# ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE LAS CAPAS DE LÓGICA DE NEGOCIO Y ACCESO A DATOS DEL SUBMÓDULO CADÁVERES, PERTENECIENTE AL MÓDULO DE INVESTIGACIÓN FORENSE DEL SISTEMA DE INVESTIGACIÓN E INFORMACIÓN POLICIAL.

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


**AUTORES:**

Eric Rodríguez Nadal

Luis Emilio Lorenzo Ballestero

**TUTOR:**

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**

Eric Rodríguez Nadal

**AUTOR**

Luis Emilio Lorenzo Ballestero

**TUTOR**

Ing. Susel Ruiz Durán

# AGRADECIMIENTOS

De Eric:

A Susel por estar siempre cuando se le necesitaba y por no perder la paciencia ni un solo segundo, gracias.

A mis padres por apoyarme siempre y por creer en mí, esta trabajo de diplomado es tan suyo como mío.

A Rosalba y Carlos por compartir el trabajo y darme ánimos cuando más los necesité.

A mis compañeros del equipo de trabajo de Investigación Forense.

A mi familia y amigos por su apoyo.

A todas las personas que de una forma u otra hicieron posible esta investigación, muchas gracias.

# DEDICATORIA

*De Eric:*

*A mis Padres.*

# RESUMEN

La actual situación delictiva en Venezuela ha llevado a la necesidad de modernizar el actual sistema con que cuenta el Cuerpo de Investigaciones Científicas Penales y Criminalísticas. Este trabajo contiene la investigación y desarrollo de las capas de Lógica de Negocio y Acceso a Datos para el submódulo Cadáveres del nuevo sistema. En él se incluyen el estudio del arte, análisis, diseño, implementación y estrategias de pruebas utilizadas, así como los distintos diagramas y tablas generados. Se presentan además un conjunto de conclusiones y recomendaciones que se consideró oportunas abordar.

# ABSTRACT

The current crime situation in Venezuela has led to the need to modernize the existing system in the *Cuerpo de Investigaciones Científicas Penales y Criminalísticas*. This work includes research and development of Business Logic Layer and Data Access Layer for the submodule *Cadáveres* of the new system. It includes the study of the state of the art technologies and tendences, analysis, design, implementation, used testing strategies and the different diagrams and tables generated. Finally, a set of conclusions and recommendations that were considered appropiate to address are presented.

# ÍNDICE DE CONTENIDOS

# ÍNDICE DE FIGURAS

# INTRODUCCIÓN

La República Bolivariana de Venezuela es actualmente escenario de una cruda realidad, en sus calles aumenta vertiginosamente el número de actos delictivos, siendo gran parte de estos, casos relacionados con la violencia, el uso de armas de fuego y narcóticos, lo cual implica que cada vez sean identificadas un mayor número de evidencias y que sea necesario el análisis de las mismas por métodos científicos, lo que exige de los cuerpos de justicia una mayor eficiencia e inmediatez en sus procesos.

El 2 de febrero de 1999, Hugo Rafael Chávez Frías asume la presidencia de la dicha nación, en medio de un ambiente de tensión política, crisis económica en todo el continente y una dura situación interna en el país. El inicio de la nueva magistratura enarbola la propuesta de refundar la República mediante el supremo recurso democrático de una Asamblea Nacional Constituyente, y dando cumplimiento a su promesa inicia el proceso para la elaboración de la nueva constitución. Atendiendo al delicado tema de la seguridad ciudadana y el considerable aumento de la actividad delictiva en el país andino, el nuevo presidente promulga el Decreto Ley de los "Órganos de Investigaciones Científicas, Penales y Criminalísticas", el cual se establece la creación del Cuerpo de Investigaciones Científicas, Penales y Criminalísticas (en lo adelante CICPC).

Transcurridos ya 5 años desde su creación el 15 de mayo de 2003, el CICPC aún no ha logrado eliminar el alto índice delictivo. Como es de esperarse, esta situación afecta en todos los sentidos la vida en el país, la tranquilidad ciudadana y el disfrute de los derechos sociales, haciendo que la actual presidencia dependa en gran medida de la eliminación de este flagelo.

