEvidencia y DCEvidencia. Para ver el diagrama completo, ver ***Anexo 2***.

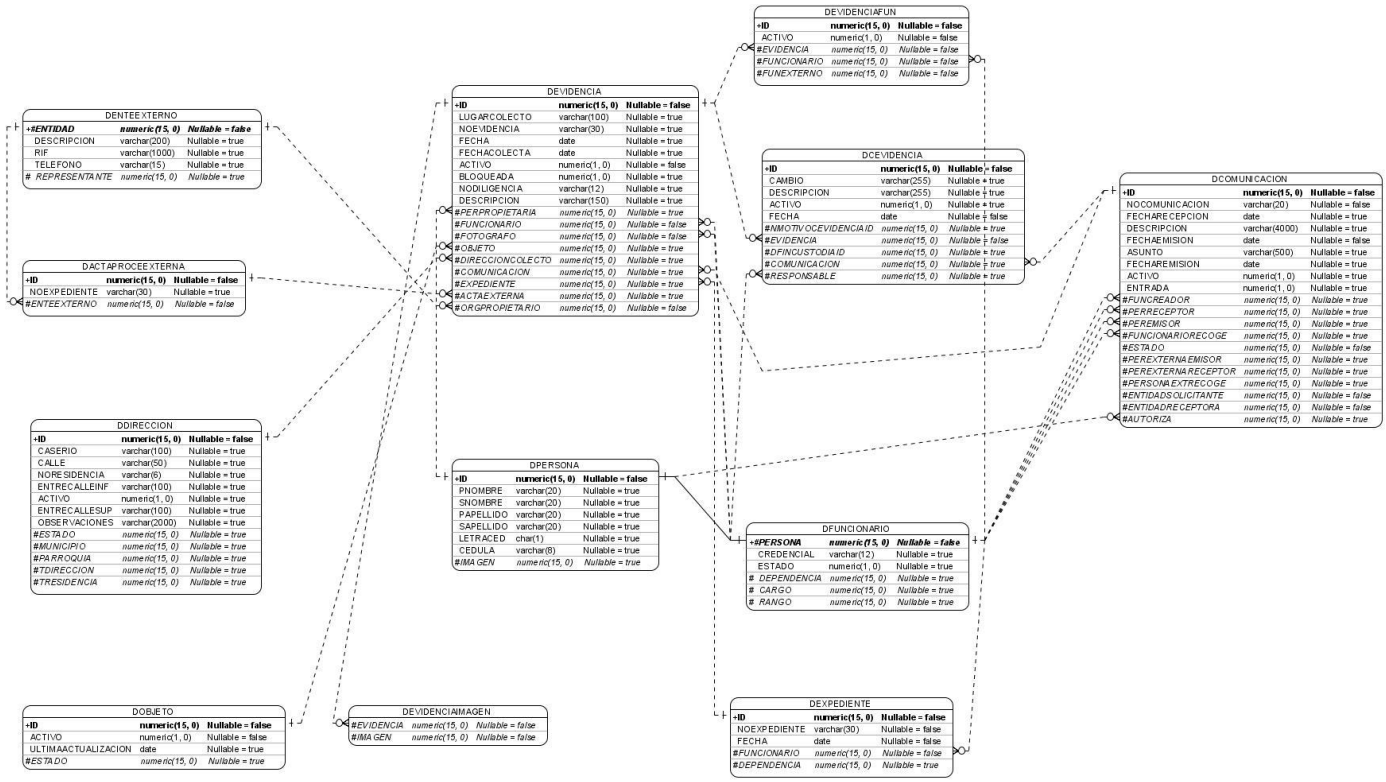


Imagen 2.28: *Diagrama de tablas del Modelo Relacional.*

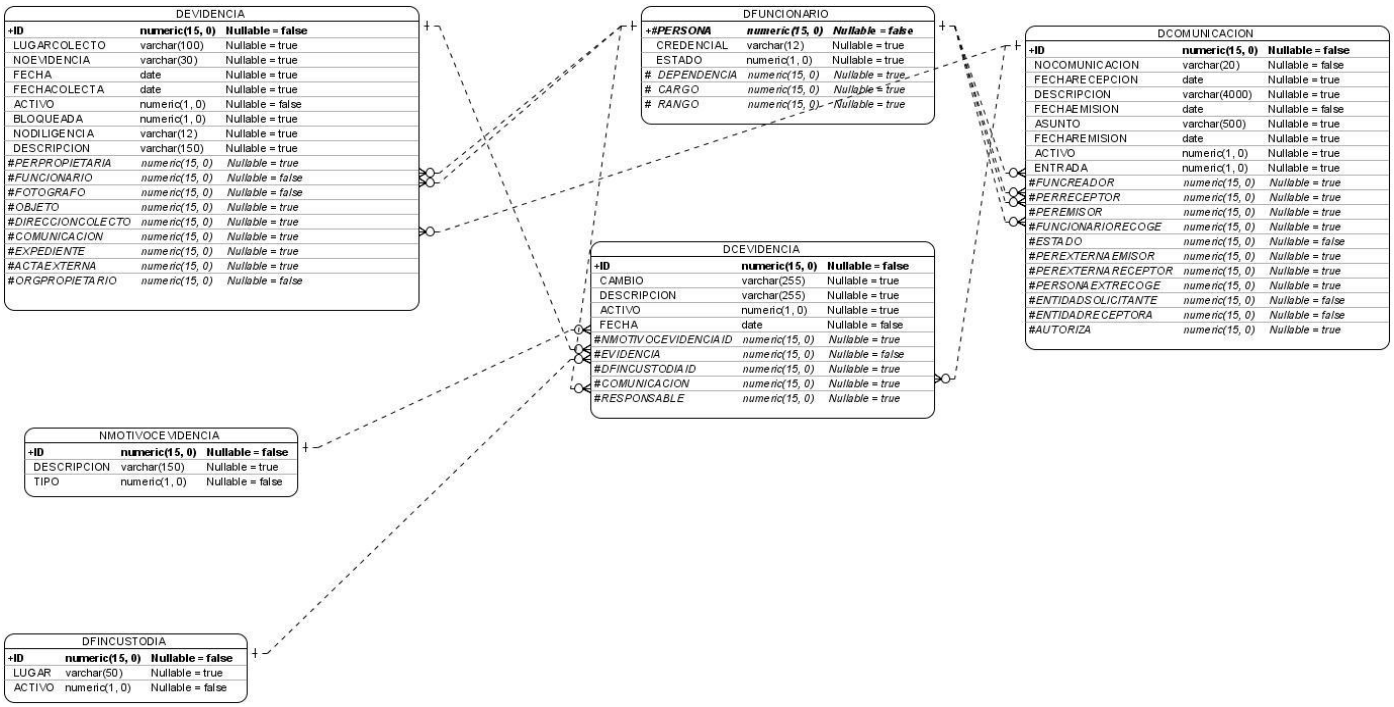


Imagen 2.29: *Diagrama de tablas del Modelo Relacional.*

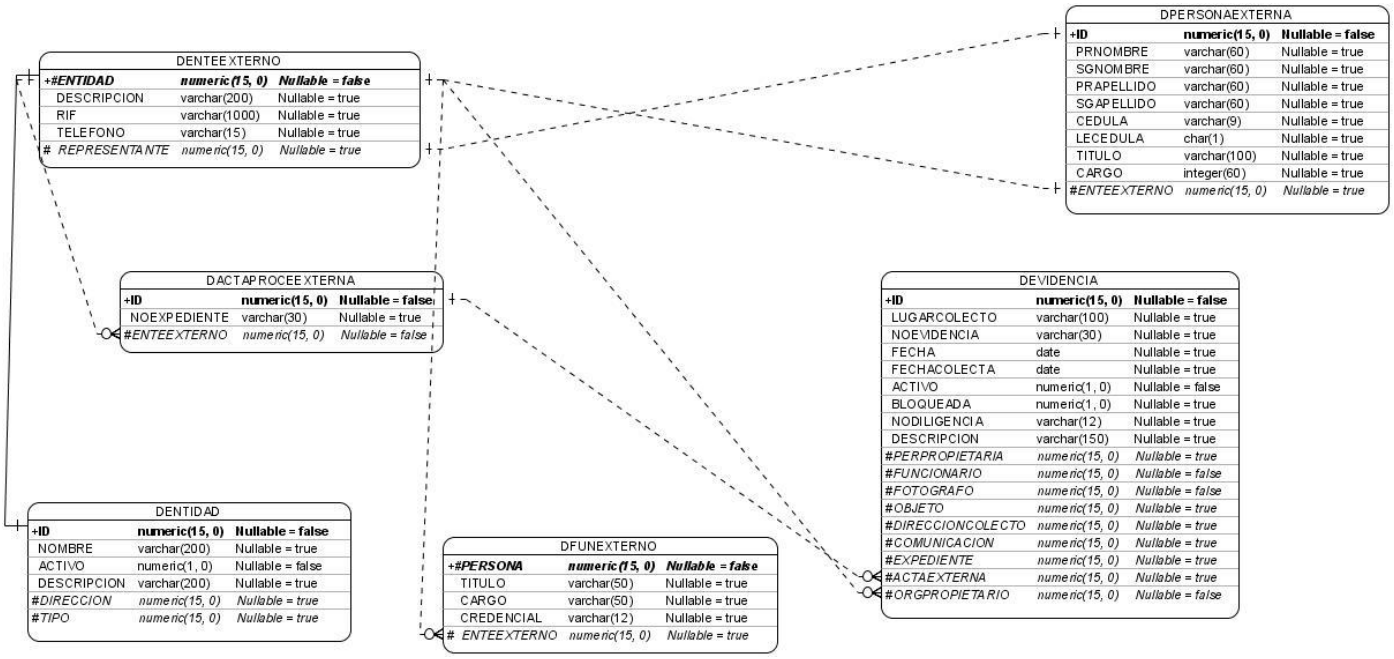


Imagen 2.30: *Diagrama de tablas del Modelo Relacional.*

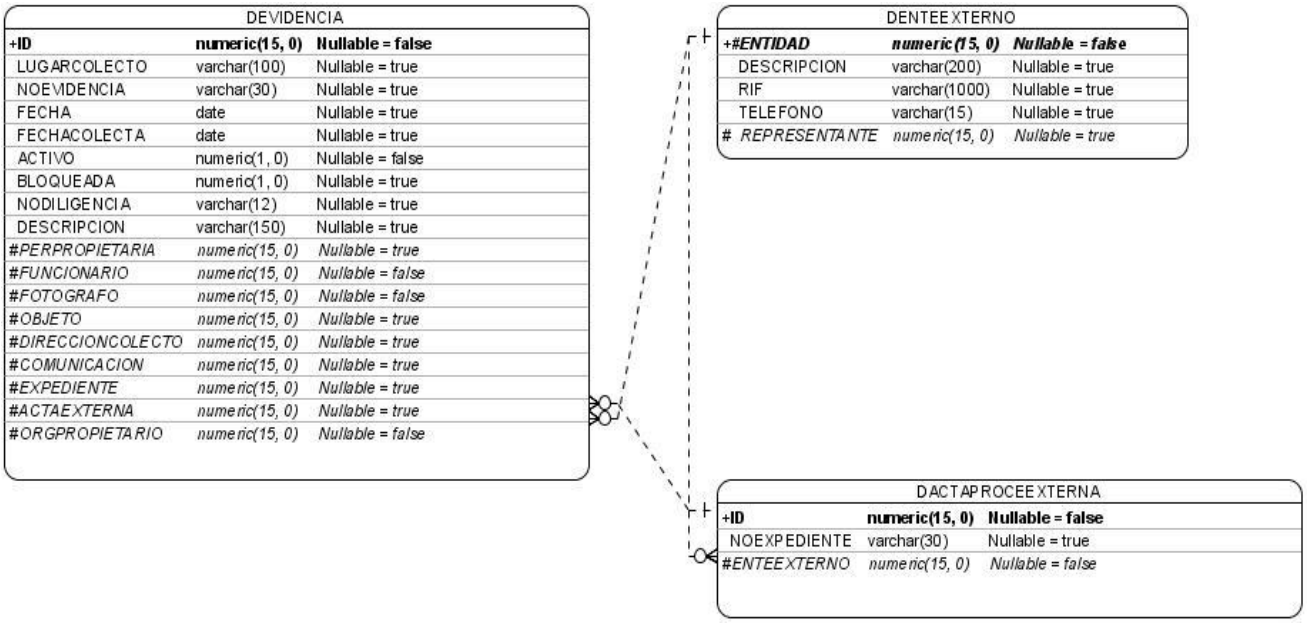


Imagen 2.31: *Diagrama de tablas del Modelo Relacional.*

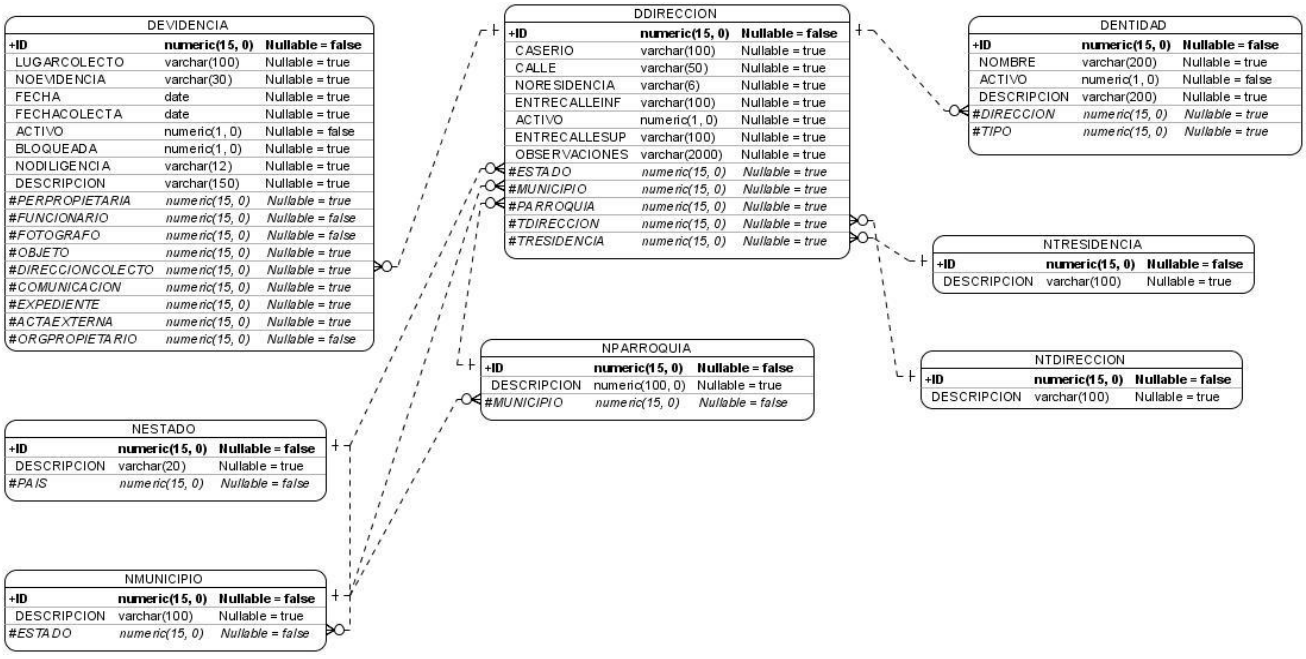


Imagen 2.32: *Diagrama de tablas del Modelo Relacional.*

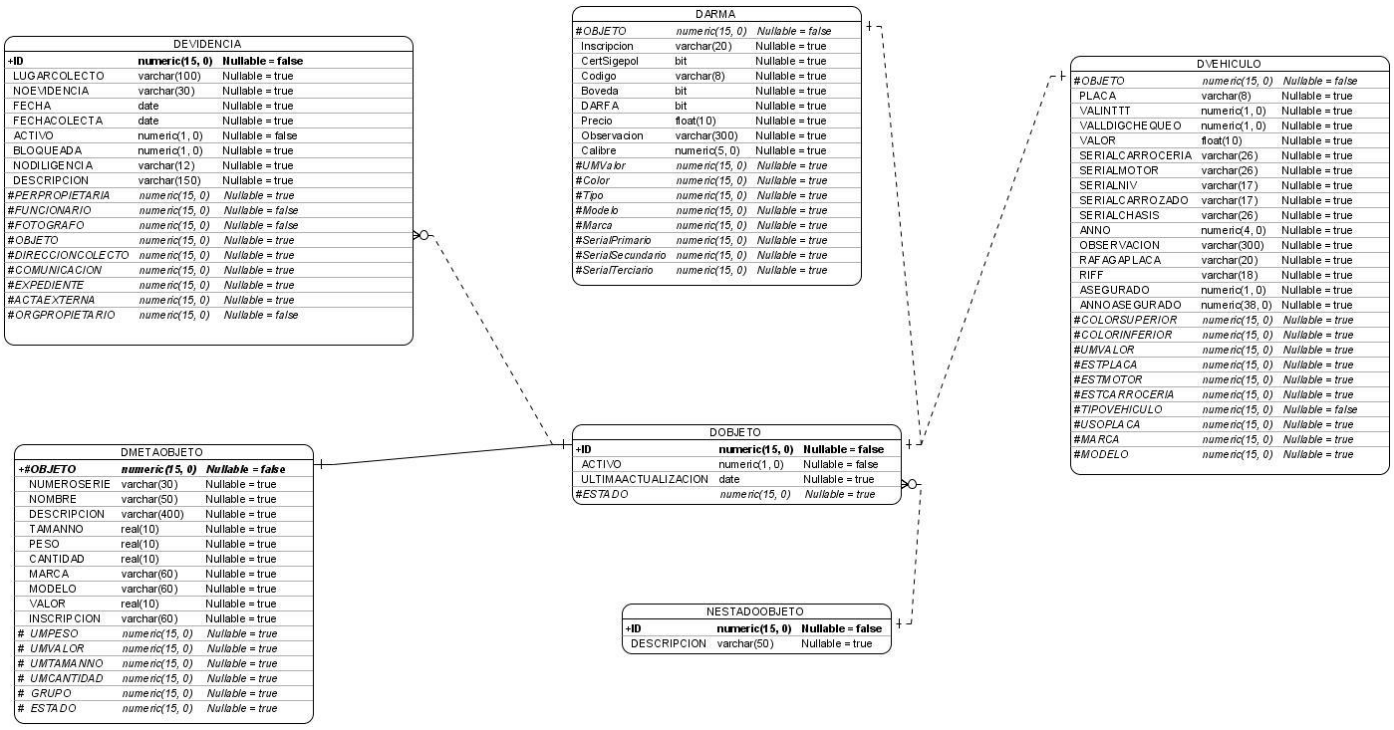


Imagen 2.33: *Diagrama de tablas del Modelo Relacional.*

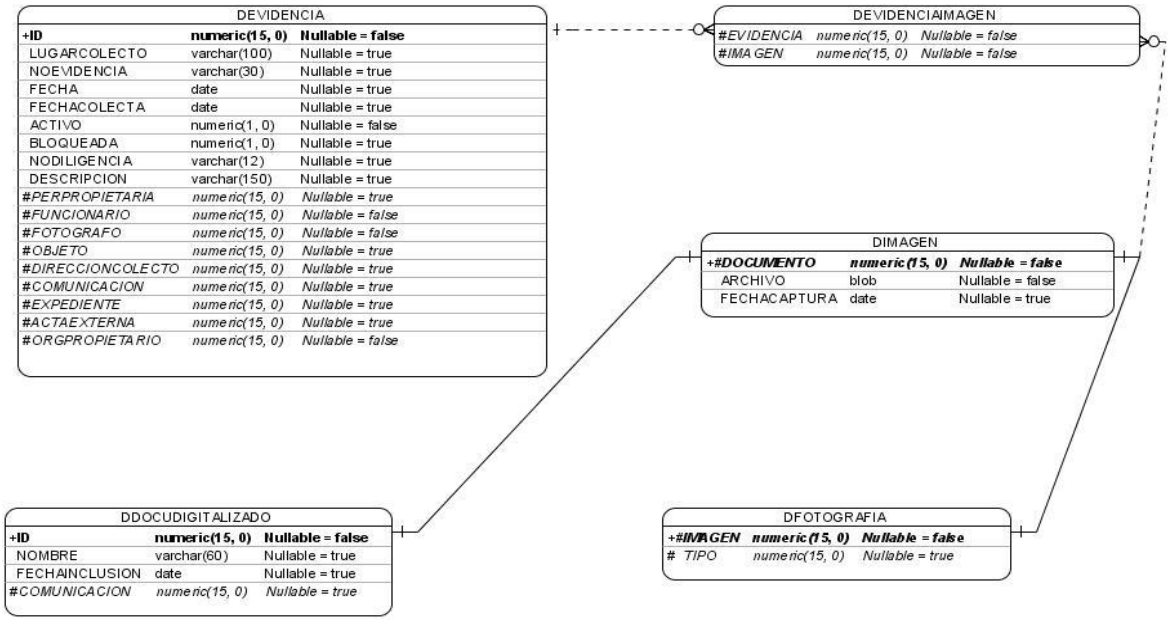


Imagen 2.34: *Diagrama de tablas del Modelo Relacional.*

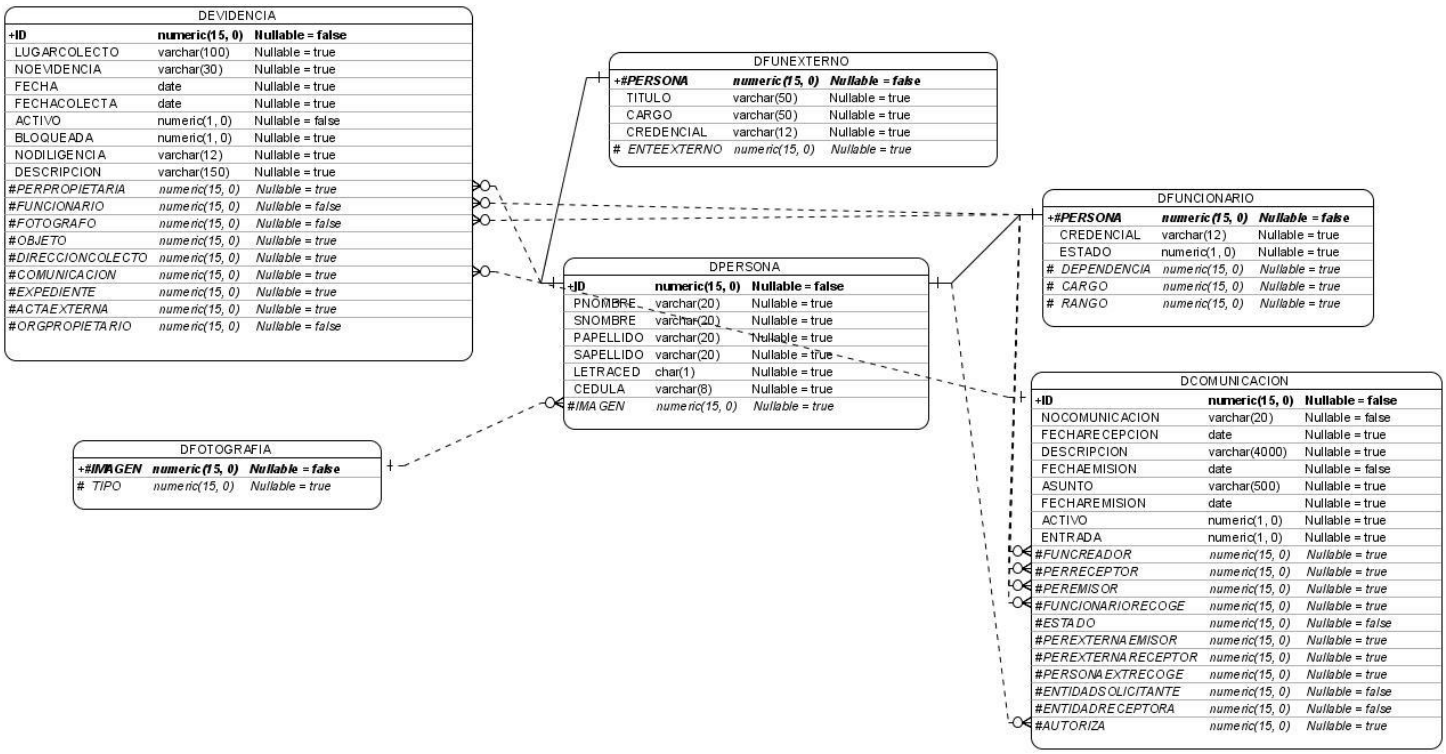


Imagen 2.35: *Diagrama de tablas del Modelo Relacional.*

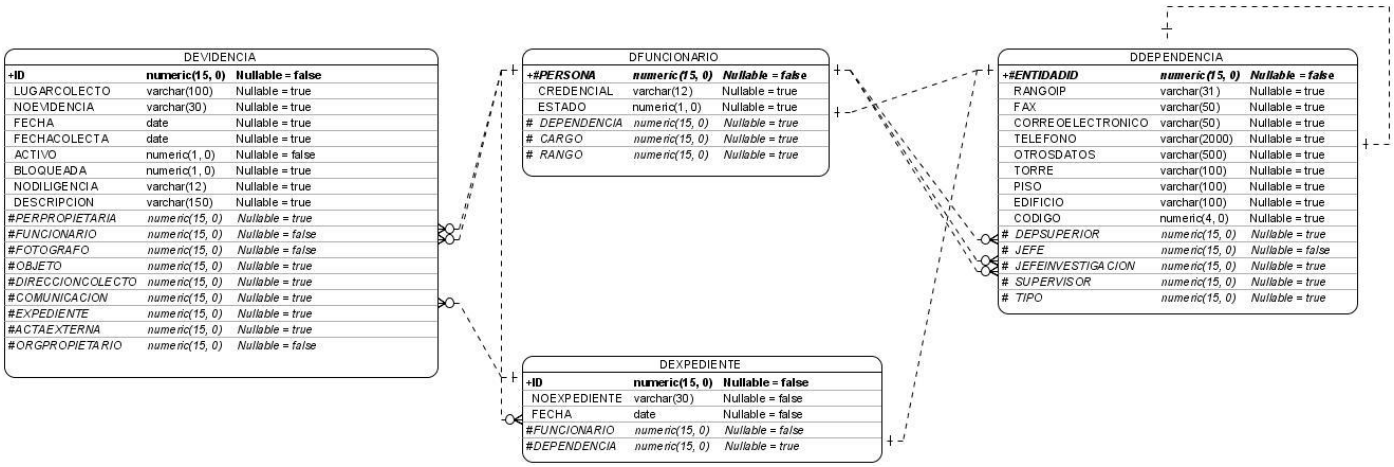


Imagen 2.36: *Diagrama de tablas del Modelo Relacional.*

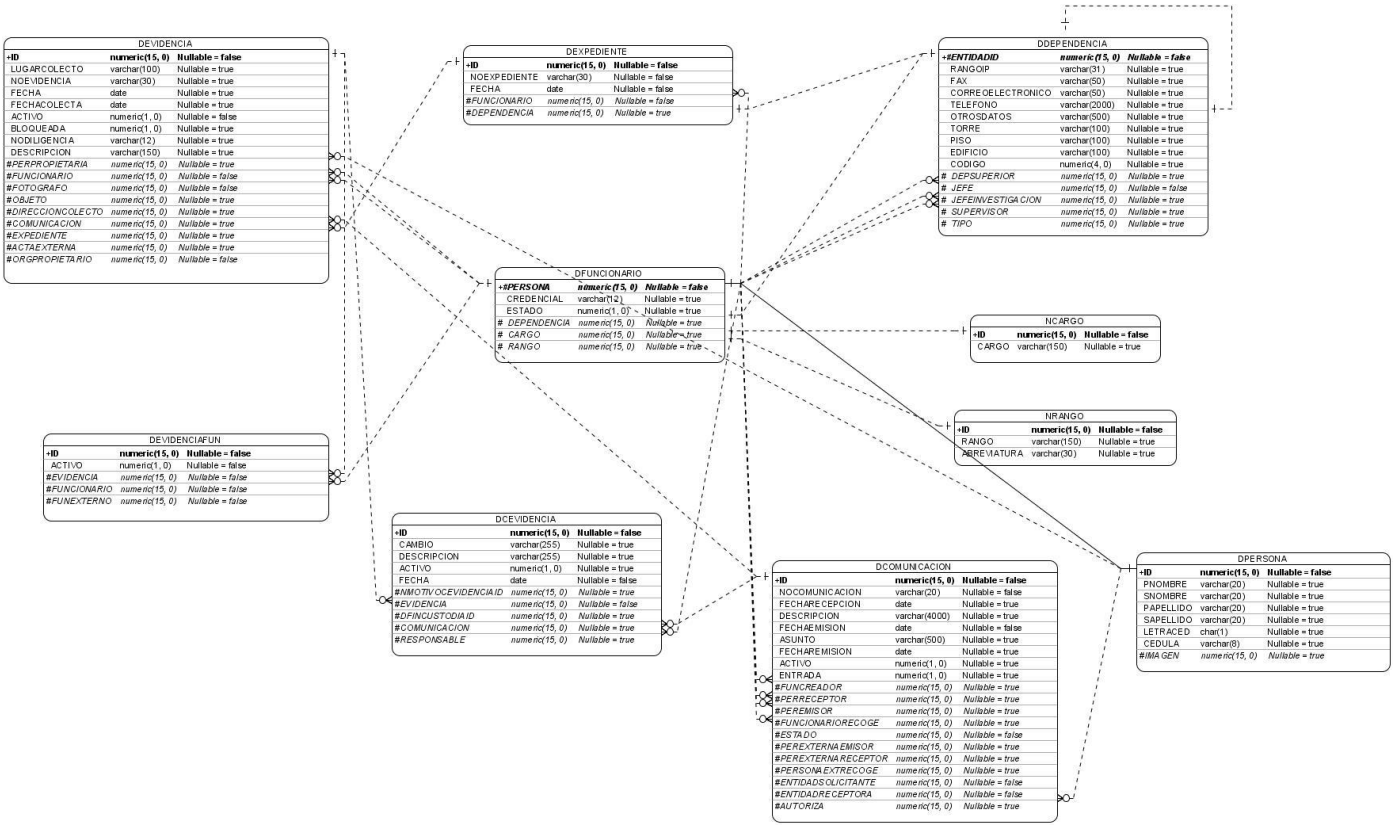


Imagen 2.37: *Diagrama de tablas del Modelo Relacional.*

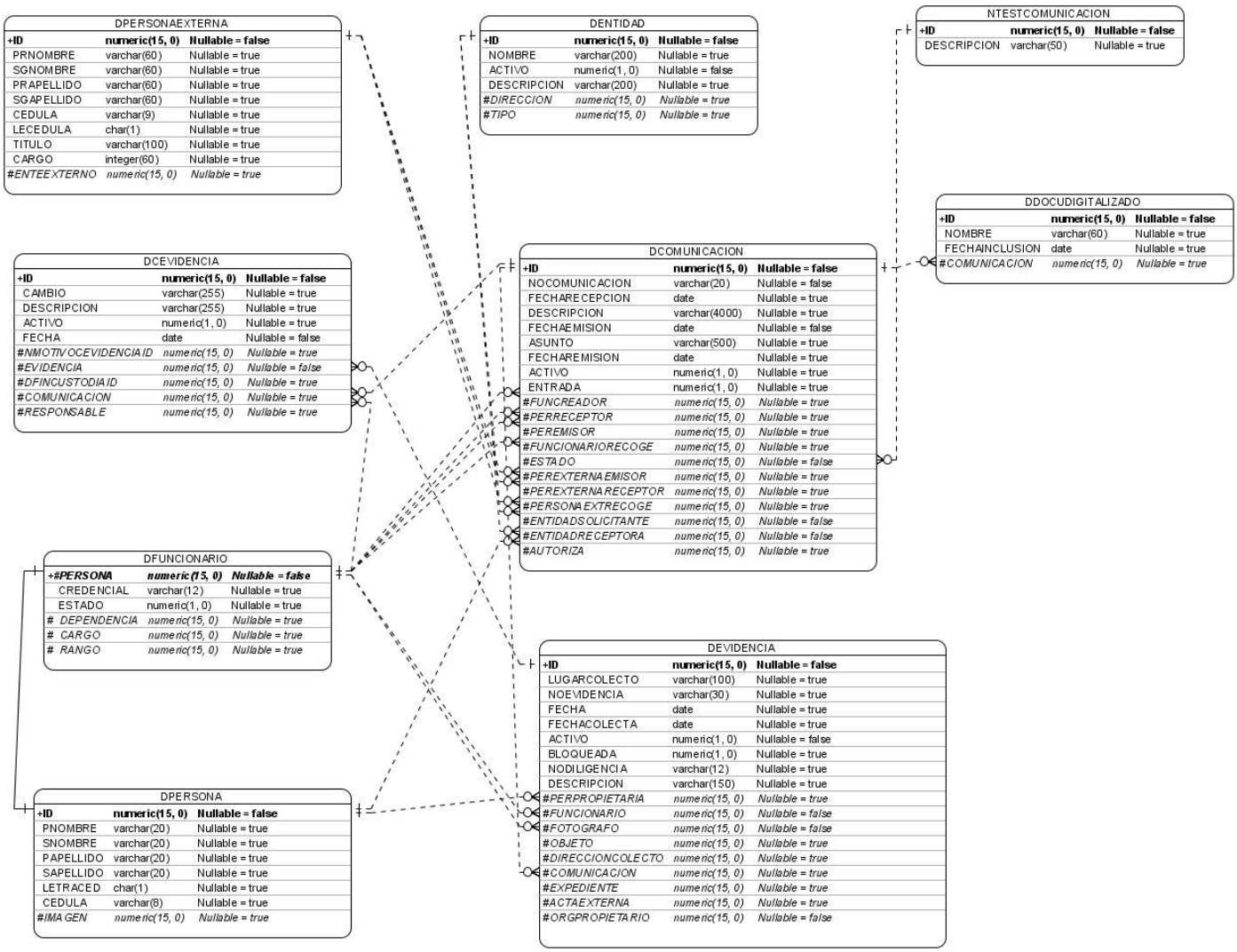


Imagen 2.38: *Diagrama de tablas del Modelo Relacional.*

La implementación en pocas palabras

El flujo de trabajo de implementación describe cómo los elementos del modelo del diseño se implementan en términos de componentes y cómo estos se organizan de acuerdo a los nodos específicos en el modelo de despliegue.

Los elementos y artefactos del flujo de análisis y diseño son el punto de partida para el flujo de trabajo de implementación, en el que se implementa el sistema en términos de componentes, es decir, ficheros de código fuente, scripts, ficheros de código binario, ejecutables y similares; estando fuertemente determinado por el lenguaje de programación.

Los diagramas de despliegue y componentes conforman lo que se conoce como un modelo de implementación al describir los componentes a construir y su organización y dependencia entre nodos físicos en los que funcionará a aplicación. (10)

Cada clase en diseño es implementada a través de su codificación en un lenguaje de programación y/o mediante el uso de componentes pre-existentes.

La correspondencia entre una clase de diseño y su implementación correspondiente dependerá del lenguaje de programación y cada lenguaje ofrece un número distinto de características que pueden ser ajustables en un proceso para un marco de trabajo general, por lo que es importante que se tenga conocimientos del lenguaje de programación para explotar al máximo las posibilidades y manipular las limitaciones del mismo.