La actual situación del CICPC no es la más adecuada para el proceso de investigación debido al gran volumen de información que se maneja, la dificultad para la obtención de estadísticas de un caso particular y el poco grado de fiabilidad de la información debido a que existen áreas que no cuentan con sistemas automatizados para el control de los procesos que realizan.

La Coordinación Nacional de Ciencias Forenses (en lo adelante CNCF) es una institución auxiliar de justicia adscrita al CICPC, la cual provee los servicios forenses a nivel nacional. Su principal función es servir como apoyo a los procesos de investigación, a la organización, administración y supervisión de todas las actividades forenses en servicio del trabajo policial, abarca todas las ramas de esta especialidad

y el personal que en ella labora cuenta con un alto grado de preparación, con los cuales pretenden garantizar un aporte óptimo, objetivo y fidedigno como medio de prueba en la administración de justicia.

La actual situación por la que atraviesa esta institución científica la limita en sus tareas y no permite el cabal e inmediato funcionamiento de la misma como órgano esclarecedor en los procesos judiciales e investigativos. Durante el estudio realizado en el CICPC se pudo determinar que los entes policiales no prestan la debida atención a las cuestiones forenses y no explotan todas las potencialidades que esta ciencia ofrece, en gran medida por desconocimiento de las mismas.

En la CNCF, durante la realización de una experticia, no llegan a manos del experto encargado de la misma, los documentos generados durante el proceso investigativo, los cuales pudieran ser de gran utilidad para la determinación de un resultado más confiable y enmarcado en el propósito de la investigación.

Las evidencias que se manejan en el CICPC son controladas a través de una cadena de custodia en la cual se registra quién es responsable de la evidencia y en qué momento, lo cual da fe de su autenticidad. Muchas veces, por no poseer un mecanismo automatizado para el control de esta cadena de custodia, se pierde esta información, por lo que muchas de las experticias que pudieran realizarse con estas evidencias dejan de aportar datos a la investigación al perder su validez legal.

El resultado de las experticias que se realizan en la CNCF muchas veces es requerido en más de una dependencia policial o institución legal, pero debido al mal manejo de estas gestiones y a la poca información que los especialistas reciben adjunta a la solicitud, estos resultados no son enviados a todos los lugares necesarios, por lo que deja de llegar información valiosa a entes participantes en la investigación.

La Coordinación Nacional de Ciencias Forenses es también la encargada de la recepción de todos los cadáveres encontrados por el CICPC, así como de realizarles las experticias que sean necesarias y la custodia de los mismos hasta la entrega al familiar o hasta su entierro por cuenta de la institución. Actualmente el control estadístico de muertes no cuenta con la calidad necesaria debido al descontrol de la información que arriba a las coordinaciones forenses, además de que los trabajos de cotejo y búsqueda de la identidad de los cadáveres se hace de forma manual, lo cual implica un mayor costo en personal y

tiempo, ambos valiosos recursos que podrían ser aprovechados de una mejor manera si existiesen mecanismos automatizados para la gestión y manejo de toda esta información.

El CICPC cuenta actualmente con un Sistema Integrado de Información Policial que centraliza la información de interés para las dependencias policiales adscritas a la institución, y a través del cual los cuerpos policiales del resto del país que tiene acceso a él, pueden realizar consultas para la obtención de datos durante un proceso investigativo. Dicho sistema no incluye soporte para el área de ciencias forenses, lo cual obviamente constituye un lastre para la organización y funcionamiento de todo el cuerpo de investigaciones.

El sistema existente además de estar desarrollado con tecnología ya obsoleta y fuera del mercado, no resuelve todas las necesidades del CICPC, y en muchos de los casos, su uso tiende a demorar el proceso de investigación, además de que las consultas en él realizadas no cuentan siempre con la inmediatez y calidad requeridas.

Concebido en el marco del ALBA y como parte de la colaboración Cuba-Venezuela se encuentra en fase de desarrollo un proyecto de software con el objetivo de sustituir el actual sistema cubriendo sus actuales funcionalidades, mejorando sus prestaciones e incorporando la solución al resto de las necesidades de la organización.

El nuevo sistema SIIPOL (acrónimo de Sistema de Investigación e Información Policial) también brindará servicios a la CNCF ofreciéndole la posibilidad de gestionar, procesar y consultar toda la información que en estas áreas se maneja. Se agregará además un submódulo para el manejo de toda la información relacionada con los cadáveres en el CICPC, con lo cual las coordinaciones forenses estarían ganando un valioso recurso para la realización de un trabajo más completo y controlado. Se espera con este trabajo contribuir a mejorar, organizar y acelerar el proceso investigativo en todo el Cuerpo de Investigaciones Científicas, Penales y Criminalísticas.

Debido a lo antes planteado se presenta el siguiente **problema científico**: ¿Cómo garantizar el cumplimiento de los requisitos funcionales y no funcionales asociados al submódulo Cadáveres perteneciente al módulo Investigación Forense del SIIPOL, desde el punto de vista de la lógica de negocio y el acceso a datos?

El **objeto de estudio** es el funcionamiento del módulo de Investigación Forense perteneciente al SIIPOL.

Y el **campo de acción** es el diseño e implementación de la capas de lógica de negocio y de acceso a datos del submódulo Cadáveres perteneciente al módulo de Investigación Forense del SIIPOL.

Como **objetivos generales** se persiguen:

1- Obtener una capa de Lógica de Negocio para el submódulo Cadáveres, que soporte los requisitos funcionales y no funcionales definidos para el mismo.

2- Obtener una capa de Acceso a Datos para el submódulo Cadáveres, que soporte los requisitos funcionales y no funcionales definidos para el mismo.

Los **objetivos específicos** son:

1- Obtener un modelo de Análisis para el submódulo Cadáveres.

2- Diseñar e implementar la capa de Lógica de Negocio para el submódulo Cadáveres.

3- Diseñar e implementar de la capa de Acceso a Datos para el submódulo Cadáveres.

4- Verificar el cumplimiento de los requisitos funcionales y no funcionales del submódulo.

Como **idea a defender** se plantea que: si se analiza, diseña e implementa una capa de Lógica de Negocio y una capa de Acceso a Datos para el submódulo Cadáveres, haciendo uso de una metodología de desarrollo adecuada y de buenas prácticas de diseño orientado a objetos, se logrará satisfacer los requisitos funcionales y no funcionales definidos para dicho submódulo.

Para dar cumplimiento a los objetivos se planificaron las siguientes **tareas investigativas**:

- Estudiar los procesos llevados a cabo en la CNCF.
- Estudiar el funcionamiento requerido para el Módulo Investigación Forense del SIIPOL.
- Investigar las herramientas definidas en el proyecto CICPC que permitan el desarrollo de este trabajo investigativo.

- Estudiar las especificaciones de los Casos de Uso del Sistema correspondientes al submódulo Cadáveres.
- Realizar diagramas de clases de diseño para cada Caso de Uso y diagramas de contrato entre paquetes.
- Implementar las capas de Lógica de Negocio y Acceso a Datos del submódulo Cadáveres.
- Procesar los resultados de las pruebas realizadas al submódulo para verificar el cumplimiento de los requisitos funcionales y no funcionales asociados al mismo.

El documento está estructurado en tres capítulos:

En el Capítulo 1 se investigan soluciones similares y se describen la metodología y herramientas que se van a utilizar para desarrollar las capas de lógica de negocio y acceso a datos.

En el Capítulo 2 se muestran todos los artefactos generados al realizar el análisis, diseño e implementación del submódulo Cadáveres.

En el Capítulo 3 se dan a conocer los resultados de las pruebas realizadas al submódulo Cadáveres y los tipos de prueba de los cuales se obtuvieron los resultados.

# CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

## 1.1. INTRODUCCIÓN

Con este capítulo se realiza una búsqueda para hallar otros sistemas que tengan las mismas funcionalidades o similitudes con el que se va a desarrollar. Se tratará la metodología de desarrollo de software a usar y se explica su funcionamiento, así como las tecnologías y herramientas con las que se van a desarrollar las capas de lógica de negocio y acceso a datos del submódulo Cadáveres, abordando las principales características y beneficios de las mismas.