En este flujo de trabajo el arquitecto es el responsable de la integridad, corrección, completitud y legibilidad del modelo de implementación de acuerdo a lo descrito en el modelo de diseño. Es responsable también de la arquitectura del modelo de implementación, es decir, de la existencia de sus partes significativas para arquitectónicamente. Un resultado importante de la implementación es la asignación de componentes ejecutables a nodos. El arquitecto es responsable de esta asignación.

Los objetivos del Flujo de Trabajo de implementación son:

- Definir la organización del sistema en términos de Subsistemas de Implementación organizados en capas.
- Implementar los elementos de diseño en términos de "Elementos de Implementación" (ficheros Fuentes, binarios, ejecutables y otros).
- Probar los componentes desarrollados independientemente como unidades.
- Integrar los resultados producidos por desarrolladores independientes o equipos en un sistema ejecutable.

2.4. MODELO DE IMPLEMENTACIÓN

El **Modelo de Implementación** es una visión general de lo que tiene que ser implementado, y un apartado para cada iteración a con los componentes y subsistemas a implementar durante esa iteración, así como de los resultados software que se han de obtener y las pruebas que se ha de realizar sobre ellos.

2.4.1. DIAGRAMAS DE SUBSISTEMAS DE IMPLEMENTACIÓN

Un **Subsistema de Implementación** es una colección de componentes y otros subsistemas de implementación usados para estructurar el modelo de implementación y dividirlos en pequeñas partes que pueden ser integradas y probadas de forma separada.

Los subsistemas de implementación incluyen dependencias y otras informaciones. También podrían incluir modelos claves del subsistema (diagramas de componentes, modelo de despliegue).

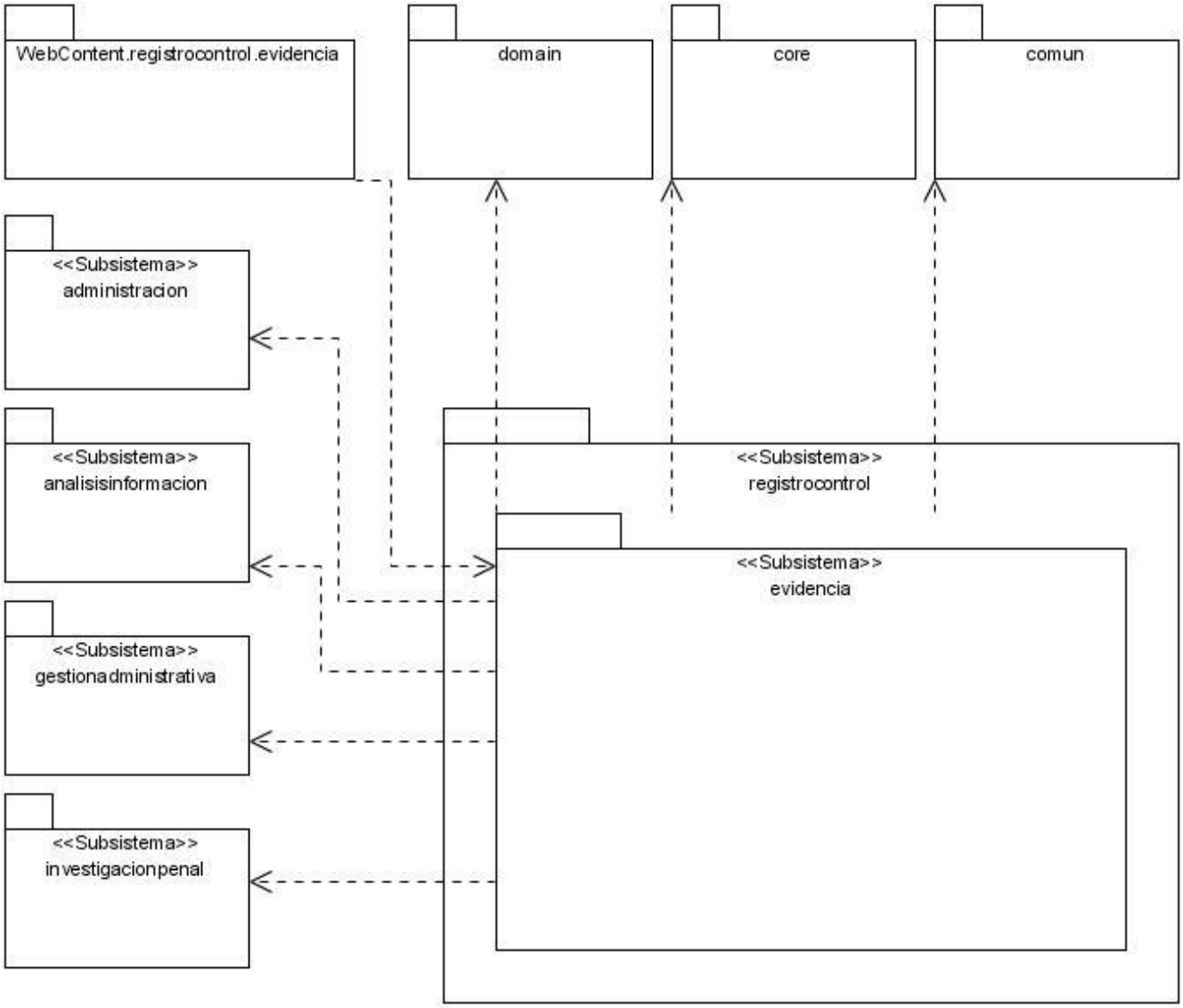


Imagen 2.39: *Diagrama de Subsistemas de Implementación.*

2.4.2. DIAGRAMA DE COMPONENTES

Un **diagrama de componentes** es un diagrama UML que representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. Los componentes físicos incluyen archivos, cabeceras, librerías compartidas, módulos, ejecutables, o paquetes. Muestra la organización y las dependencias entre un conjunto de componentes. Uno de los usos principales es que puede servir para ver qué componentes pueden compartirse entre sistemas o entre diferentes partes de un sistema.

A continuación se muestra el diagrama del subsistema Evidencia y la relación entre los paquetes que lo conforman y para un mayor entendimiento se desglosa cada uno de los paquetes en el diagrama de componentes para un mayor detalle y claridad del mismo.