## 1.2. SOFTWARE DE GESTIÓN POLICIAL

El avance de las Ciencias Informáticas ha sido el pilar fundamental de la creciente informatización de los organismos policiales. Esto se debe a que la información generada por los mismos se centra en un sistema al cual se puede acceder desde cualquier lugar y que une a todas las entidades que intervienen en un caso, haciendo más rápida la comunicación, envío y búsqueda de documentos que ayudan a desarrollar una investigación policial.

Actualmente en Argentina, en la provincia Mendoza, se realiza un proyecto con el cual algunos organismos como Migraciones, Poder judicial, Colegio de Escribanos, Reparticiones Nacionales e Internacionales, Obras Sociales y Bancos puedan disponer de una base de datos biométrica, a fin de poder identificar unívocamente a cada uno de los ciudadanos de la provincia de Mendoza, de manera ágil y confiable. (1)

También en la provincia de Neuquén la policía cuenta con un departamento de informática en el cual se administran los procesos electrónicos de información y documentación electrónica policial, controlando y auditando la seguridad operativa de los mismos, así como los recursos materiales y tecnológicos provistos para el desarrollo y explotación de la informática policial, su utilización y mantenimiento, asesorando constantemente en lo referente a la distribución equitativa de los mismos, y los recursos humanos profesionales o técnicos de su área, proponiendo las necesidades de incorporación y capacitación al personal policial que desarrolla tareas relacionadas a la informática. (2)

En Mérida, España, existe un sistema policial encargado de manejar toda la información referente a denuncias, multas, armas, registros. En Brasil existe el Sistema Informatizado de Acompañamiento Criminal (SIAC) en la sub-área de la 128ª Compañía de la Policía Militar del Estado de Minas Gerais. Es un programa informático, desarrollado empíricamente en la sede de la Fracción Policía Militar que ha traído resultados palpables en lo que se refiere a las estadísticas criminales y que hoy ha pasado a ser adoptado por el Comando de la Policía de la Capital (CPC) como política de comando para todas las Compañías con responsabilidad territorial de Belo Horizonte.

## 1.3. EL CUERPO DE INVESTIGACIONES CIENTÍFICAS, PENALES Y CRIMINALÍSTICAS

Con motivo de incluir la CNCF en el nuevo SIIPOL se decidió informatizar el tratamiento de cadáveres para que la información generada por los especialistas forenses vinculados con esta rama esté disponible en el nuevo sistema.

El Cuerpo de Investigaciones Científicas Penales y Criminalísticas (CICPC) es el organismo rector del enfrentamiento al delito en La República Bolivariana de Venezuela, integra las investigaciones de tipo científicas y criminalísticas sobre la base de las leyes vigentes y pone los resultados a disposición de los órganos judiciales del Ministerio Público, responsable legal de la investigación. El CICPC tiene como visión convertirse en la institución de obligada consulta a nivel nacional en el enfrentamiento al delito por sus capacidades científicas y su equipamiento técnico.

Esta institución cuenta con el Sistema Integrado de Información Policial (SIIPOL) para centralizar la información de interés para los cuerpos policiales, mediante el cual los funcionarios, cuerpos policiales estadales y municipales a lo largo de todo el país que tienen acceso, pueden consultar personas buscadas por la justicia, antecedentes penales, delictivos, así como vehículos robados; no siendo así con la información generada por los servicios forenses. Este sistema está desarrollado sobre una tecnología actualmente obsoleta: sobre el lenguaje de programación Adabas-Natural y los servidores de base datos SUN 6500 ya fuera del mercado.

SIIPOL posibilita el intercambio de información entre las diferentes áreas y entidades del CICPC a fin de llegar a un resultado exitoso en la investigación; sin embargo el SIIPOL no resuelve todas las necesidades

de la institución, llegando incluso a demorar los procesos de investigación al complejizarse su uso y brindar información que no siempre cuenta con la calidad y la inmediatez necesaria.

Cuba y Venezuela, en el marco de colaboración de los países del ALBA, se encuentran involucrados en el proyecto de modernización del CICPC, el cual abarca transformación organizacional, equipamiento con tecnología avanzada, capacitación de personal y creación de un sistema de gestión e información policial.

### 1.3.1. PROCESOS DE LA COORDINACIÓN NACIONAL DE CIENCIAS FORENSES

La Coordinación Nacional de Ciencias Forenses está estructurada en cuatro direcciones, las cuales a su vez se dividen en divisiones, departamentos y áreas.

La Dirección de Patología Forense es la principal área que se dedica al tratamiento de cadáveres, está compuesta por tres divisiones: Anatomía Patológica, Antropología y Odontología.

La división de Anatomía Patológica es la encargada de proporcionar a los órganos instructores la causa de muerte y demás elementos que sean solicitados por estos o que puedan ser útiles en la investigación. A esta División le brindan apoyo los departamentos de Radiología Forense y Fotografía, las áreas de Histología Forense y Morgue.

La División de Odontología se encarga de la identificación de cadáveres a través de la estructura odontológica, así como a la determinación de la edad cronológica en individuos vivos, mientras que la división de Antropología participa en la identificación de personas a partir de estudios antropométricos y antropológicos, en la determinación de edad cronológica de individuos vivos y en el estudio de caracteres físico morfológicos.

La Dirección de Medicina Forense se encarga, a solicitud del Ministerio Público o de los organismos instructores de casos, de hacer el reconocimiento médico de los lesionados con el fin de determinar la ocurrencia o no de hechos punibles y de aportar elementos de validez para el proceso judicial. Esta Dirección es la primera que se enfrenta al proceso de tratamiento de los cadáveres, pues tiene la responsabilidad de hacer el levantamiento, ya sea en el sitio de la muerte o en cualquier otro lugar al cual se haya trasladado el cadáver sin que haya sido reconocido por un Médico forense.

La Dirección de Toxicología se divide en varias áreas especializadas en determinados tipos de análisis, con el fin de aportar las pruebas toxicológicas que determinen los componentes de las muestras en estudio y finalmente emitir un informe pericial.

La Dirección de Diagnóstico Mental Forense se dedica a la evaluación de personas que están vinculadas a hechos punibles ya sea como víctimas, imputados o presupuestos imputados para emitir un informe de la situación y salud mental de la persona en un momento dado de su vida, esta Dirección está compuesta por la División de Psiquiatría Forense, la División de Psicología Forense, División de Análisis Social Forense, el Área de Neurología Forense y los Departamentos de Psiquiatría Forense a Nivel de Estados.

## 1.4. EL SISTEMA DE INVESTIGACIÓN E INFORMACIÓN POLICIAL

En el marco de las relaciones entre Cuba y Venezuela por la colaboración de los países del ALBA, se ha concebido el proyecto de Modernización del CICPC, el cual se centra en la construcción de un nuevo sistema informático que sustituya las prestaciones del actual SIIPOL, mejore e incluya nuevas funcionalidades, contribuya a la disminución de los tiempos de respuesta de las investigaciones de cada área, sin obviar la Coordinación Nacional de Ciencias Forenses, para la cual se incorporará el módulo Investigación Forense, contando el mismo con el submódulo Cadáveres, el que centralizará las tareas de desarrollo investigativo y control a nivel nacional de la actividad de la CNCF en lo relativo a los cadáveres encontrados y procesados.

### 1.4.1. SUB-MÓDULO CADÁVERES. REQUISITOS.

Es este subsistema se procesa el tratamiento de los cadáveres, agrupándose las funcionalidades necesarias para que los especialistas forenses manejen toda la información obtenida. En él se realiza un seguimiento completo al cadáver, desde el levantamiento, la autopsia hasta los estudios complementarios. Al finalizar la investigación el cadáver es entregado a los familiares si es reclamado, en caso contrario es sepultado. Para ver el diagrama de casos de uso del subsistema Cadáveres dirigirse al Anexo 1.