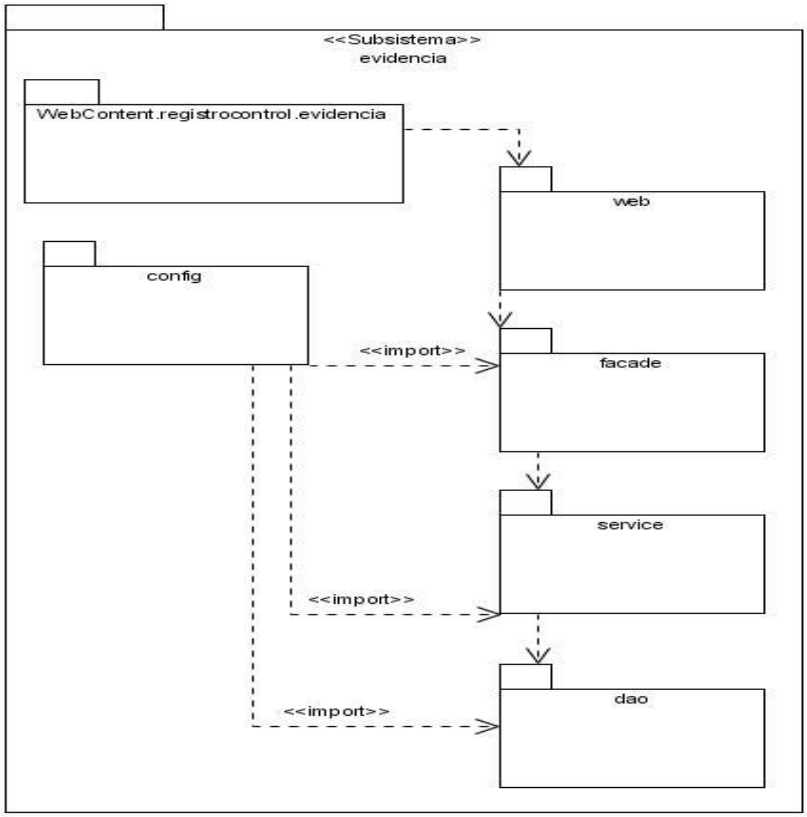


Imagen 2.40: *Diagrama de paquetes del Subsistema de Implementación Evidencias.*

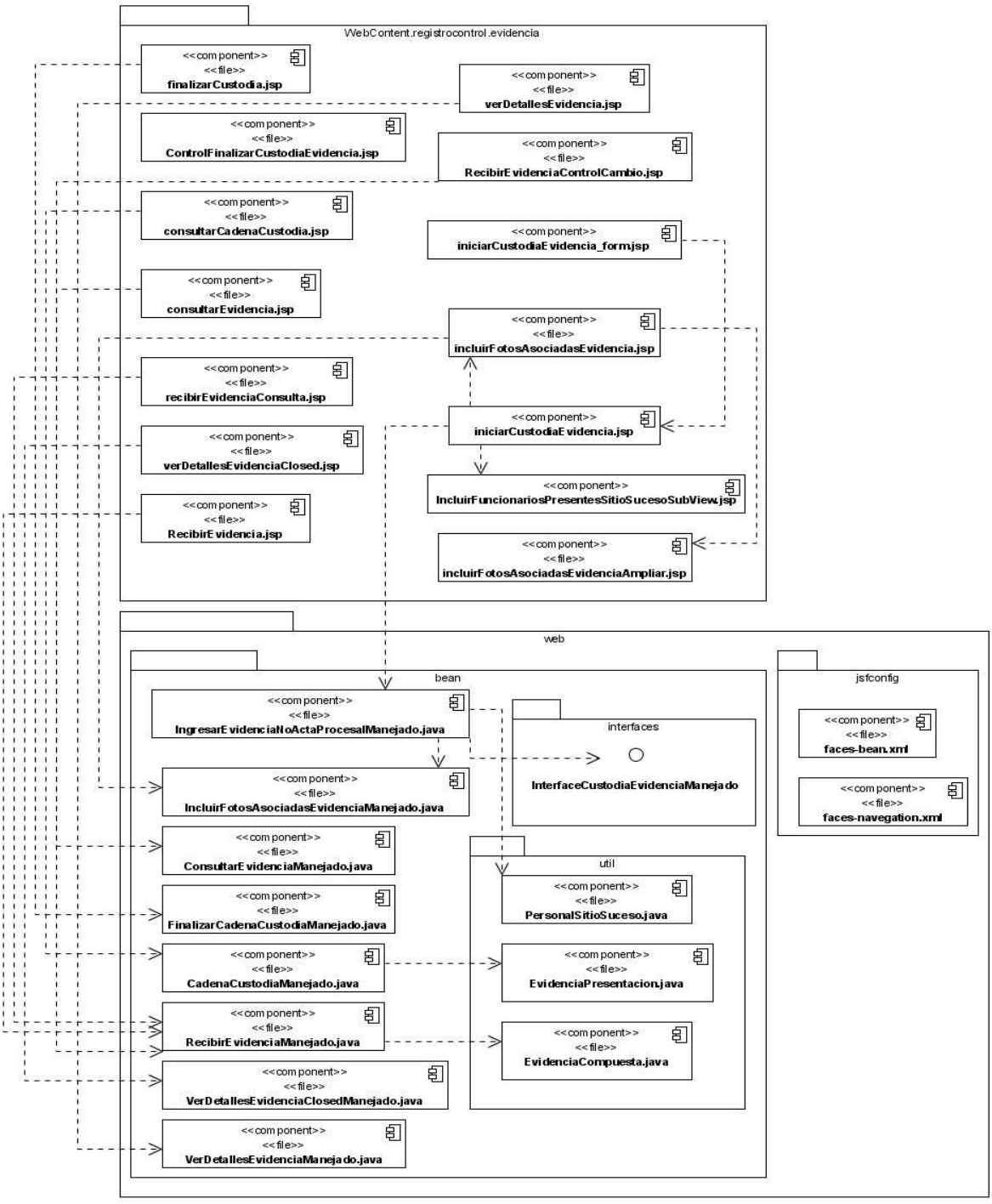


Imagen 2.41: *Diagrama de Componentes de la relación de las páginas con el paquete Web y los beans manejados.*

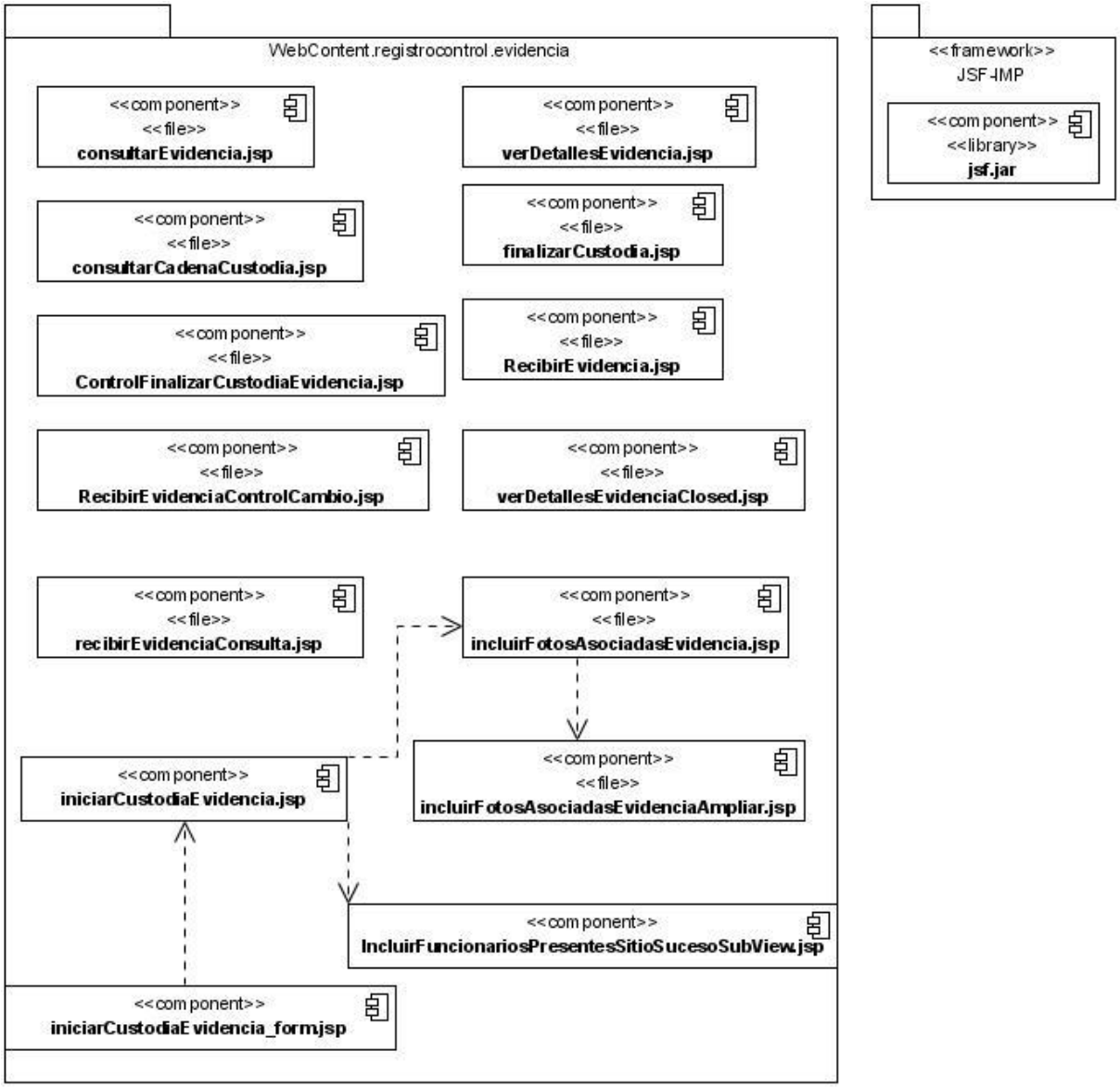


Imagen 2.42: *Diagrama de componentes de la carpeta de las páginas Web en el WebContent.*

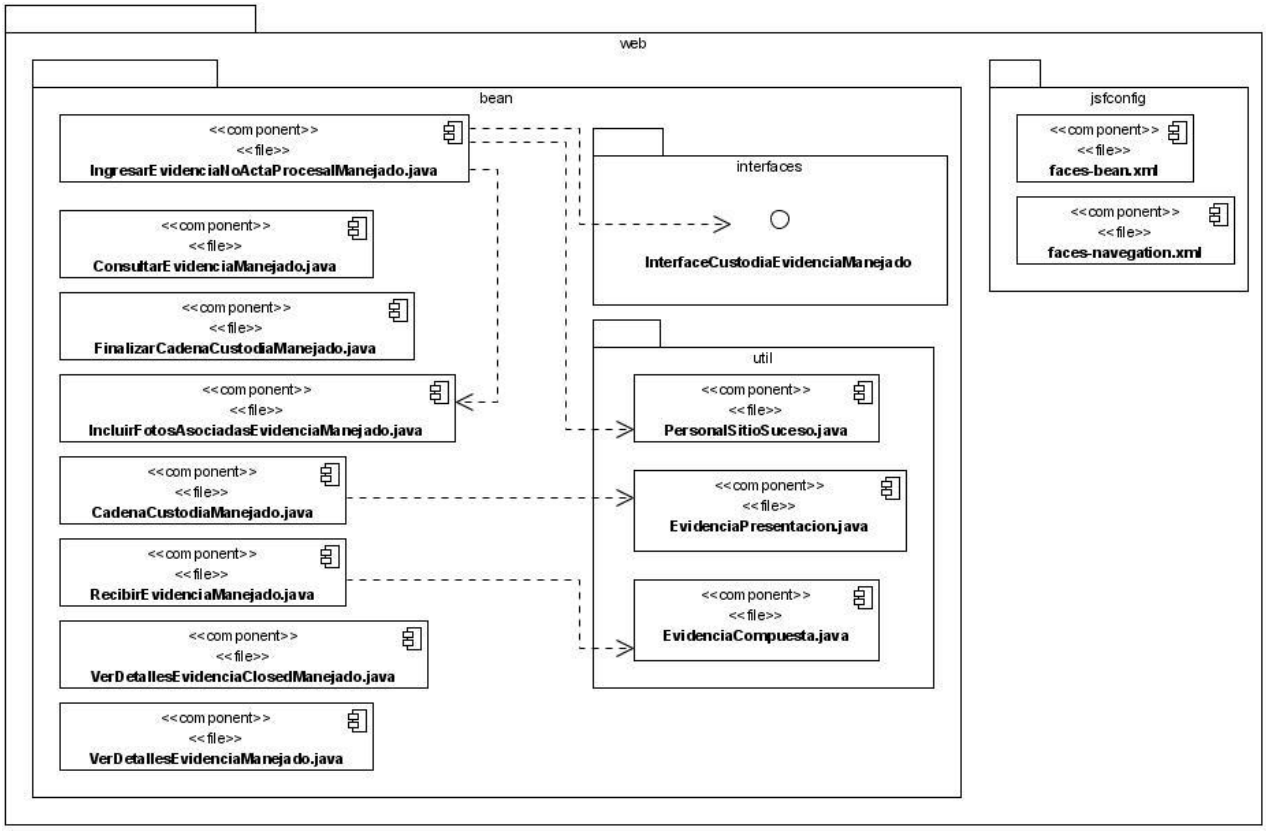


Imagen 2.43: *Diagrama de componentes paquete web y los beans de respaldo.*

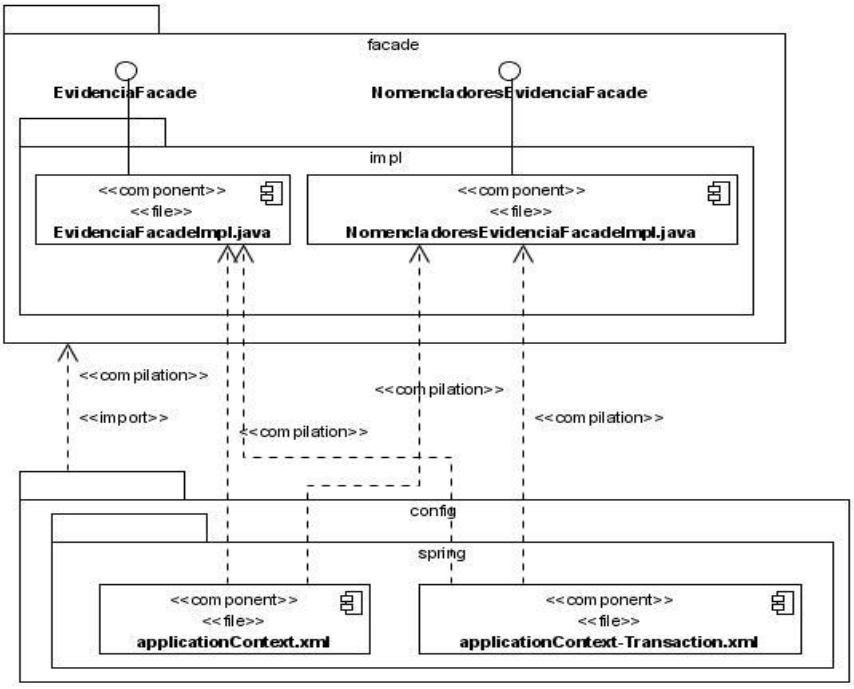


Imagen 2.44: *Diagrama de componentes paquete facade y su relación con el paquete config.*

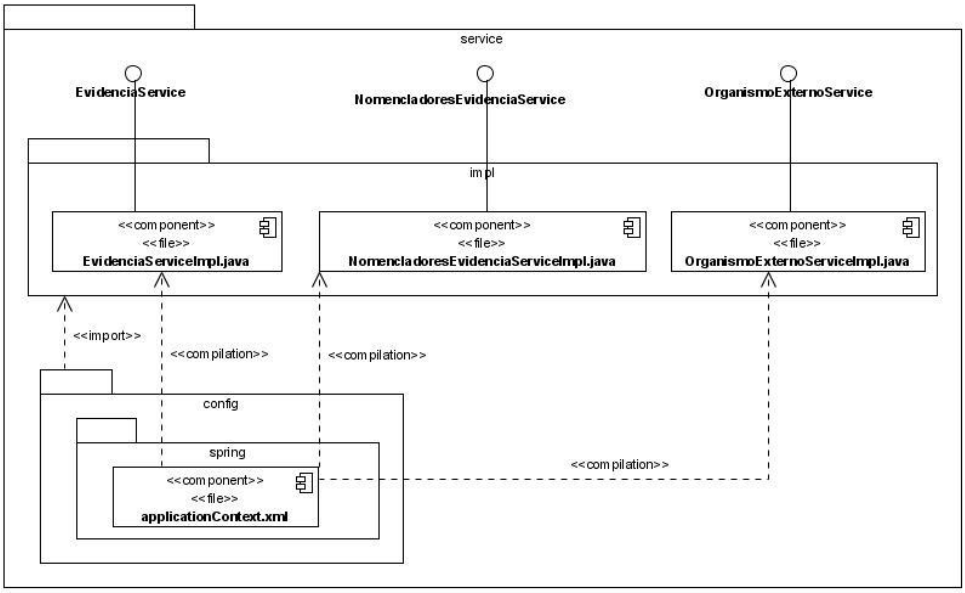


Imagen 2.45: *Diagrama de componentes del paquete service y su relación con el paquete config y los archivos de configuración.*

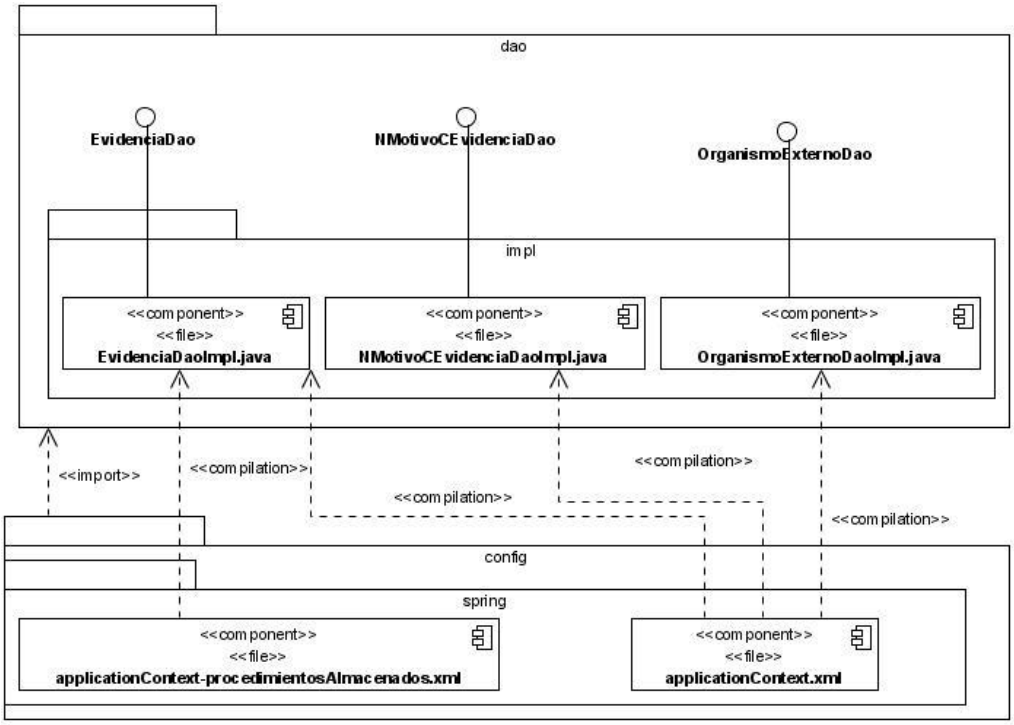


Imagen 2.46: *Diagrama de componentes paquete dao y su relación con los archivos de configuración.*

2.4.3. ESTÁNDAR DE CODIFICACIÓN

La **normalización o estandarización** es la redacción y aprobación de un conjunto de normas que se establecen para garantizar el acoplamiento de elementos construidos independientemente, así como garantizar el repuesto en caso de ser necesario, garantizar la calidad de los elementos fabricados y la seguridad de funcionamiento.

Un **estándar** es una especificación que regula la realización de ciertos procesos o la fabricación de componentes para garantizar la interoperabilidad.

Un **estándar de codificación** comprende todos los aspectos de la generación y escritura de código.

Al comenzar un proyecto de software, es necesario establecer un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada y genérica. La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software, por lo que el mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutinas.

Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento. Además, si se aplica de forma continuada un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas, y, posteriormente, se efectúan revisiones del código de rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

La adopción de un estándar de codificación sólo es viable si se sigue desde el principio hasta el final del proyecto de software. No es práctico, ni prudente, imponer un estándar de codificación una vez iniciado el trabajo de elaboración.

Las técnicas de codificación incorporan muchos aspectos del desarrollo del software. Aunque generalmente no afectan a la funcionalidad de la aplicación, sí contribuyen a una mejor compresión del código fuente. En esta fase se tienen en cuenta todos los tipos de código fuente, incluidos los lenguajes de programación, de marcado o de consulta. Estas no pretenden formar un conjunto inflexible de estándares de codificación, sino que intenta servir de guía en el desarrollo de un estándar de codificación para un proyecto específico de software.

Dentro de los Estándares de Codificación utilizados en desarrollo de la aplicación final se encuentran los elementos establecidos en el documento **Conveciones de código para el lenguaje de programación Java™** establecido por la Sun Microsystems para el desarrollo de aplicaciones utilizando código java y otro conjunto que fueron establecidos por el Arquitecto

Principal en el documento **Guía de estilo de código para el proyecto CICPC** en el cual se establecen entre un grupo de estándares por ejemplo:

Como principio todos los nombres serán en español exceptuando aquellos que correspondan al nombre de un patrón conocido, ejemplo: **ServiceLocator**, **Facade**, **Builder**, entre otros.

Paquetes

Los nombres de todos los paquetes comenzaran por cicpc, este será nuestro prefijo de alto nivel, luego seguirán los nombres de acuerdo a la estructura de paquetes definida por la arquitectura, a saber:

<cicpc>.<nombre del subsistema>.<nombre del módulo>.
<facade\web\config\dao\service\model>

Ejemplo: <cicpc>.<core><config\web\mensajeria\seguridad\util>

Clases

Los nombres de las clases deben ser sustantivos, cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas. Intentar mantener los nombres de las clases simples y descriptivos. Usar palabras completas, evitar acrónimos y abreviaturas (a no ser que la abreviatura sea mucho más conocida que el nombre completo, como URL o HTML).

Convención de nombre para los principales tipos de clases:

1. **Beans manejados de JSF**: NombreManejado.

   Ejemplo: DatosDistribuidoraManejado.

2. **Clases del dominio**: Se les deja su nombre del dominio.

   Ejemplo: Expediente

3. **Validadores de JSF**: NombreValidador.

   Ejemplo: NombreUsuarioValidador.

4. **Convertidores de JSF**: NombreConvertidor.

   Ejemplo: EstudianteConvertidor.

5. **Clases de Fachada**: NombreFacade.

   Ejemplo:ManejadorDistribucionesFacade

6. **Clases que tengan rol de servicio**: NombreService.

   Ejemplo.: CasoService.

7. **Las clases que implementan interfaces**: NombreInterfazImpl.

   Ejemplo: CasoServiceImpl,

8. **Servlets de java**: NombreServlet.

   Ejemplo: EscuchadorOcultoServlet

9. **Filtros de java**: NombreFilter.

   Ejemplo: RegistradorFilter.

Interfaces

Los nombres de las interfaces siguen las mismas reglas que las clases.

Métodos

Los métodos deben ser verbos, cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula.

Ejemplo: enviarSolicitud().

Para más información referente a los estándares de codificación utilizados en el desarrollo, referirse a la documentación especificada anteriormente.

2.5. CONCLUSIONES

En este capítulo se mostraron los artefactos obtenidos durante los diferentes flujos de trabajo desarrollados para el submódulo de Evidencias teniendo como entrada los requisitos funcionales descritos para el mismo. En el modelo de análisis se muestran los diagramas de clases del análisis y las realizaciones de Casos de Uso como punto previo al diseño donde se describieron las clases más importantes con la explicación de algunos de los patrones de diseño utilizados, generando un conjunto de artefactos de vital importancia para luego llegar a la implementación donde se realizó una descripción de los estándares de codificación utilizados y se obtuvo una aplicación ya funcional. Luego de realizar la implementación queda probar si los requisitos previstos para los Casos de Usos definidos se cumplen en la propuesta de solución.

CAPÍTULO 3. VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

3.1. INTRODUCCIÓN

En el presente capítulo se realizará la validación de la propuesta de solución planteada para agrupar y dar respuesta a los Requisitos Funcionales obtenidos en el proceso inicial. Se dará un pequeño bosquejo teórico de los temas a tratar, una explicación de los tipos de pruebas y niveles que se aplican, así como la evaluación de los resultados esperados y obtenidos luego de aplicarle un conjunto de pruebas a la propuesta, para de esta forma fundamentar su correcta implementación y funcionamiento.

Para la validación de la propuesta se realizaron un conjunto de pruebas luego de integrar el submódulo Evidencias al módulo Registro y Control y al SIIPOL.

Las pruebas en pocas palabras

Las pruebas son un conjunto de actividades en las cuales un sistema o componente es ejecutado bajo unas condiciones o requerimientos específicos, donde los resultados son observados y registrados para dar una evaluación de algún aspecto del sistema o componente que es evaluado y determinar la calidad del mismo.

En ellas se describen como comprobar cada versión operacional del sistema durante la integración del mismo, describiendo también como hacer las pruebas al sistema verificando que todos los requerimientos hayan sido implementados, determinado los defectos del mismo.

Es importante aclarar que las pruebas no pueden asegurar la ausencia de defectos; solo pueden demostrar que existen defectos en el software y que cada prototipo que se quiera entregar al final de una iteración debe ser probado y evaluado a diferentes niveles.

Un **Caso de prueba** es un conjunto de entradas de pruebas, condiciones de ejecuciones y resultados esperados desarrollados para cumplir un objetivo en particular o una función esperada. No solo tienen que ser escritos para entradas válidas y esperadas, sino también condiciones no válidas e inesperadas.

Objetivos de las Pruebas:

- Encontrar y documentar los defectos que puedan afectar la calidad del software.
- Validar que el software trabaje como fue diseñado.
- Validar y probar los requisitos que debe cumplir el software.
- Validar que los requisitos fueron implementados correctamente.

TIPOS DE PRUEBAS

Pruebas de Caja Blanca

La prueba de **Caja Blanca** del software comprueba los caminos lógicos dentro de este, proponiendo casos de prueba que se ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coinciden con el esperado o mencionado. Ellas requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas de diseño o el código.

Mediante los métodos de prueba de la caja blanca, el ingeniero de software puede obtener casos de prueba que garanticen que:

1) Se ejerciten por lo menos una vez todos los caminos independientes para cada módulo.
2) Se ejerciten todas las decisiones lógicas en sus vertientes verdaderas y falsa.
3) Ejecuten todos los bucles en sus límites y con sus límites operacionales.

4) Se ejerciten las estructuras internas de datos para asegurar su validez.
5) Se obtengan los resultados esperados luego de ejecutar una funcionalidad dada.

Pruebas de la Caja Negra

En las pruebas de **Caja Negra** los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene.

Los métodos de prueba de caja negra son apropiados para los sistemas orientados a objetos. Los casos de uso brindan datos de entrada muy útiles en el diseño de pruebas de caja negra y basada en estados. Las pruebas de caja negra además:

- Verifican las especificaciones funcionales y no consideran la estructura interna del programa.
- Es hecha sin el conocimiento interno del producto.
- No validan funciones ocultas (por ejemplo funciones implementadas pero no descritas en las especificaciones funcionales del diseño) por tanto los errores asociados a ellas no serán encontrados.

En otras palabras, la prueba de la caja negra se refiere a las pruebas que se llevan a cabo sobre la interfaz del software, o sea los casos de prueba pretenden:

1) Demostrar las funciones del software son operativas.
2) Que las entradas se aceptan de la forma adecuada y que se produce el resultado correcto.
3) Determinar errores en la interfaz y las funciones visibles al cliente.

Dado las alternativas de la caja negra y la caja blanca; y de la necesidad de que las pruebas abarquen los requerimientos, las funciones y la lógica interna del programa entonces se utilizaron

para la evaluación de los requerimientos se la estrategia de la caja negra y para la lógica interna se empleó la estrategia de la caja blanca.

NIVELES DE PRUEBAS

Nivel de Unidad: Enfocada al código fuente de los componentes. Son utilizadas para verificar todos los flujos de control realizados dentro de la aplicación y primero pasa por la revisión del programador.

Nivel de Integración: En este nivel se prueban los componentes combinados para ejecutar un Caso de Uso para verificar descubrir errores o incompletitud en las especificaciones de las interfaces de las clases y las funcionalidades que ellas brindan.

Nivel de Sistema: Nivel en el que se prueba el software funcionando como un todo. Estas son aceptables para cuando el software se encuentra en la Fase de Construcción principalmente.

Nivel de Aceptación: Prueba final antes del despliegue de la aplicación final para su uso comercial. Estas generalmente lo realizan los usuarios finales y comúnmente es denominada "Prueba Piloto".

Prueba independiente: Pruebas realizadas por el programador en el proceso de desarrollo, aplicándolas a las funcionalidades creadas y el código generado durante las diferentes iteraciones.

Para la realización de las pruebas de caja blanca se utilizó el framework JUnit soportado por java.

JUnit es un framework que se utiliza en programación para hacer pruebas unitarias de aplicaciones Java. Permite realizar la ejecución de clases Java de manera controlada y evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta de la forma esperada (11). Este, en función de algún valor de entrada, evalúa el valor de retorno esperado; y si la clase

cumple con la especificación, devuelve que pasó exitosamente la prueba; en caso contrario, devolverá un fallo en el método correspondiente. Es también un medio de controlar los nuevos resultados cuando una parte del código ha sido modificado y se desea saber si el nuevo código cumple con los requerimientos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación. El propio framework incluye formas de ver los resultados de las pruebas realizadas.

El framework de utilizó para probar los resultados de las funcionalidades implementadas en la fachada del submódulo de Evidencias denominada EvidenciaFacade para corroborar los resultados que se necesitan de cada una. Para más información sobre las pruebas unitarias realizadas a la clase de fachada (ver ***Anexo 3***).

Posterior a su implementación, el submódulo de Evidencias ha pasado por diferentes iteraciones del flujo de trabajo de Pruebas definido por RUP y se han corregido un conjunto de errores existentes, dando el visto bueno por parte del equipo de trabajo de control de la calidad, confirmando el nivel de calidad y el correcto funcionamiento de los requisitos funcionales y no funcionales determinados para el submódulo y el software en general. La especificación de los casos de prueba a aplicar fue elaborada por dicho equipo de aseguramiento de la calidad. Ello previno que en posteriores iteraciones de Calidad a otros niveles se detectaran errores simples y fallas de riesgo para el correcto funcionamiento del sub-módulo y sistema en general.

Iteración de Prueba	Total No Conformidades SIIPOL	No Conformidades de Registro y Control	No Conformidades del submódulo Evidencias	% Evidencia/total
Pruebas Internas 2da Etapa 1ra Iteración	667	55	8	1
Pruebas Calidad Interna 2da Entrega	618	106	60	10
Calidad UCI SIIPOL 1ra Etapa 4ta Iteración	3	1	0	0
Pruebas Cruzadas al SIIPOL V2.0	357	23	8	2

Tabla No. 4 *No Conformidades referentes a las Pruebas Internas realizadas.*

La descripción textual de los Casos de Prueba utilizados se encuentra dentro de la documentación generada por el proyecto, para más información dirigirse a la misma.

Se realizaron 4 iteraciones de pruebas de aceptación y una Piloto del software SIIPOL y al submódulo de Evidencias como parte de este, las cuales contaron con la presencia del cliente

que interactuó directamente con la aplicación, de las cuales se arrojaron los siguientes resultados:

Iteración de Prueba	Total No Conformidades	No Conformidades de Registro y Control	No Conformidades del submódulo Evidencias	Pedidos de Cambio para el submódulo de Evidencias
Aceptación 1.5	42	12	6	2
Aceptación 2.0	23	7	0	-
Aceptación 2.01	15	0	0	-
Aceptación 2.02	25	0	0	-
Pruebas Piloto	78	10	3	-

Tabla No. 5 *No conformidades detectadas en pruebas con la presencia del cliente.*

Como se ha podido constatar con las iteraciones de pruebas desarrolladas, el nivel de errores y no conformidades detectadas por parte del cliente fue muy bajo, con un alto nivel de aceptación de la implementación del submódulo y el software en general, verificando así que fueron

desarrollados de forma correcta los requisitos determinados en la etapa inicial para satisfacer las necesidades requeridas. Los pedidos de cambio formulados por el cliente solo fueron en la primera iteración de las pruebas aplicadas, lo de demuestra la calidad del trabajo realizado y el acercamiento a la respuesta de las necesidades detectadas.

3.2. CONCLUSIONES

En el desarrollo del capítulo se pudo evaluar la propuesta de solución mediante la aplicación de los diferentes tipos de pruebas utilizadas para la validación de la propuesta, luego de explicar un conjunto de conceptos asociados a los temas tratados. Luego de realizadas las pruebas al submódulo Evidencias, inmediatamente de ser integrado al software SIIPOL, se analizaron los resultados obtenidos de su aplicación constatándose el nivel de aceptación y calidad requeridos por el cliente el correcto despliegue y puesta en marcha del software, para así mejorar el nivel de respuesta a las actividades delictivas.

CONCLUSIONES

La presente investigación recoge el proceso de desarrollo del submódulo de Evidencias perteneciente al módulo Registro y Control del nuevo SIIPOL iniciada con un análisis de las diferentes tecnologías adoptadas para el desarrollo de aplicaciones empresariales de alta calidad.

Los procesos de análisis, diseño e implementación del submódulo Evidencias perteneciente al módulo Registro y Control para el nuevo SIIPOL se realizaron de forma satisfactoria, lo que quedó demostrado con la validación de la propuesta realizada, de la que se obtuvieron las siguientes conclusiones:

- ✓ Se logró una alta familiarización con los procesos que a automatizar y los conceptos manejados por los desarrolladores.
- ✓ Se le ha dado solución a los requisitos funcionales y no funcionales que se obtuvieron como resultado de aplicar la Ingeniería de Requerimientos al Submódulo en cuestión.
- ✓ La utilización de los frameworks determinados por el equipo de Arquitectura permitieron un desarrollo rápido a pesar de la alta complejidad presentada por el software en general.
- ✓ Se generaron los artefactos necesarios para realizar una correcta documentación del software, como parte de la aplicación de RUP como metodología de desarrollo para la guía y control de todo proceso.
- ✓ Se logró la integración satisfactoria del submódulo de Evidencias al módulo Registro y Control a y al SIIPOL, obteniéndose como resultado una aplicación funcional y operativa, llevando a cabo buenas prácticas de diseño e implementación, respondiendo a los objetivos propuestos como cumplimiento de las actividades planificadas.

- ✓ La aplicación de diversas iteraciones de pruebas al submódulo y al software en general permitieron la entrega de una aplicación funcional y de calidad corroborada, estando el cliente conforme con el producto final.
- ✓ El submódulo de Evidencias integrado al SIIPOL aportará grandes beneficios a los funcionarios del CICPC, a los órganos judiciales y criminalísticos que utilizan y procesan las evidencias en la República Bolivariana de Venezuela.

RECOMENDACIONES

Se recomienda:

- ✓ Realizar las refactorizaciones necesarias para cumplir con los objetivos trazados por los arquitectos y mejorar el código y la aplicación en general.
- ✓ Realizar otras pruebas de unidad a los elementos no probados o funcionalidades incluidas en el proceso de refactorización.
- ✓ Realizar un proceso investigativo a nivel nacional para determinar de de ser posible el desarrollo de una herramienta informática similar que apoye los procesos desarrollados por la policía nacional que brinde apoyo a cada uno de los departamentos que se distribuye la misma, aportando a la sociedad de forma positiva y ganar en experiencia con relación a los sistemas policiales u otros sistemas importantes para la informatización de la sociedad.
- ✓ Utilización de la información generada y la documentación para proyectos futuros similares, siempre cumpliendo con las normas y políticas de confidencialidad requeridos.

BIBLIOGRAFÍA

1. **Rivero Guevara, Humberto.** Trabajo de Diploma: Analisis, Diseño e Implementación del Modulo Aprehensión del SIIPOL. Ciudad Habana : UCI, 2008.

2. **Pereira Ojeda, Maikel y Gago Martinez, Yudanis.** Trabajo de Diploma: Análisis, Diseño e Implementación del módulo Experticias Criminalísticas del Sistema de Investigación e Información Policial. Ciudad Habana : UCI, 2008.

3. **Group, Object Management.** UML Resource Page. *sitio web Unified Modeling Language.* [En línea] Object Management Group, Inc, 2008. [Citado el: 9 de diciembre de 2008.] http://www.uml.org/.

4. **Álvarez Marañón, Gonzalo.** ¿Qué es Java? *Sitio oficial de la IEC.* [En línea] 1999. [Citado el: 4 de Noviembre de 2008.] http://www.iec.csic.es/CRIPTONOMICON/java/quesjava.html#top.

5. **IBM Corporation & Sun Microsystems, Inc.** Eclipse Platform Technical Overview. *sitio web oficial comunidad de desarrollo Eclipse.* [En línea] Febrero de 2003. [Citado el: 27 de Noviembre de 2008.] http://www.eclipse.org/whitepapers/eclipse-overview.pdf.

6. **Browne, Paul.** O´Relly on Java.com. The independent Sourse for Enterprise Java. *sitio web O´Relly on Java.com.* [En línea] O'Reilly Media, Inc., 2007. [Citado el: 12 de Noviembre de 2008.] http://www.oreillynet.com/onjava/blog/2007/09/red_hat_developer_studio_good.html.

7. Visual Paradigm. *sitio oficial Visual Paradigm.* [En línea] 2008. [Citado el: 29 de noviembre de 2008.] http://www.visual-paradigm.com.

8. **Lago, Ramiro.** Introducción a JSF. [En línea] Mayo de 2007. [Citado el: 30 de Octubre de 2008.] http://www.proactiva-calidad.com/java/jsf/introduccion.html.

9. **Equipo de desarrollo Acegi Security.** Acegi Security System for Spring. *sitio oficial Acegi Security System for Spring.* [En línea] Interface21, Inc, 2008. [Citado el: 11 de Diciembre de 2009.] http://www.acegisecurity.org/.

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

11. **Comunidad de Desarrolladores Junit.** sitio web oficial Junit.org. *JUnit.org Resources for Test Driven Development.* [En línea] 2009. [Citado el: 2 de abril de 2009.] http://www.junit.org/.

12. **IEEE.** IEEE. *sitio oficial de la IEEE.* [En línea] Copyright IEEE , 2009. [Citado el: 27 de Enero de 2009.] http://www.ieee.org/portal/site.

13. **Jacomson, Ivar, Booch, Grady y Rumbaugh, James.** *El proceso unificado de desarrollo de Software.* s.l. : Editorial Felix Varela, 1999.

14. **Iborra, Ignacio.** *JUnit.* [En línea] Departamento de Ciencia de la Computación e Inteligencia Artificial, Universidad de Alicante, 2004. [Citado el: marzo de 30 de 2009.] http://www.jtech.ua.es/tutoriales/apuntes/sesion-junit-apuntes.htm.

15. Ajax4jsf Developer Guide. *sitio Web Ajax4jsf Developer Guide.* [En línea] 2006. [Citado el: 10 de diciembre de 2008.] https://ajax4jsf.dev.java.net/nonav/documentation/ajax-documentation/index.html.

16. **Bauer, Christian y King, Gavin.** *Hibernate in Action.* Greenwich : Manning Publications Co., 2005. ISBN 1932394-15-X.

17. **Breidenbach, Craig Walls y Ryan.** *Spring in Action. Secon Edition.* Greenwick : Manning Publications Co., 2008. ISBN 1-933988-13-4.

18. **Mann, Kito D.** *JavaServer Faces in Action.* Greenwich : Manning Publications Co., 2005. ISBN 1-932394-11-7.

19. **Microsystems, Sun.** Sun Microsystems. *sitio Oficial de la Compañía Sun Microsystems.* [En línea] Sun Microsystems, Inc., 2009. [Citado el: 8 de enero de 2009.] http://java.sun.com/javaee/technologies/jta/.

20. **Middleware, Red Hat.** Relational Persistence for Java and .NET. *sitio oficial de Hibernate.* [En línea] Red Hat Middleware, LLC, 2006,. http://www.hibernate.org/.

21. **Schildt, Herbert.** *Java;A Beginner's Guide,Third Edition.* Estados Unidos de América : The McGraw-Hill Companies., 2005. ISBN 0-07-146650-9.

22. **Sun Microsystems, Inc.** Java Platform, Enterprise Edition (Java EE).JavaServer Faces Technology. *sitio oficial de la Corporacion Sun Microsystems.* [En línea] 2009. [Citado el: 5 de enero de 2009.] http://java.sun.com/javaee/javaserverfaces/.

23. **Real Academia Española.** DICCIONARIO DE LA LENGUA ESPAÑOLA - Vigésima segunda edición. *Real Academia Española.* [En línea] 2009. [Citado el: 10 de Diciembre de 2008.] http://www.rae.es/rae.html.

24. **Rumbaugh, James, Jacobson, Ivar y Booch, Grady.** *Lenguaje Unificado de Modelado. Manual de Referencia.* Nadrid : Pearson Educacion, 2000.

25. **Gamma, Erich, y otros.** *Design Patterns Elements of Reuseable Object-Oriented Software.* s.l. : Addison Wesley, 1994.

26. **Larman, Craig.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* México : Prentice Hall, 1999. 970-17-0261-1.

27. **Pressman, Roger S.** *Ingenirría del Software. Un enfoque práctico.* España : McGraw-Hill, 2002.

28. **Matos López, Yordankis y Vázquez Sánchez, Angel Alberto.** Trabajo de Diploma: Análisis, diseño e implementación de la capa de lógica de negocio del módulo Investigación en Ciencias Forenses del SIIPOL. Ciudad de la Habana : UCI, 2008.

GLOSARIO DE TÉRMINOS

AOP: Acrónimo de Programación Orientada a Aspectos, paradigma de programación relativamente reciente cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos.

Apache Tomcat: Servidor web con soporte de servlet y JSP. Incluye el compilador Jasper, que compila las páginas JSP convirtiéndolas en servlet.

API:(Application Programming Interface), acrónimo de Interfaz de Programación de Aplicaciones, es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Una API representa una interfaz de comunicación entre componentes software.

Aplicación: En informática, tipo de programa informático diseñado para facilitar al usuario la realización de un determinado tipo de trabajo. Esto lo diferencia principalmente de otros tipos de programas como los sistemas operativos, las utilidades y los lenguajes de programación, que realizan tareas más avanzadas y no pertinentes al usuario común.

Applet: Componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo un navegador web.

Banda Ancha: En telecomunicaciones a la transmisión de datos en el cual se envían simultáneamente varias piezas de información, con el objeto de incrementar la velocidad de transmisión efectiva.

Base de Datos: Conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En este sentido, una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta.

Caso de Uso: En ingeniería del software, técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico.

Cliente: En informática, equipo o proceso que accede a recursos y servicios brindados por otro llamado Servidor, generalmente de forma remota.