**Breve descripción de Casos de Uso**

| | |
|---|---|
| **Nombre del CU** | Registrar Levantamiento del Cadáver. |

| | |
|---|---|
| **Actor** | Funcionario de Guardia. |
| **Descripción** | El caso de uso se inicia cuando el actor accede a la opción de Registrar un Levantamiento de Cadáver. El sistema crea un nuevo Expediente Tanatológico, asignándole de forma automática un número de Expediente Tanatológico, único, que sirve para identificar al expediente y a todos los documentos, informes, comunicaciones asociados al procesamiento del cadáver en cuestión. Asocia la persona al Expediente Tanatológico recién creado y se genera, automáticamente, una Solicitud de Experticia asociada al Expediente Tanatológico. El actor puede imprimir o exportar a formato PDF la Solicitud generada. |
| **Referencia** | RF. Validar la integridad de los datos introducidos por el usuario.<br>RF. Generar un número único para cada Expediente Tanatológico.<br>RF. Incluir Expediente Tanatológico.<br>RF. Asociar Levantamiento del Cadáver a Expediente Tanatológico.<br>RF. Incluir y asociar al Expediente Tanatológico una Solicitud de Experticia Identificativa.<br>RF. Incluir y asociar al Expediente Tanatológico una Solicitud de Confirmación/Búsqueda de identidad.<br>RF. Mantener informado al usuario del resultado de las operaciones.<br>RF. Imprimir o exportar a formato a PDF.<br>RC. (Seguridad) La información en el sistema no puede ser eliminada. |

| | |
|---|---|
| **Nombre del CU** | Gestionar Entrevista Antropológica. |
| **Actor** | Gestor de Entrevistas. |
| **Descripción** | El caso de uso inicia cuando el actor accede a la opción de realizar una acción sobre una Entrevista a Familiar seleccionada. En caso que el actor haya seleccionado la opción de incluir una Entrevista, el sistema crea una nueva Entrevista a Familiar, dando la posibilidad de introducir los datos de la misma. En caso que el actor haya seleccionado la opción de ver los detalles de una Entrevista a Familiar, el sistema muestra el contenido de la misma, así como un listado de los familiares que han aportado información. El actor puede imprimir o exportar a formato PDF la Entrevista a Familiar. En caso que el actor haya seleccionado la opción de modificar una Entrevista a Familiar, el sistema muestra el contenido de la misma, permitiendo su edición, así como asociar un nuevo familiar (Persona) que aporta datos. |
| **Referencia** | RF. Gestionar Resultados de la Autopsia.<br>RF. Imprimir o exportar a PDF los Resultados de la Autopsia.<br>RF. Validar la integridad de los datos introducidos por el usuario.<br>RF. Mantener informado al usuario del resultado de las operaciones.<br>RC. (Seguridad) La información en el sistema no puede ser eliminada. |

| | |
|---|---|
| **Nombre del CU** | Gestionar Diagrama Corporal Antropológico. |
| **Actor** | Gestor de Informes. |
| **Descripción** | El caso de uso se inicia cuando el actor selecciona la opción que le permite realizar una acción sobre un Diagrama Corporal. El actor puede incluir, ver y modificar el mismo. En caso de que seleccione la opción de incluir un Diagrama Corporal, el sistema dará la posibilidad de ubicar en un diagrama vacío, los elementos antropológicos que el actor considere necesarios. Si el actor elige la opción de ver el Diagrama Corporal, el sistema mostrará todos los elementos antropológicos contenidos en el mismo. Si el actor elige la opción de modificar el Diagrama Corporal, el sistema mostrará todos los elementos antropológicos contenidos en el mismo, permitiendo eliminarlos y adicionar nuevos elementos, y una vez realizados los cambios, guardará las modificaciones. |
| **Referencia** | RF. Gestionar Diagrama Corporal Antropológico.<br>RF. Imprimir o exportar a PDF las vistas del Diagrama Corporal Antropológico.<br>RF. Validar la integridad de los datos introducidos por el usuario.<br>RF. Mantener informado al usuario del resultado de las operaciones.<br>RC. (Seguridad) La información en el sistema no puede ser eliminada. |

Requisitos suplementarios relevantes a tener en cuenta, para más información dirigirse al documento de Requisitos Suplementarios del proyecto CICPC**:**

Funcionalidad

- El sistema permitirá la inclusión y manejo de datos estructurados y no estructurados (archivos, imágenes, entre otros).