Criminalística: Como ciencia, elaborada sobre los medios y métodos especiales para el descubrimiento, recolección, análisis, investigación y apreciación de las pruebas con el fin de esclarecer las manifestaciones delictivas, estudia procesos, regularidades, fenómenos y hechos punibles desde el punto de vista jurídico, el descubrimiento de los autores de tales hechos , así como la determinación del valor probatorio de determinadas huellas, mediante el análisis integral del suceso para la obtención del esclarecimiento del delito.

DAO: (Data Access Object) acrónimo de Objeto de Acceso a Datos, es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo.

Delito: El delito, en sentido estricto, es definido como una conducta o acción típica (tipificada por la ley), antijurídica (contraria a Derecho), culpable y punible. Supone una conducta infraccional

del Derecho penal, es decir, una acción u omisión tipificada y penada por la ley.

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.

Depuración de código: Comprobación de los estados de variables y objetos en tiempo de ejecución para determinar errores.

EJB: Acrónimo de Enterprise JavaBeans, son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE de Sun Microsystems (ahora JEE 5.0). Su especificación detalla cómo los servidores de aplicaciones proveen objetos desde el lado del servidor que son, precisamente, los EJB.

Empresa: La empresa es la unidad económico-social, con fines de lucro, en la que el capital, el trabajo y la dirección se coordinan para realizar una producción socialmente útil, de acuerdo con las exigencias del bien común.

Experticia: Prueba realizada por un perito o experto sobre un elemento relacionado con un hecho, generalmente criminal, que se pretende esclarecer

HQL: (Hibernate Query Language) acrónimo de lenguaje de consultas de Hibernate, similar al SQL pero trabaja a nivel de objetos no de tablas.

HTTP:(HyperText Transfer Protocol) acrónimo del protocolo de transferencia de hipertexto, es el protocolo usado en cada transacción de la Web (WWW). HTTP fue desarrollado por el consorcio W3C y la IETF, colaboración que culminó en 1999 con la publicación de una serie de RFC,

siendo el más importante de ellos el RFC 2616, que especifica la versión 1.1. HTTP define la sintaxis y la semántica que utilizan los elementos software de la arquitectura web (clientes, servidores, proxies) para comunicarse.

HTTPS: (Hypertext Transfer Protocol Secure) acrónimo de protocolo seguro de transferencia de hipertexto, es un protocolo de red basado en el protocolo HTTP, destinado a la transferencia segura de datos de hipertexto, es decir, es la versión segura de HTTP. El sistema HTTPS utiliza un cifrado basado en las Secure Socket Layers (SSL) para crear un canal cifrado (cuyo nivel de cifrado depende del servidor remoto y del navegador utilizado por el cliente) más apropiado para el tráfico de información sensible que el protocolo HTTP.

Internet: conjunto descentralizado de redes de comunicación interconectadas, que utilizan la familia de protocolos TCP/IP, garantizando que las redes físicas heterogéneas que la componen funcionen como una red lógica única, de alcance mundial.

Intranet: Una Intranet es una red de ordenadores privados que utiliza tecnología Internet para compartir de forma segura cualquier información o programa del sistema operativo para evitar que cualquier usuario de internet pueda entrar a robar archivos privados.

Investigación: La investigación científica es la búsqueda de conocimientos o de soluciones a problemas de carácter científico y cultural. Es la búsqueda intencionada de conocimientos o soluciones a problemas.

IoC: (Invertion of Control) acrónimo de Inversión del control, técnica de programación utilizada para manejar las dependencias entre objetos.

Itext: Biblioteca Open Source para crear y manipular archivos PDF, RTF, y HTML en Java. Fue escrita por Bruno Lowagie, Paulo Soares, y otros; está distribuida bajo la Mozilla Public License con la LGPL como licencia alternativa. El mismo documento puede ser exportado en múltiples formatos, o múltiples instancias del mismo formato.

JDBC: (Java Database Connectivity) acrónimo de conectividad a base de datos con java, es una 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.

JRE: (Java Runtime Environment) Acrónimo de 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.

JSP: acrónimo de JavaServer Pages, tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo. Esta tecnología es un desarrollo de la compañía Sun Microsystems.

Juicio: El juicio es una causa jurídica y actual entre partes, y sometido al conocimiento de un tribunal de justicia. Esto presupone la existencia de una controversia, que constituye el contenido del proceso, la cual va a ser resuelta por el órgano jurisdiccional a través de un procedimiento.

Metodología de desarrollo de software: En Ingeniería de Software, es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información. Se refiere a un framework que es usado para estructurar, planear y controlar el proceso de

desarrollo en sistemas de información.

Módulo: Término que denota una unidad para el almacenamiento y manipulación del software. La palabra no corresponde a una única estructura de UML, sino que incluye varias estructuras.

Navegador Web o Browser: Programa que permite visualizar la información que contiene una página web, ya esté alojada en un servidor dentro de la World Wide Web o en uno local. El navegador interpreta el código, HTML generalmente, en el que está escrita la página web y lo presenta en pantalla permitiendo al usuario interactuar con su contenido y navegar hacia otros lugares de la red mediante enlaces o hipervínculos.

ORM: (Object Relational Mapping) acrónimo de mapeo de objetos relacionales, es la persistencia automática y transparente de objetos en una aplicación Java a tablas en una base de datos relacional, usando metadatos que describen el mapeo entre objetos y la base de datos, ORM, en esencia, trabaja transformando datos de una representación a la otra.

Paquete: Término que denota un mecanismo de propósito general para organizar en grupos los elementos. Se pueden anidar dentro de otros paquetes, y en él pueden aparecer tanto elementos del modelo como diagramas.

POI: El proyecto POI consiste en APIs para manipular varios formatos de ficheros basados en el formato de Documento Compuesto OLE 2 de Microsoft, utilizando Java puro. En concreto, se pueden leer y escribir ficheros MS Excel utilizando Java. Pronto se podrá leer y escribir ficheros Word utilizando Java. POI es su solución Java Excel así como su solución Java Word.

POJO: Acrónimo de Plain Old Java Object es una sigla empleada por desarrolladores de Java

para enfatizar el uso de clases simples y que no dependen de un framework en especial.

POO: La Programación Orientada a Objetos (POO u OOP según sus siglas en inglés) es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo y encapsulamiento.

Protocolo TCP/IP: La familia de protocolos de Internet es un conjunto de protocolos de red en la que se basa Internet y que permiten la transmisión de datos entre redes de computadoras. En ocasiones se le denomina conjunto de protocolos TCP/IP, en referencia a los dos protocolos más importantes que la componen: Protocolo de Control de Transmisión (TCP) y Protocolo de Internet (IP), que fueron los dos primeros en definirse, y que son los más utilizados de la familia.

Requisito o requerimiento funcional: Define el comportamiento interno del software: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que muestran cómo los casos de uso serán llevados a la práctica. Son complementados por los requerimientos no funcionales, que se enfocan en cambio en el diseño o la implementación. Como se define en la ingeniería de requerimientos, los requerimientos funcionales establecen los comportamientos del sistema.

Requisito o requerimiento no funcional: En la Ingeniería de Sistemas y la Ingeniería de Software, un requerimiento que especifica criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos, ya que éstos corresponden a los requerimientos funcionales.

Requisito o requerimiento: Circunstancia o condición necesaria para algo. En la Ingeniería de

Sistemas, es una necesidad documentada sobre el contenido, forma o funcionalidad de un producto o servicio. Se usa en un sentido formal en la ingeniería de sistemas o la ingeniería de software.

Servidor de aplicaciones: En informática, se denomina servidor de aplicaciones a un servidor en una red de computadores que ejecuta ciertas aplicaciones. Usualmente se trata de un dispositivo de software que proporciona servicios de aplicación a las computadoras cliente. Un servidor de aplicaciones generalmente gestiona la mayor parte (o la totalidad) de las funciones de lógica de negocio y de acceso a los datos de la aplicación.

Servidor: En informática, un servidor es una computadora que, formando parte de una red, provee servicios a otros denominados clientes.

Servlet o java servlet: Los servlets son objetos que corren dentro del contexto de un contenedor de servlets (ej: Tomcat) y extienden su funcionalidad. También podrían correr dentro de un servidor de aplicaciones (ej: OC4J Oracle), que, además de contenedor para servlet, tendrá contenedor para objetos más avanzados, como son los EJB (Tomcat sólo es un contenedor de servlets).

Sistema Gestor de Base de Datos: Los sistemas de gestión de base de datos (SGBD); (en inglés: Database Management System, abreviado DBMS) son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan.

Sistema Operativo: Software de sistema, conjunto de programas de computación destinados a realizar muchas tareas entre las que destaca la administración eficaz de sus recursos. Comienza

a trabajar cuando en memoria se carga un programa específico y aun antes de ello, que se ejecuta al iniciar el equipo, o al iniciar una máquina virtual, y gestiona el hardware de la máquina desde los niveles más básicos, brindando una interfaz con el usuario.

Software de aplicación: Aquel que permite a los usuarios llevar a cabo una o varias tareas específicas, en cualquier campo de actividad susceptible de ser automatizado o asistido, con especial énfasis en los negocios.

Software: Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación. (12)

SQL: (Structured Query Language) acrónimo de Lenguaje de consulta estructurado, lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo lanzar consultas con el fin de recuperar -de una forma sencilla- información de interés de una base de datos, así como también hacer cambios sobre la misma.

Telnet: (TELecommunication NETwork) acrónimo de red de telecomunicación, es el nombre de un protocolo de red (y del programa informático que implementa el cliente), que sirve para acceder mediante una red a otra máquina, para manejarla remotamente como si estuviéramos sentados delante de ella. Para que la conexión funcione, como en todos los servicios de Internet, la máquina a la que se acceda debe tener un programa especial que reciba y gestione las conexiones.

Tiles: acrónimo de Apache Tiles, motor de plantillas diseñado para simplificar el desarrollo de interfaces de usuario de aplicaciones Web. Permite definir a los autores fragmentos de página que pueden montarse en una página completa en tiempo de ejecución. Estos fragmentos, o tiles,

se puede utilizar como simples inclusiones con el fin de reducir la duplicación de los elementos de la página común o tiles incrustados dentro de otros para desarrollar una serie de plantillas reutilizables

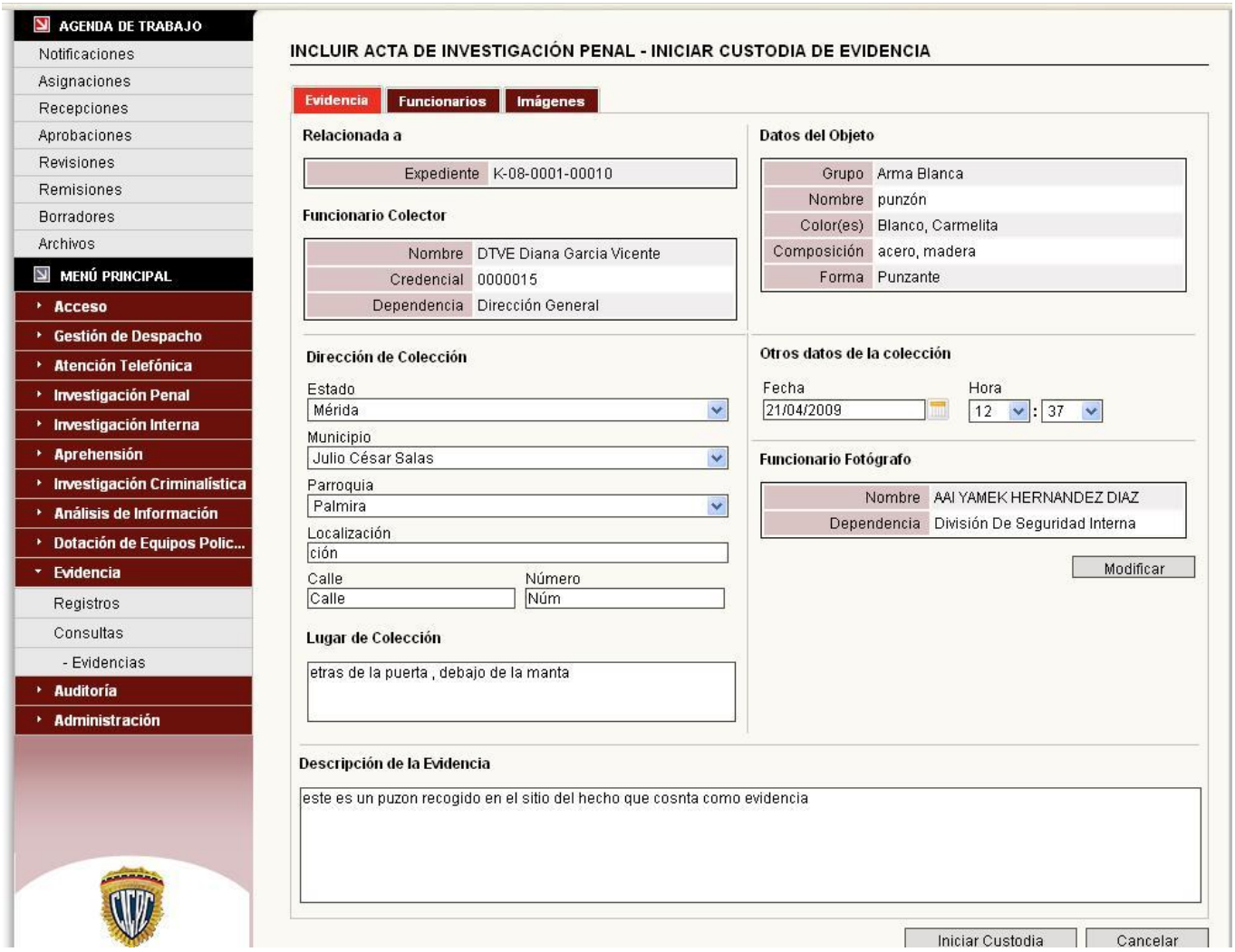
Usuario: Un usuario es la persona que utiliza o trabaja con algún objeto o que es destinataria de algún servicio público o privado, empresarial o profesional.

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

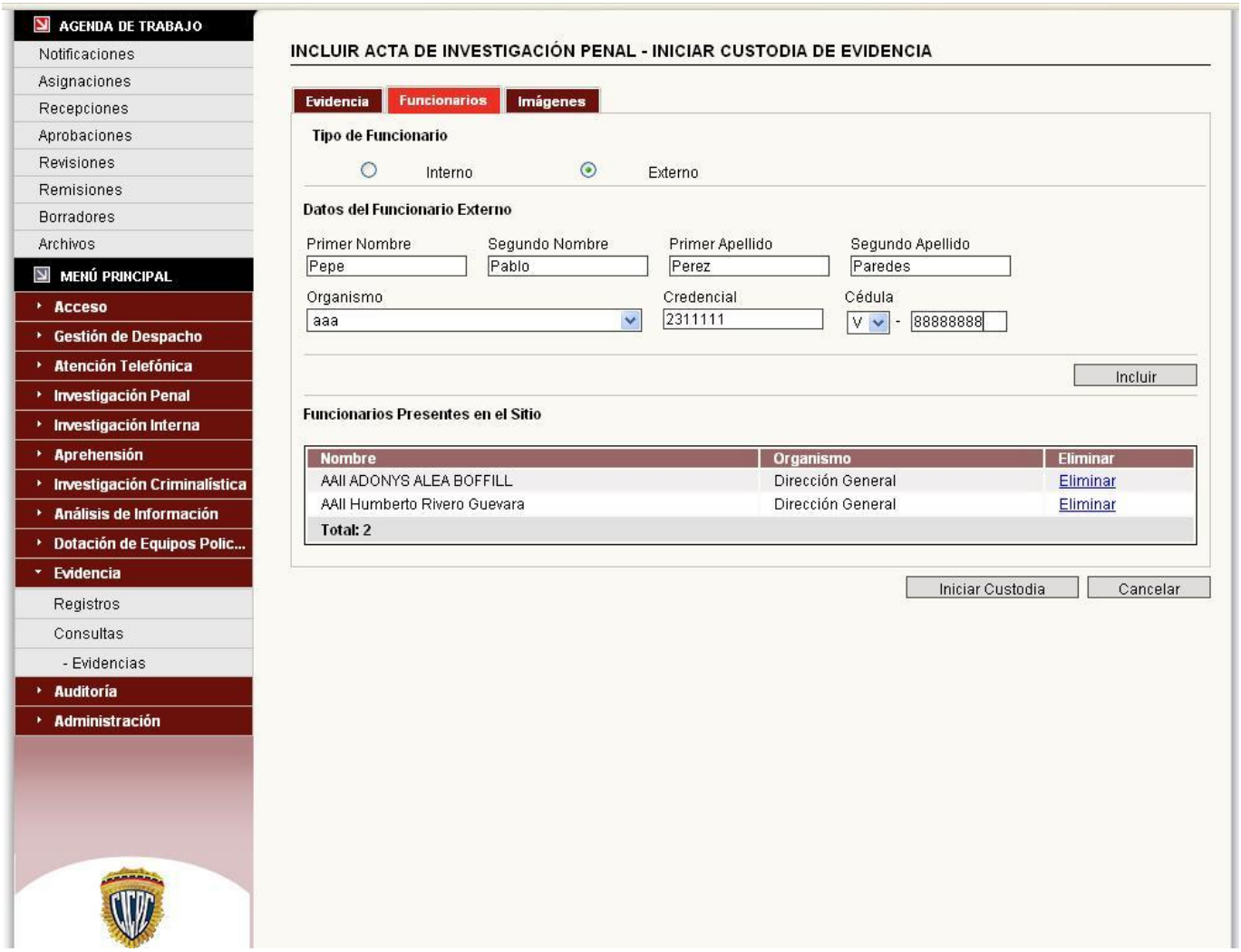
Velocity: acrónimo de Apache Velocity, motor de plantillas basado en Java que permite a los diseñadores de páginas hacer referencia a métodos definidos dentro del código Java. Los diseñadores Web pueden trabajar en paralelo con los programadores Java para desarrollar sitios de acuerdo al modelo de Modelo-Vista-Controlador (MVC), permitiendo que los diseñadores se concentren únicamente en crear un sitio bien diseñado y que los programadores se encarguen solamente de escribir código de primera calidad.

ANEXOS

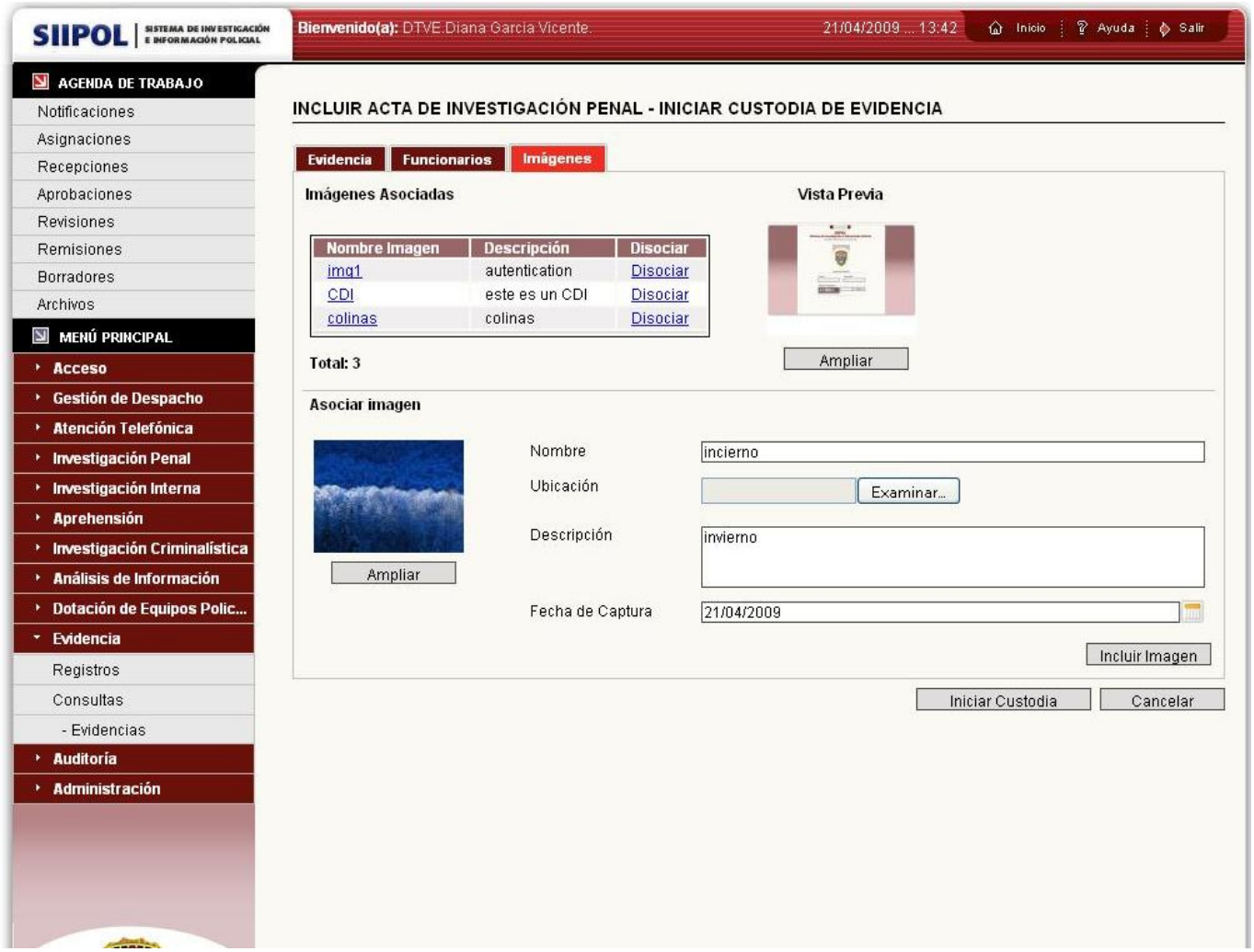
Anexo 1 *INTERFACES DE USUARIO DETERMINADAS PARA LOS CASOS DE USO DEL SUBMÓDULO DE EVIDENCIAS.*



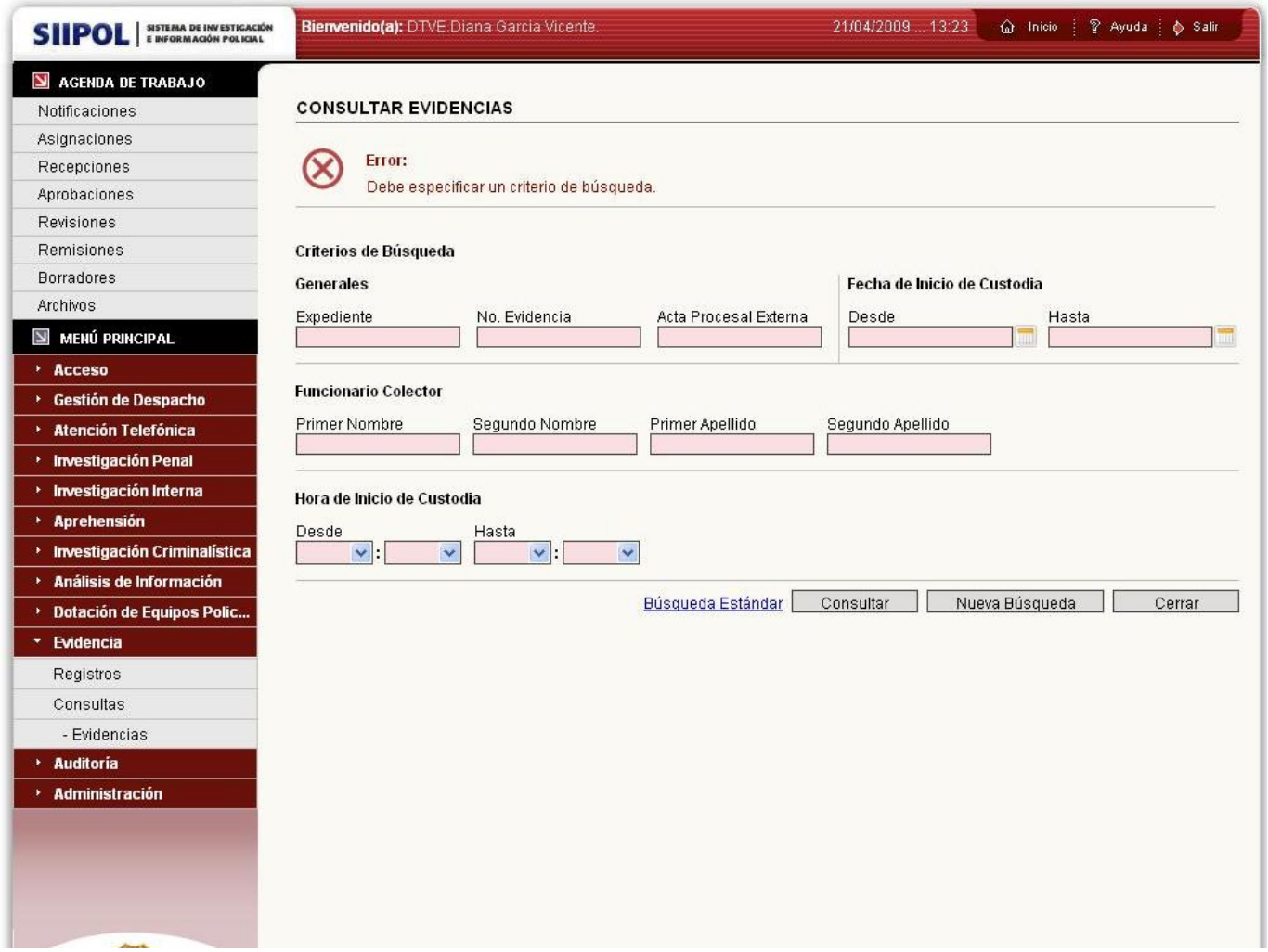
Interfaz del Caso de Uso Iniciar Cadena de Custodia para incluir los datos Generales.



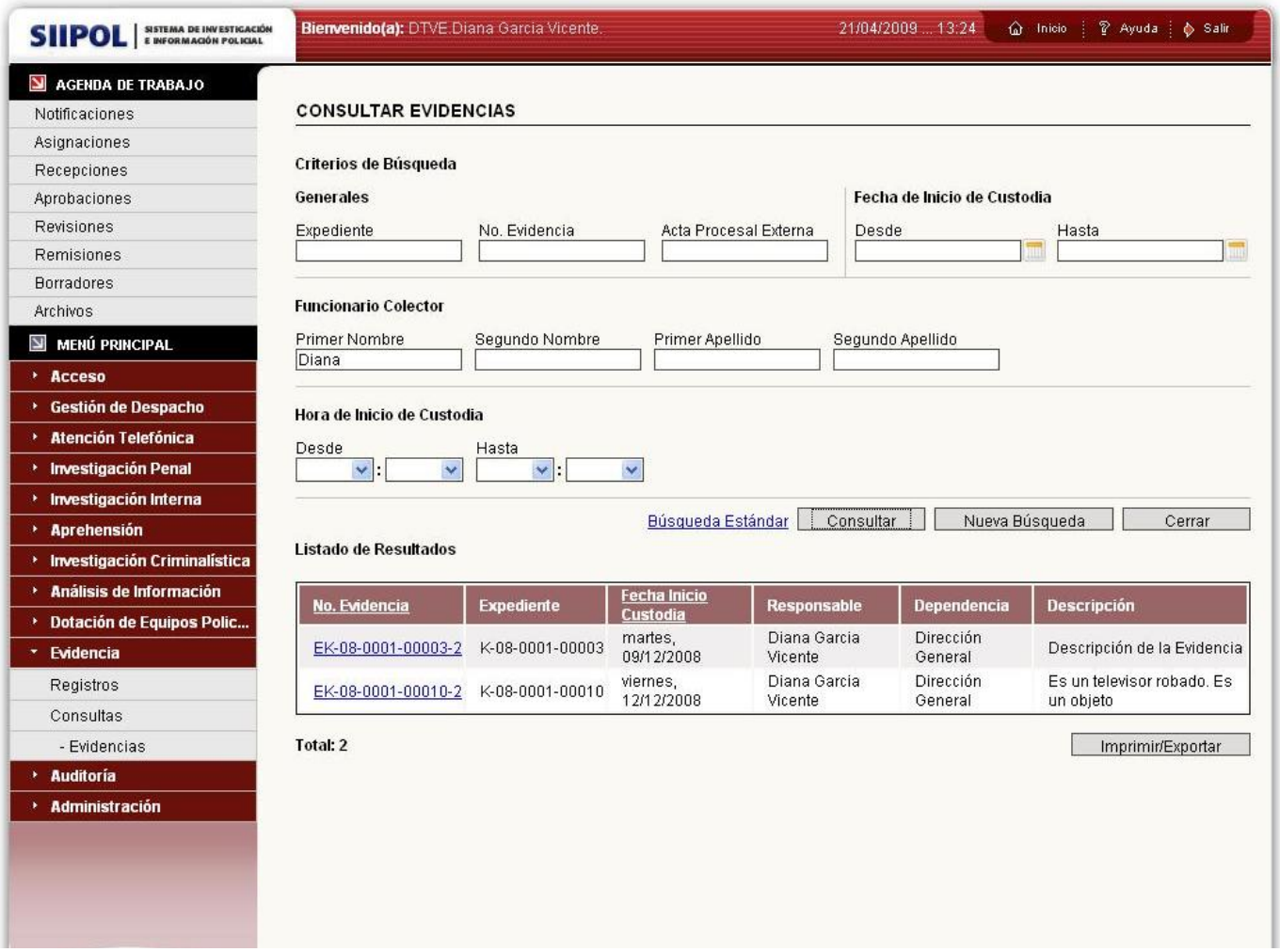
Interfaz del Caso de Uso Iniciar Cadena de Custodia para incluir los datos de los Funcionarios presentes en el Sitio del suceso.



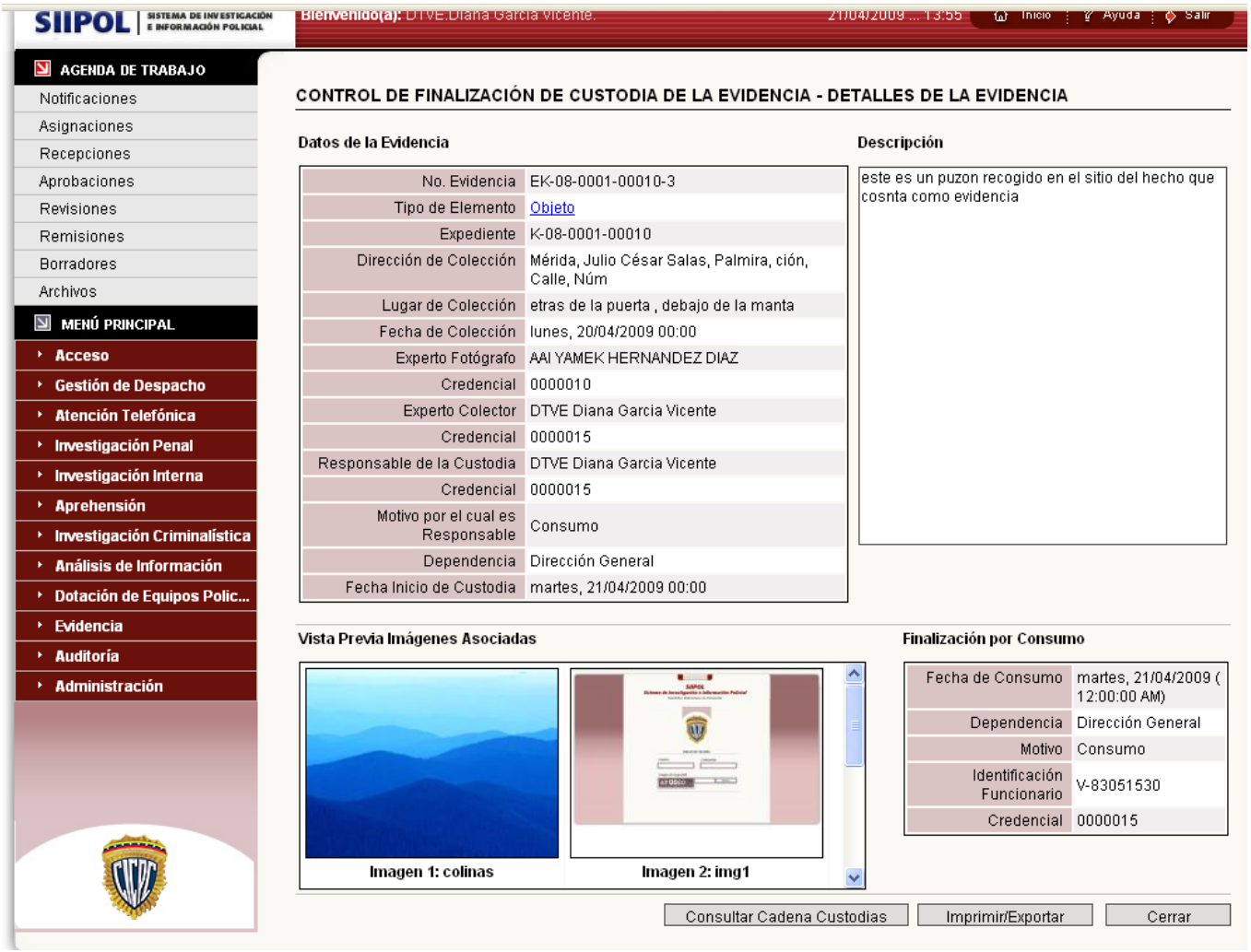
Interfaz del Caso de Uso Iniciar Cadena de Custodia para incluir imágenes asociadas a la Evidencia.



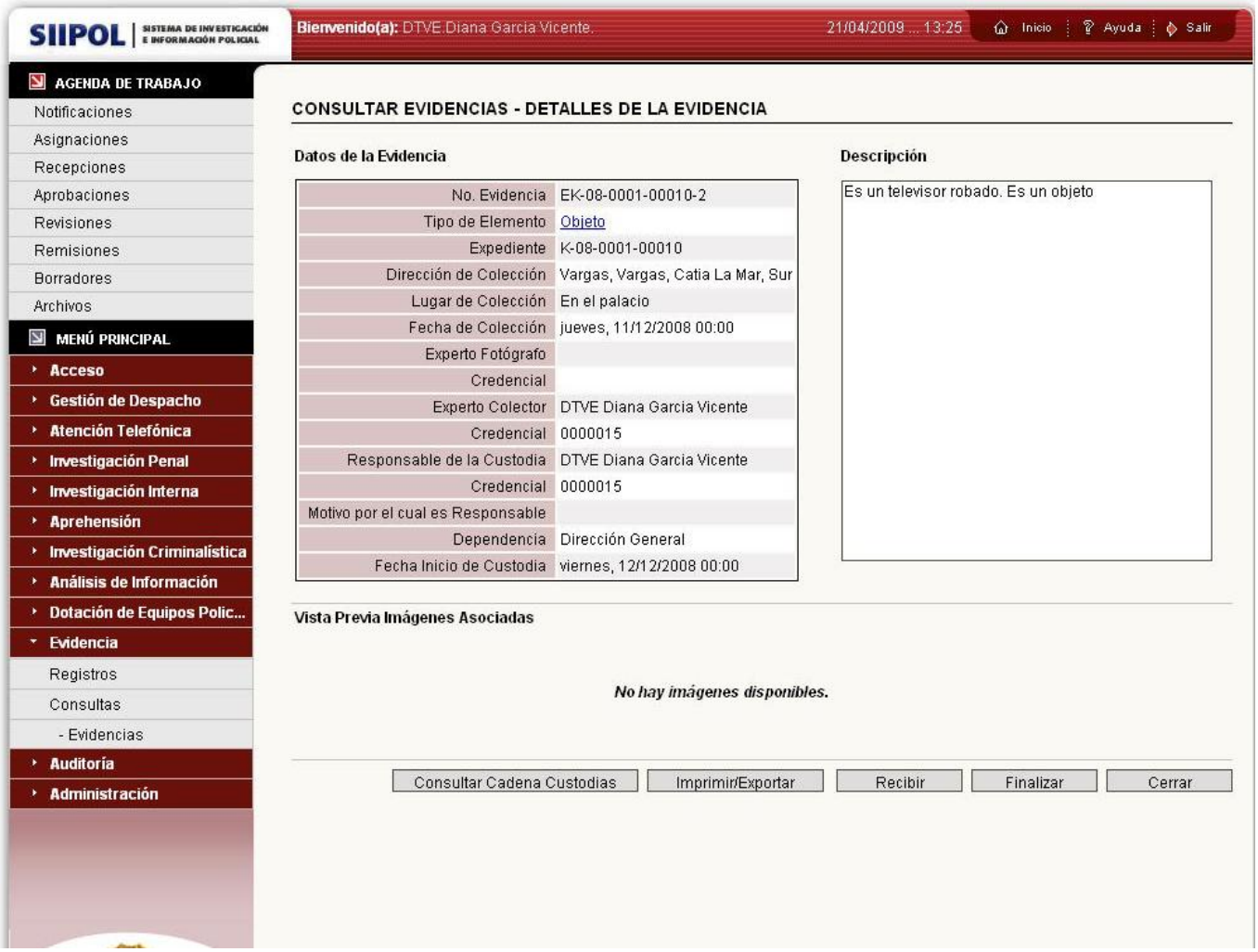
Interfaz del Caso de Uso Consultar Evidencias donde se muestra un mensaje de error por no selección ningún criterio para realizar la búsqueda.



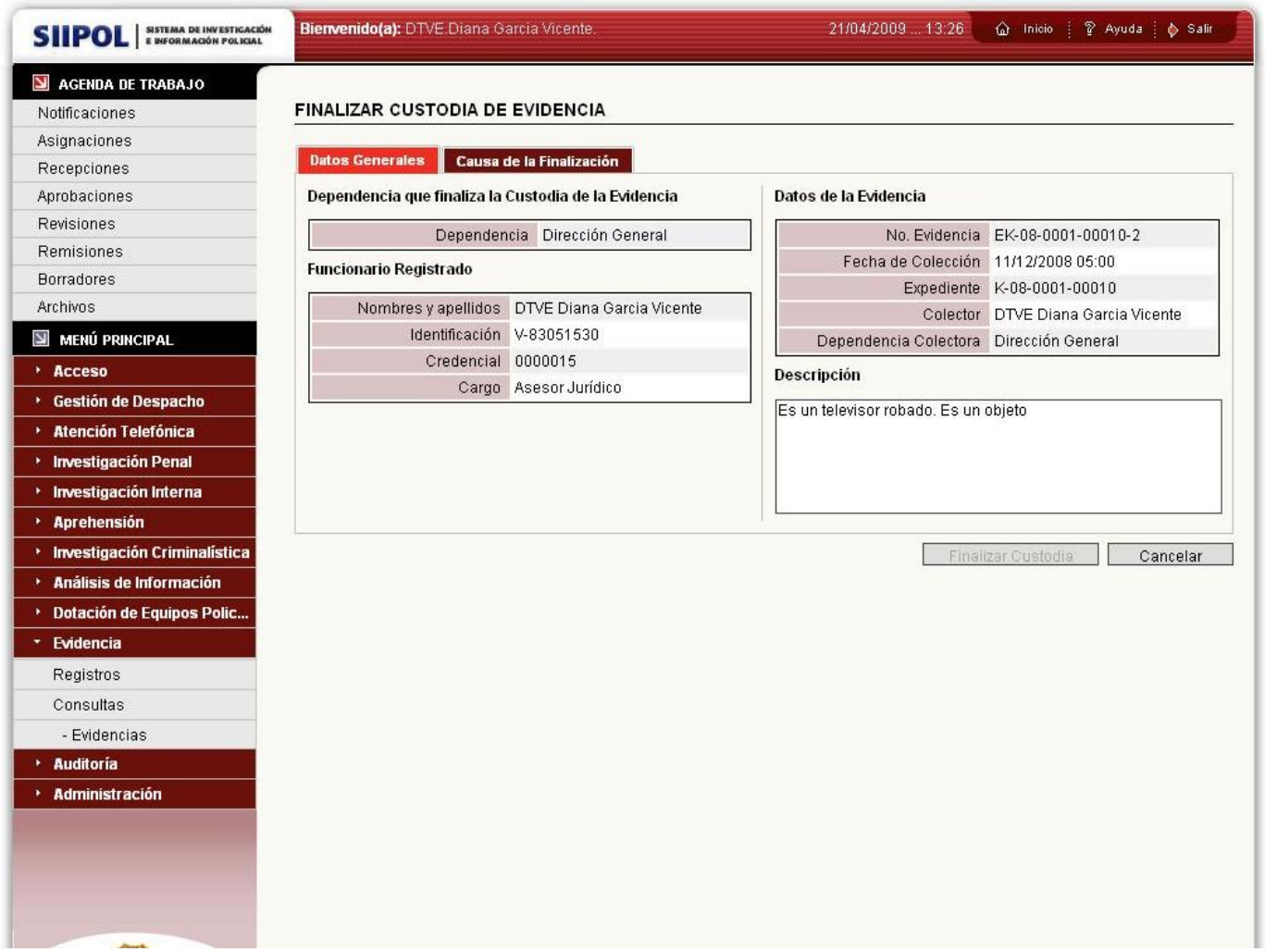
Interfaz del Caso de Uso Consultar Evidencias donde se muestra los resultados obtenidos luego de efectuar la consulta.