Seguridad:

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

Rendimiento:

- El sistema procesará una transacción en un tiempo no mayor a 1 segundo, a partir de que la petición se recibe en el servidor, es decir, esta cifra no incluye los retardos por concepto de tráfico de red.
- El sistema respetará buenas prácticas de programación para incrementar el rendimiento en operaciones costosas para la máquina virtual como la creación de objetos. Se deberá usar siempre que sea posible el patrón Singleton, destruir referencias que ya no estén siendo usadas, optimizar el trabajo con cadenas, entre otras buenas prácticas que ayudan a mejorar el rendimiento.
- El sistema identificará por cada caso de uso aquellas operaciones que impliquen un elevado nivel de procesamiento en la base de datos y usará procedimientos almacenados para manejarlas.

## 1.5. METODOLOGÍA, LENGUAJES Y HERRAMIENTAS DE DESARROLLO

### 1.5.1. METODOLOGÍA. PROCESO DE DESARROLLO UNIFICADO

El Proceso Unificado de Desarrollo (RUP) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado (UML), constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

El RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

El RUP está basado en 5 principios clave que son:

Adaptar el proceso: El proceso deberá adaptarse a las características propias del proyecto y su organización. El tamaño del mismo, así como su tipo o las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto.

Balancear prioridades: Los requerimientos de los diversos participantes pueden ser diferentes, contradictorios o disputarse recursos limitados. Debe encontrarse un balance que satisfaga los deseos de todos. Debido a este balanceo se podrán corregir desacuerdos que surjan en el futuro.

Demostrar valor iterativamente: Los proyectos se entregan, aunque sea de un modo interno, en etapas iteradas. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados.

Elevar el nivel de abstracción: Este principio dominante motiva el uso de conceptos reutilizables tales como patrón del software, lenguajes 4GL o marcos de referencia (*frameworks*) por nombrar algunos. Esto evita que los ingenieros de software vayan directamente de los requisitos a la codificación de software a la medida del cliente, sin saber con certeza qué codificar para satisfacer de la mejor manera los requerimientos y sin comenzar desde un principio pensando en la reutilización del código. Un alto nivel de abstracción también permite discusiones sobre diversos niveles y soluciones arquitectónicas. Éstas se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con el lenguaje UML.

Enfocarse en la calidad: El control de calidad no debe realizarse al final de cada iteración, sino en todos los aspectos de la producción. El aseguramiento de la calidad forma parte del proceso de desarrollo y no de un grupo independiente.