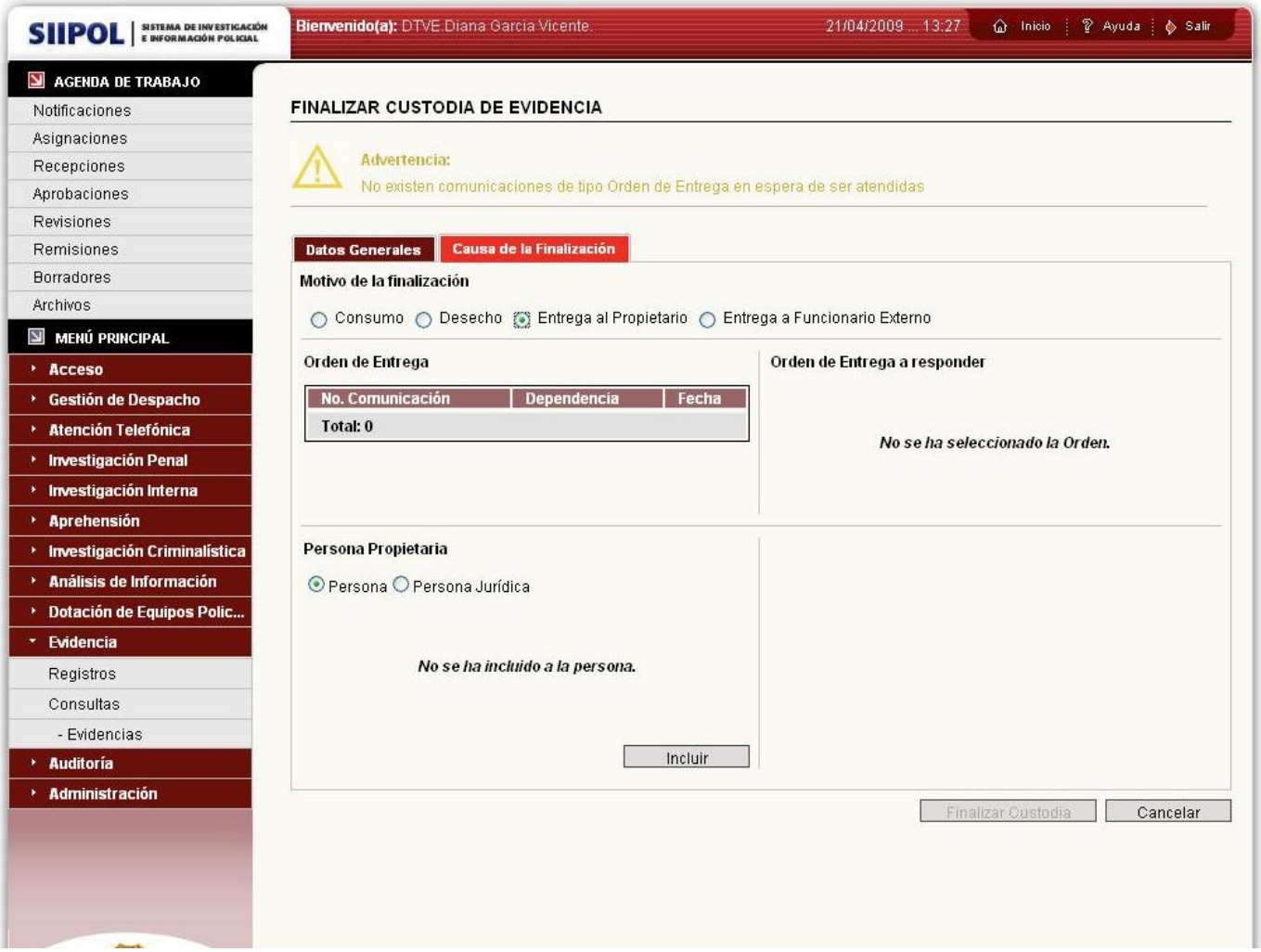
Interfaz del Caso de Uso Ver Detalles de Evidencia donde se muestran los datos relacionados a la Evidencia, en este caso esta tiene imágenes asociadas y ha sido finalizada por consumo. Nótese que después de finalizada la Evidencia, solo se puede mostrar la cadena de Custodia de la misma o Imprimir/Exportar los datos que se muestran.



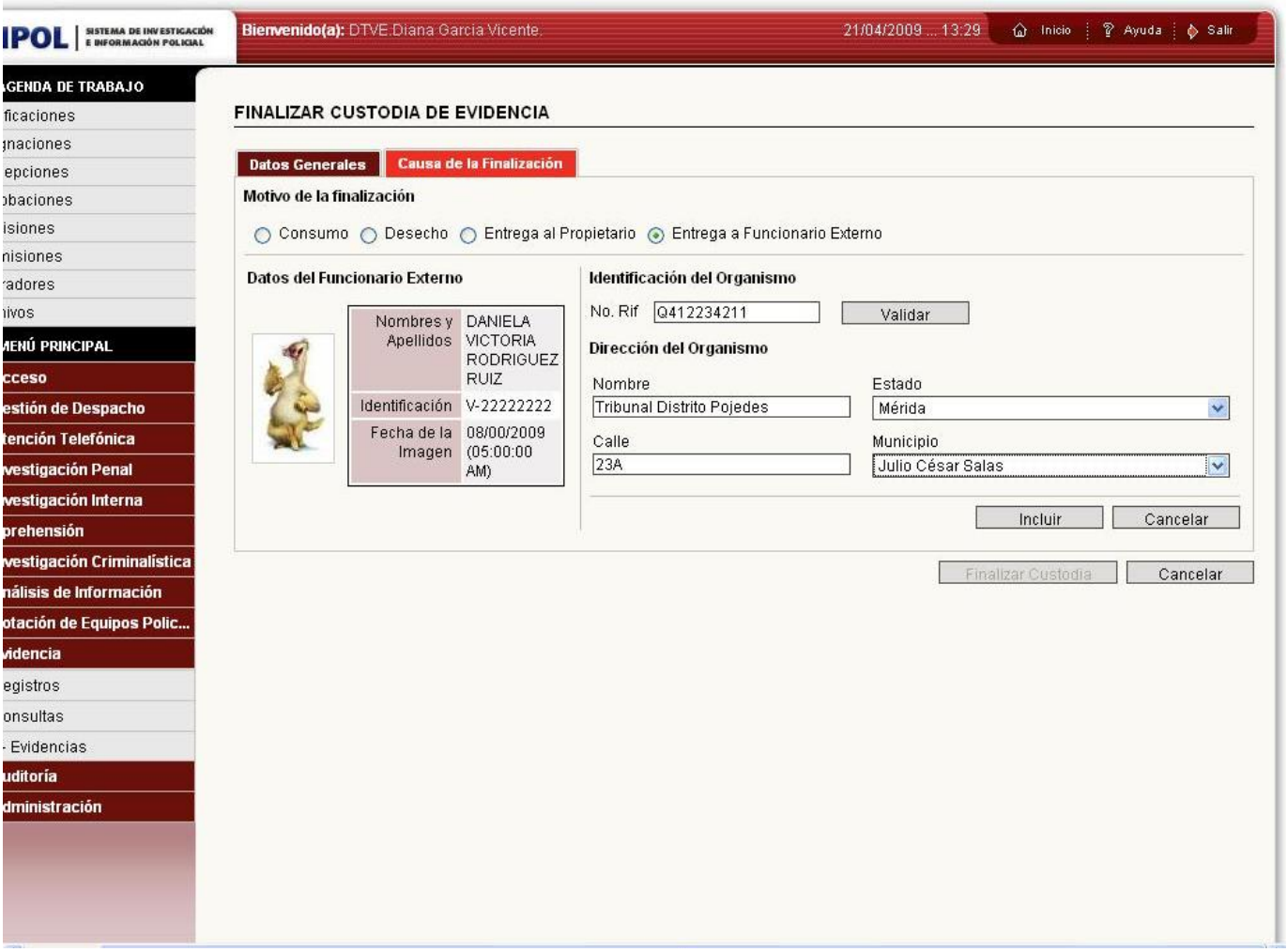
Interfaz del Caso de Uso Ver Detalles de Evidencia donde se muestran los datos relacionados a la Evidencia, en este caso el usuario autenticado es el responsable de la Evidencia que se muestra, dándole un grupo de opciones además de las vistas en el caso anterior.



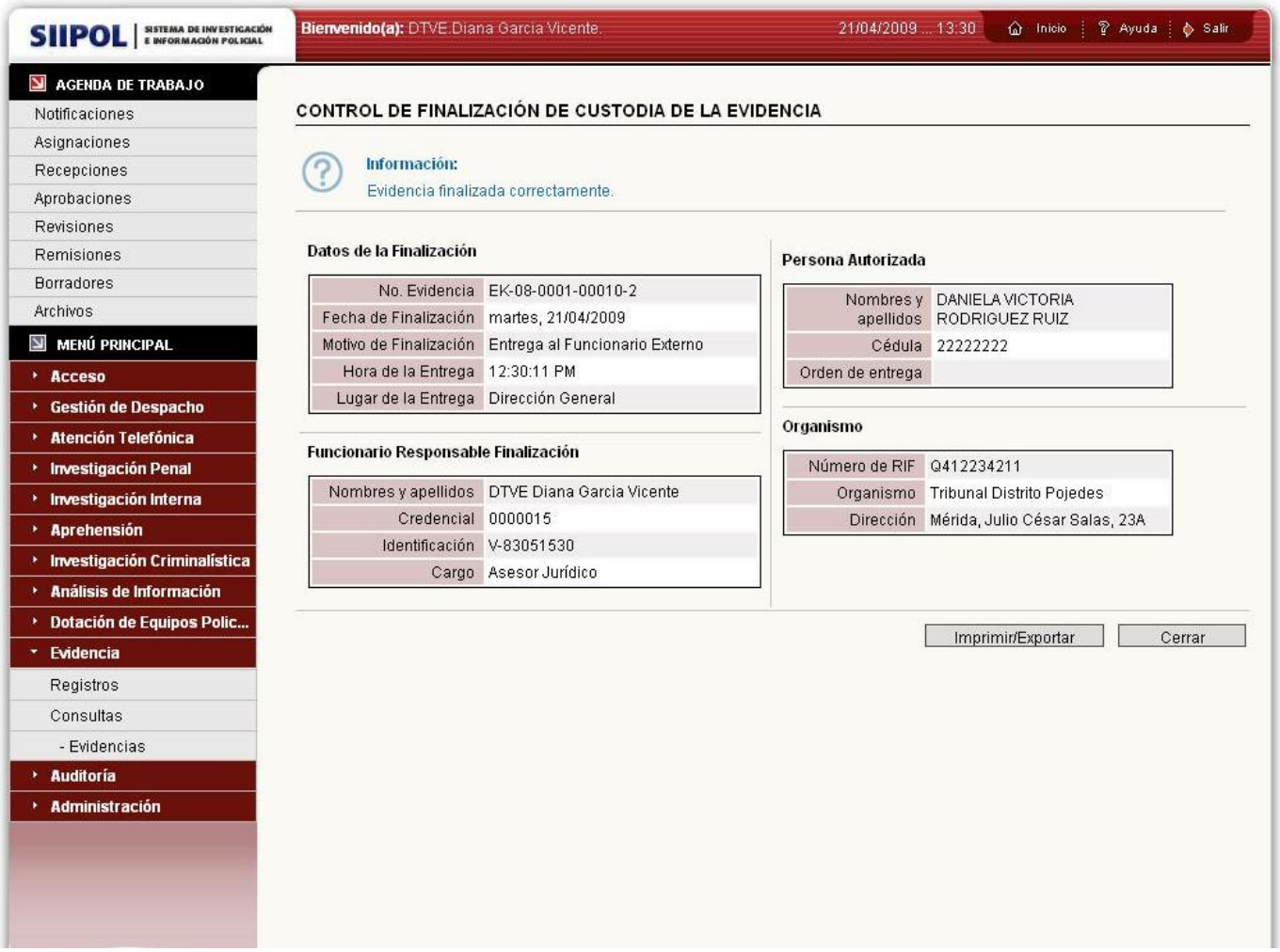
Interfaz del Caso de Uso Finalizar Cadena de Custodia donde se muestran los datos relacionados a la finalización de la Evidencia, y una pestaña para determinar las causas por la cual se va a finalizar la custodia.



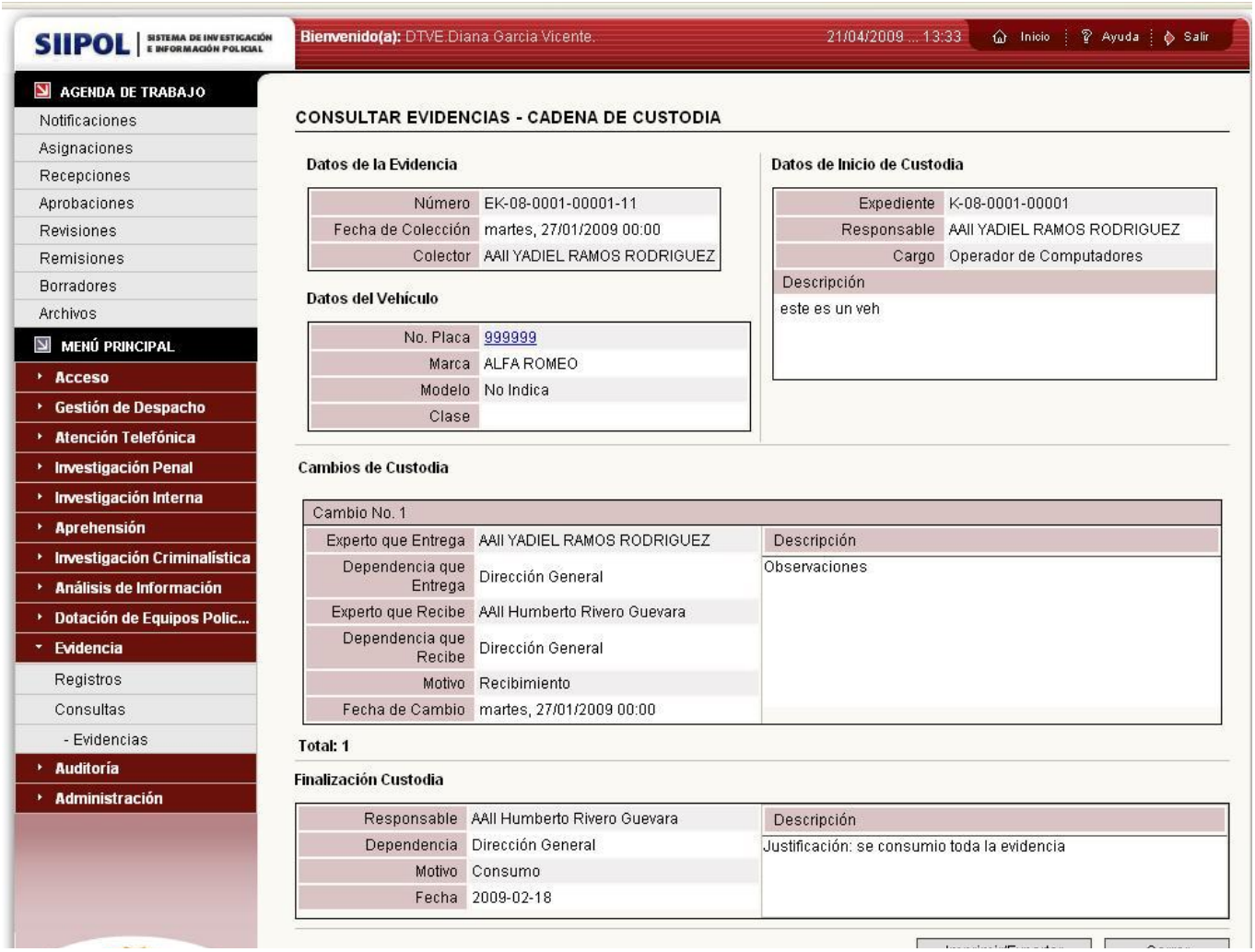
Interfaz del Caso de Uso Finalizar Cadena de Custodia donde se muestran los datos relacionados a la causa de finalización de la Evidencia, en la que se encuentran un conjunto de opciones, en este caso se seleccionó Entrega al propietario y se muestra un mensaje de Advertencia debido a que no existen Ordenes de Entrega por atender, por lo que no se puede finalizar por esta causa.



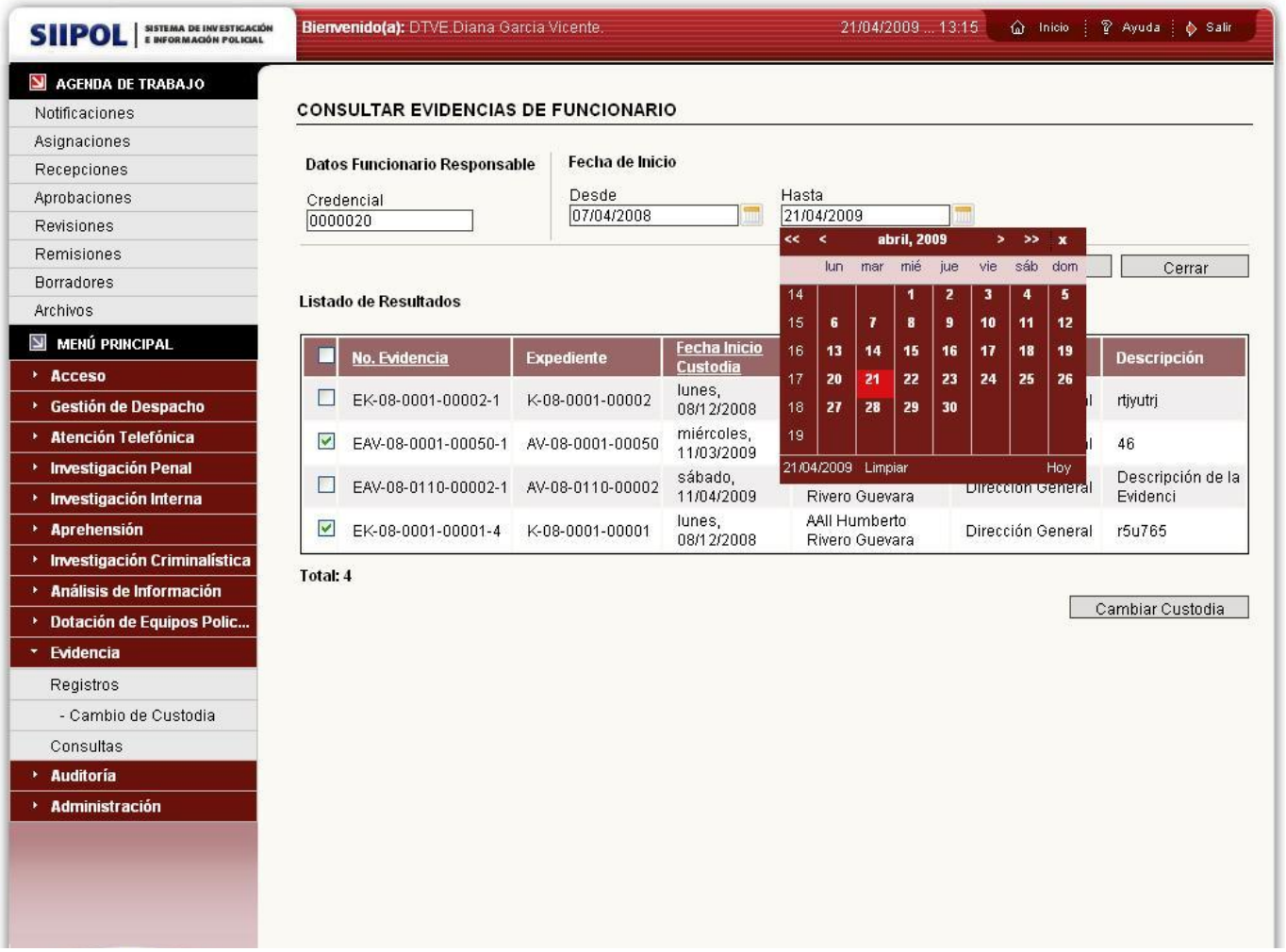
Interfaz del Caso de Uso Finalizar Cadena de Custodia donde se muestran los datos relacionados a la causa de finalización de la Evidencia, en la que se seleccionó la opción de finalizar la Evidencia por entrega al Funcionario Externo donde se muestran los datos correspondientes al Funcionario seleccionado previamente y el Organismo al cual responde.



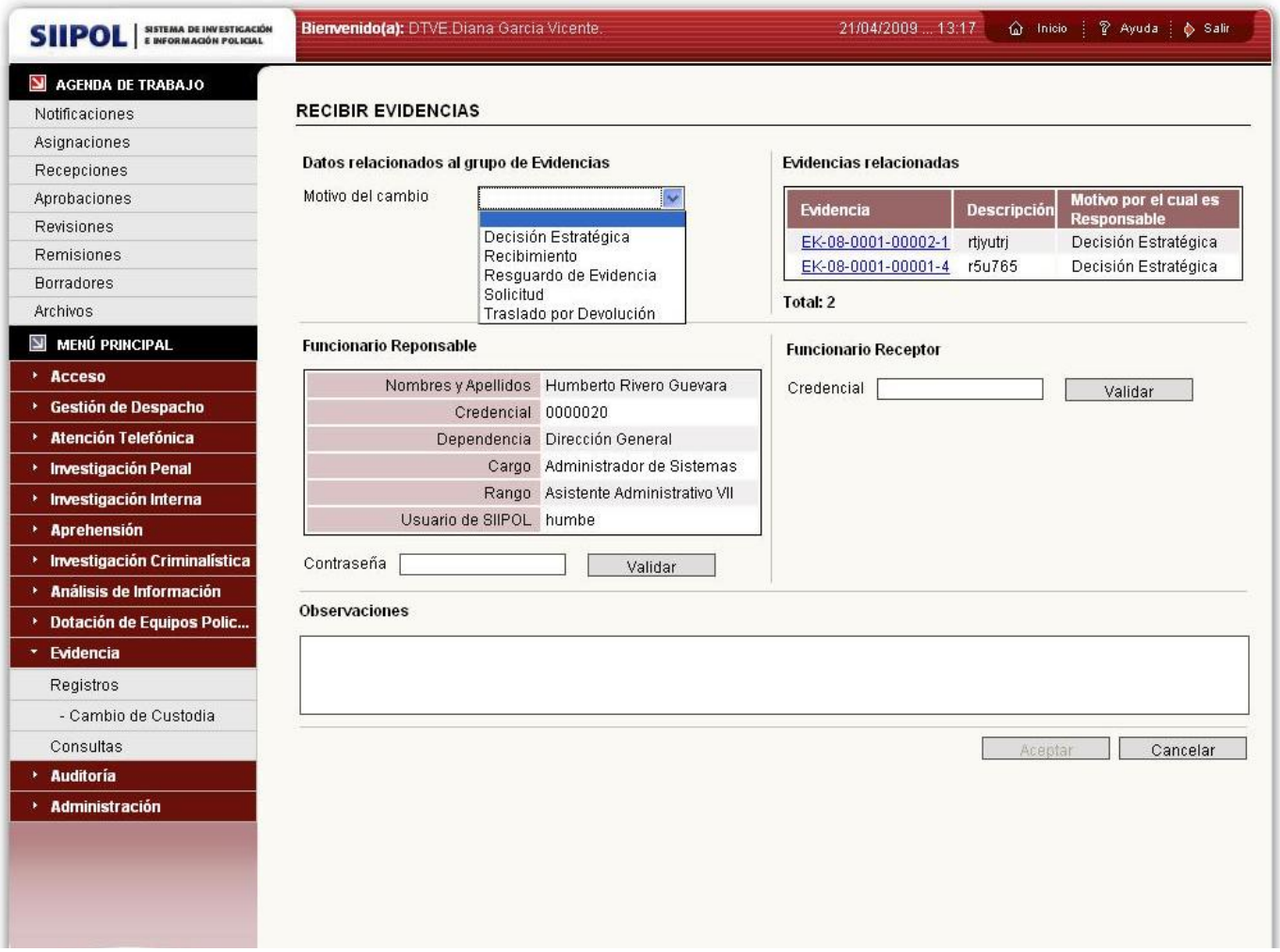
Interfaz del Caso de Uso Finalizar Cadena de Custodia donde se muestra el Control de Finalización de la Evidencia en el cual se recogen los datos de la acción realizada, permitiendo Imprimirlos para tener una constancia de la misma.



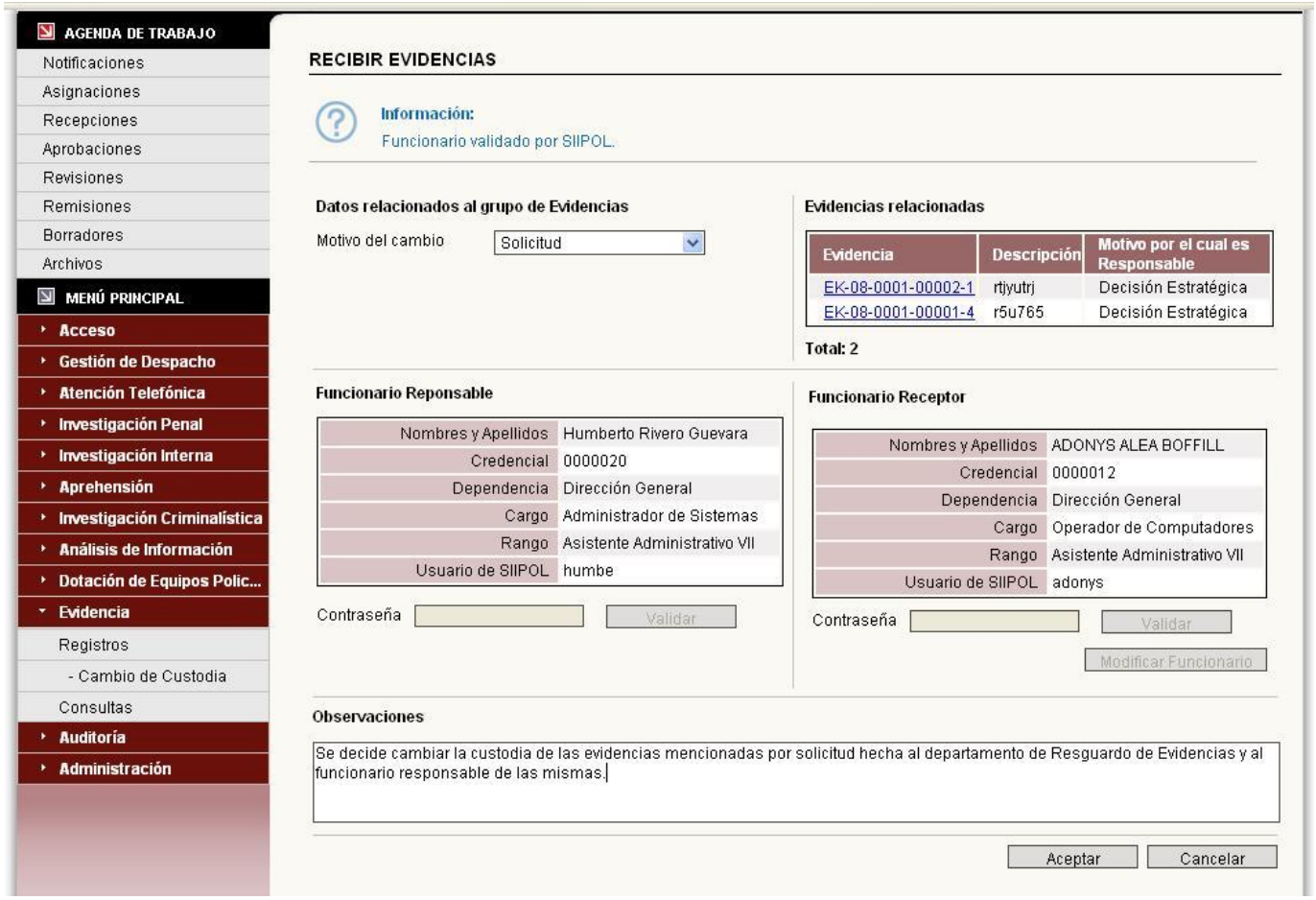
Interfaz del Caso de Uso Consultar Cadena de Custodia donde se muestran los datos referentes a los cambios de Cadena de custodia de la Evidencia y, en este caso, los datos de la finalización de la Cadena de Custodia, permitiendo Imprimirlos para tener una constancia de dichas acciones.



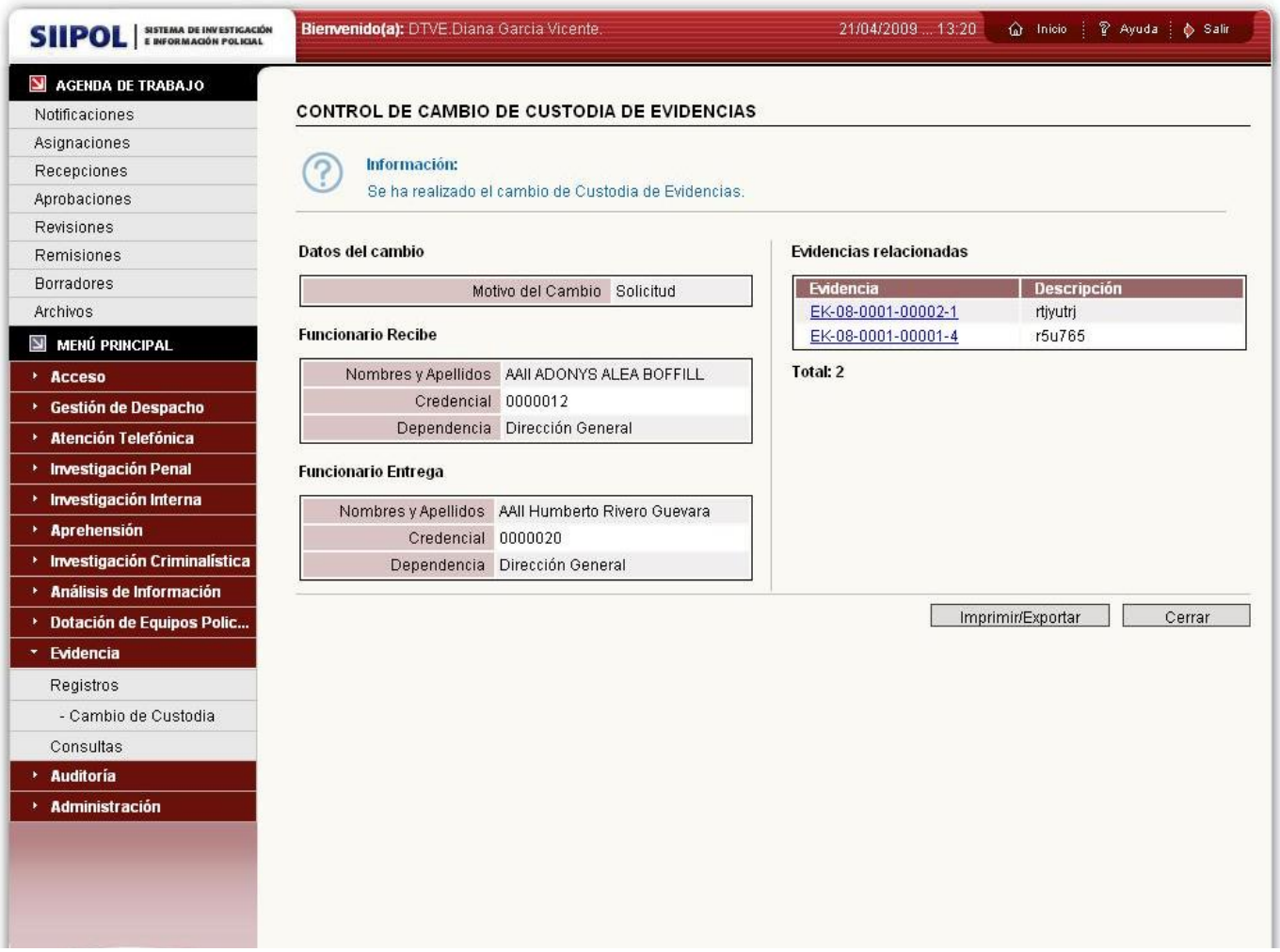
Interfaz del Caso de Uso Recibir Evidencias, Accediendo desde el Menú Principal, donde se realiza una consulta para seleccionar del funcionario responsable de la Custodia, cuál o cuáles Evidencias se van a cambiar de custodia.



Interfaz del Caso de Uso Recibir Evidencias, donde se gestiona el cambio de custodia de un funcionario a otro. A esta interfaz se llega también desde el Caso de Uso Ver Detalles de Evidencia.



Interfaz del Caso de Uso Recibir Evidencias, donde se gestiona el cambio de custodia de un funcionario a otro. En este Caso han sido validados ya los responsables y se proceda a realizar el cambio. A esta interfaz se llega también desde el Caso de Uso Ver Detalles de Evidencia.



Interfaz del Caso de Uso Recibir Evidencias, donde se Muestra el Control de Cambio de Custodia en el que se muestran los elementos relevantes de la acción realizada, permitiendo Imprimir/Exportar los mismos.

REPORTE DE SISTEMA

martes, 21 de abril de 2009
Dirección General", Distrito Capital

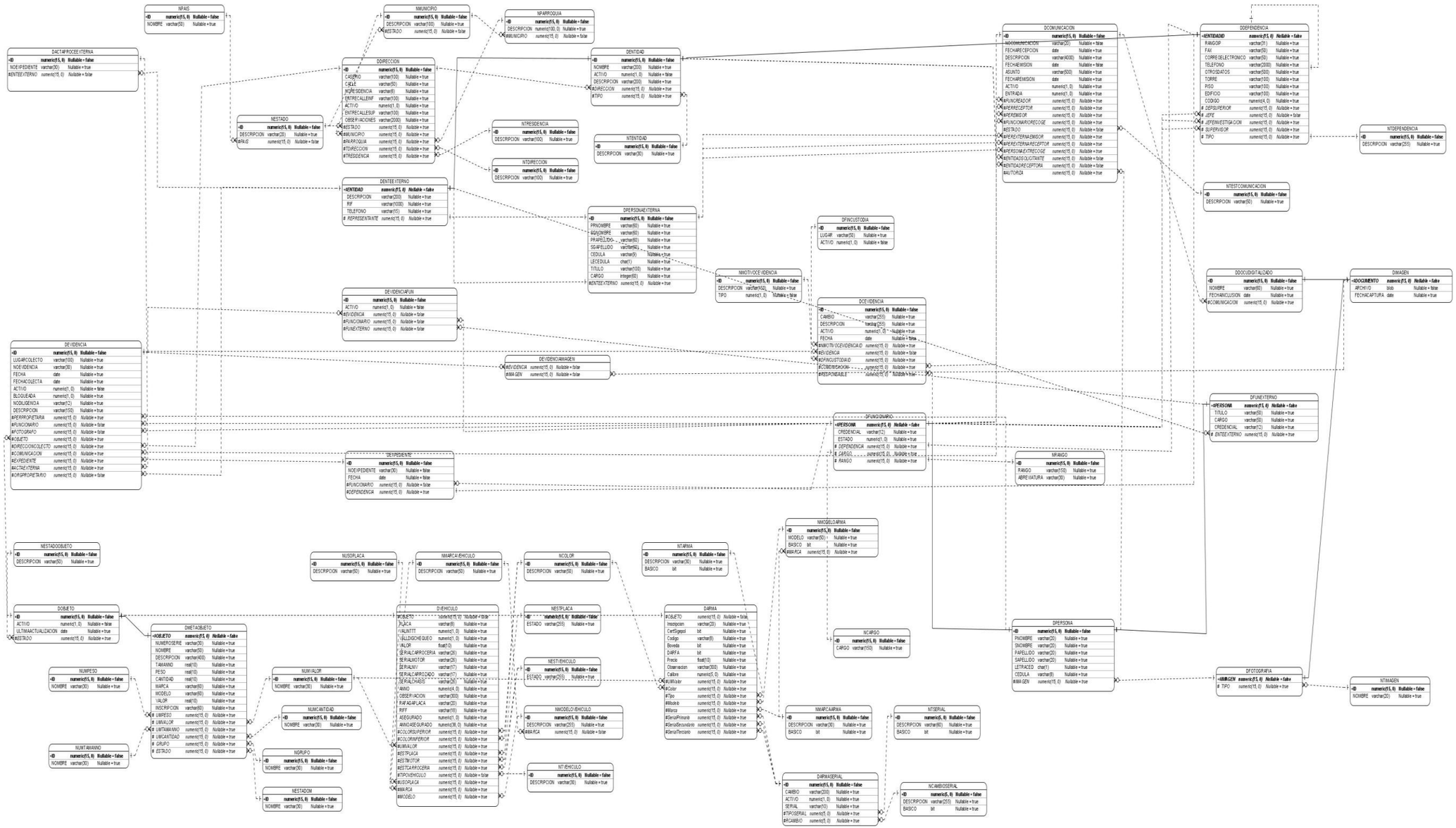
ENTIDAD	CONTROL DE CAMBIO DE CUSTODIA DE EVIDENCIAS

DATOS DEL CAMBIO	
FUNCIONARIO QUE RECIBE	AAII ADONYS ALEA BOFFILL
CREDENCIAL	0000012
DEPENDENCIA	Dirección General
FUNCIONARIO QUE ENTREGA	AAII Humberto Rivero Guevara
CREDENCIAL	0000020
DEPENDENCIA	Dirección General
MOTIVO DEL CAMBIO	Solicitud

EVIDENCIAS RELACIONADAS	
NO. EVIDENCIA	DESCRIPCIÓN
EK-08-0001-00002-1	rtjyutrj
EK-08-0001-00001-4	r5u765

Ejemplo de diseño de Reporte del Sistema, este hace referencia al Control de Cambio del Caso de Uso Recibir Evidencias.

Anexo 2 *DIAGRAMA DE TABLAS DEL MODELO RELACIONAL SUBMÓDULO EVIDENCIAS.*



Anexo 3 *CLASE UTILIZADA PARA LAS PRUEBAS UNITARIAS DEL CÓDIGO DE LA FACHADA DEL SUBMÓDULO EVIDENCIAS, EVIDENCIASFACADEIMPL.*

```
public class PruebaEvidencias extends TestCase {
        private ApplicationContext app;
        private NomencladorFacade nomencladorFacade;
        private InvestigacionPenalFacade investigacionPenalFacade;
        private AnalisisInformacionFacade analisisInformacionFacade;
        private EvidenciaFacade evidenciaFacade;
        private AdministracionFacade administracionFacade;

        @Override
        protected void setUp() throws Exception {
                app = new
ClassPathXmlApplicationContext(CargadorContextos.cargarContextosDeWebXml());
                nomencladorFacade = (NomencladorFacade) app.getBean("nomencladorFacade");
                investigacionPenalFacade = (InvestigacionPenalFacade)
app.getBean("investigacionPenalFacade");
                analisisInformacionFacade = (AnalisisInformacionFacade)
app.getBean("analisisInformacionFacade");
                evidenciaFacade = (EvidenciaFacade) app.getBean("evidenciaFacade");
                administracionFacade = (AdministracionFacade)
app.getBean("administracionFacade");
        }
```


```
public ApplicationContext getApp() {
        return app;
}

public void setApp(ApplicationContext app) {
        this.app = app;
}

public void estProbarContextos() {
        System.out.println("Contextos cargados Correctamente");
}

public void estbuscarEvidenciaPorElemento() {
        Evidencia evidencia = evidenciaFacade.buscarEvidenciaPorElemento(46);
        assertNotNull(evidencia);
}

public void estcambiarCustodiaEvidencia(Integer motivo, Date fechaEntrega, Integer
idEvidencia, Funcionario solicitante, Comunicacion comunicacion, String observaciones) {

}

public void estcambiarCustodiaEvidencia(Elemento elemento, Funcionario funcionario) {
```

```
    }

    public void estconsultarCadenaCustodia() {
        List<CustodiaEvidencia> lista = evidenciaFacade.consultarCadenaCustodia(348);
        assertNotNull(lista);
        if (lista.get(0) != null)
            System.out.println("okok");
    }

    public void estconsultarEvidencias() {
        List<CustodiaEvidencia> lista = evidenciaFacade.consultarEvidencias(null, "EK-09-
0001-00001-1", null, null, null, null, null, null, null, null, null, null, null);
        assertNotNull(lista);
        if (lista.get(0) != null)
            System.out.println("La funcionalidad de Consultar Evidencias trabaja
correctamente");
    }

    public void estelementoEsEvidenciaCustodiadaPor() {
        Integer resultado = evidenciaFacade.elementoEsEvidenciaCustodiadaPor(48, 249);
        assertTrue(resultado == 1);
    }
```

```
public void estestaFinalizadaEvidencia() {
        Integer resultado = evidenciaFacade.estaFinalizadaEvidencia(348);
        assertTrue(resultado == -1);
}

public void estevidenciaCustodiadaPor() {
        Boolean resultado = evidenciaFacade.evidenciaCustodiadaPor(225, 348);
        assertTrue(resultado);
}

public void testguardarOModificar() {
        Evidencia evidencia = evidenciaFacade.obtenerPorId(367);

        evidencia.setDescripcion("evidencia modificada");

        evidenciaFacade.guardarOModificar(evidencia);
        Evidencia evidenciaModificada = evidenciaFacade.obtenerPorId(367);
        assertEquals("evidencia modificada", evidenciaModificada.getDescripcion());
}

public void estiniciarCustodiaEvidencia() {
        Evidencia evidencia = new Evidencia();
        TipoDireccion tipoDireccion = nomencladorFacade.listarTipoDireccion().get(0);
```

```
        EstadoVenezolano estadoVenezolano =
nomencladorFacade.obtenerEstadoPorId(128);

        Municipio municipio =
nomencladorFacade.obtenerMunicipioPorIdEstadoIdMunicipio(128, 1699);

        Parroquia parroquia =
nomencladorFacade.obtenerParroquiaPorIdestadoIdMunicipioIdParroquia(1699, 5853);

        TipoResidencia tipoResidencia = nomencladorFacade.listarTipoResidencia().get(0);

        Direccion direccion = new Direccion(tipoDireccion, estadoVenezolano, municipio,
parroquia, tipoResidencia, "caserio1", "calle1", "2a", "2a", "2b");


        Funcionario funcionario = analisisInformacionFacade.obtenerFuncionarioPorId(225);

        Funcionario fotografo = analisisInformacionFacade.obtenerFuncionarioPorId(226);

        Funcionario funcionarioPresente =
analisisInformacionFacade.obtenerFuncionarioPorId(253);

        EvidenciaFuncionario evidenciaFuncionario = new EvidenciaFuncionario();

        evidenciaFuncionario.setEvidencia(evidencia);

        evidenciaFuncionario.setFuncionarioInt(funcionarioPresente);

        Elemento elemento = analisisInformacionFacade.obtenerElementoPorId(110);

        Expediente caso = investigacionPenalFacade.obtenerActaProcesalPorId(630);

        Persona persona = analisisInformacionFacade.obtenerPersonaPorId(169);


        evidencia.setFuncionario(funcionario);

        evidencia.setFotografo(fotografo);

        evidencia.setLugarColecto("Mi casa");

        evidencia.setFecha(new Date());
```

```
            evidencia.setFechaColecta(new Date());
            evidencia.setObjeto(elemento);
            evidencia.setDireccionColecto(direccion);
            evidencia.setCaso(caso);
            evidencia.setPersonaPropietario(persona);
            evidencia.setDescripcion("esto es para probar");
            evidencia.getFuncionariosPresentes().add(evidenciaFuncionario);

            evidenciaFacade.iniciarCustodiaEvidencia(evidencia);
        }

        public void estobtenerPorId(Integer id) {
            Evidencia evidencia = evidenciaFacade.obtenerPorId(367);
            assertNotNull(evidencia);
        }
}
```