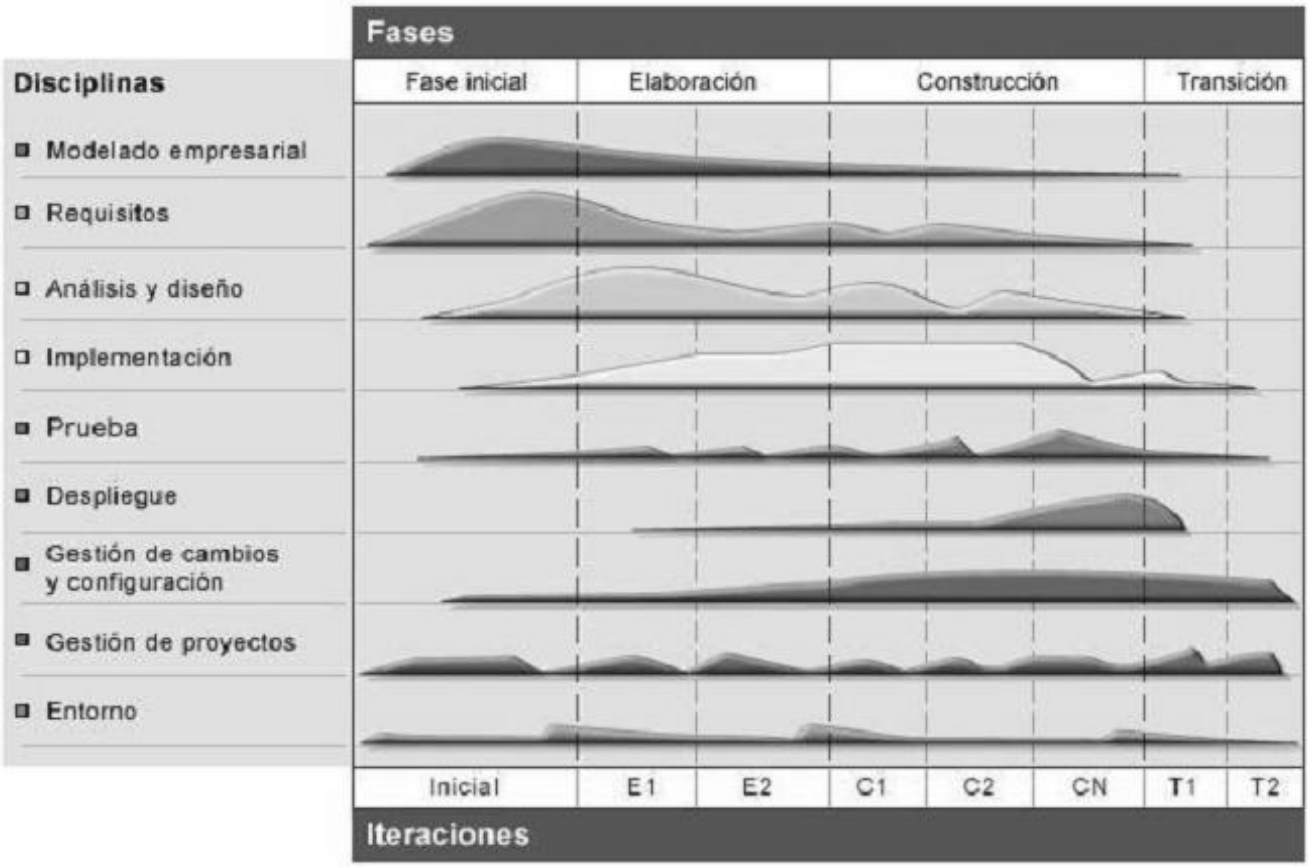



**Fig. 1 Ciclo de vida de RUP.**

El ciclo de vida RUP es una implementación del Desarrollo en espiral. Fue creado ensamblando los elementos en secuencias semi-ordenadas. El ciclo de vida organiza las tareas en fases e iteraciones. RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades. En la Figura se muestra cómo varía el esfuerzo asociado a las disciplinas según la fase en la que se encuentre el proyecto RUP.

Las primeras iteraciones (en las fases de Inicio y Elaboración) se enfocan hacia la comprensión del problema y la tecnología, la delimitación del ámbito del proyecto, la eliminación de los riesgos críticos, y al establecimiento de una línea base de la arquitectura.

Durante la fase de inicio las iteraciones hacen mayor énfasis en actividades de modelado del negocio y de requerimientos. En la fase de elaboración, las iteraciones se orientan al desarrollo de la línea base de la arquitectura, abarcan más los flujos de trabajo de requerimientos, modelo de negocios (refinamiento), análisis, diseño y una parte de implementación orientado a la línea base de la arquitectura. En la fase de construcción, se lleva a cabo la construcción del producto por medio de una serie de iteraciones.

Para cada iteración se seleccionan algunos Casos de Uso, se refina su análisis y diseño y se procede a su implementación y pruebas. Se realiza una pequeña cascada para cada ciclo. Se realizan tantas iteraciones hasta que se termine la implementación de la nueva versión del producto.

En la fase de transición se pretende garantizar que se tiene un producto preparado para su entrega a la comunidad de usuarios. Como se puede observar en cada fase participan todas las disciplinas, pero que dependiendo de la fase, el esfuerzo dedicado a una disciplina varía.

RUP se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. Incluye artefactos (que son los productos tangibles del proceso como por ejemplo, el modelo de casos de uso, el código fuente, etc.) y roles (papel que desempeña una persona en un determinado momento, una persona puede desempeñar distintos roles a lo largo del proceso).

**Dirigido por casos de uso**

Un sistema de software se crea para servir a sus usuarios. Por lo tanto, para construir un sistema exitoso se debe conocer qué es lo que quieren y necesitan los usuarios prospectos. El término usuario se refiere no solamente a las personas que lo utilizaran, sino también a otros sistemas. En este contexto, el término usuario representa algo o alguien que interactúa con el sistema por desarrollar.

Un caso de uso es una pieza en la funcionalidad del sistema que le da al usuario un resultado de valor, los mismos capturan los requerimientos funcionales. Es su totalidad describen la funcionalidad completa del sistema. Este modelo reemplaza la tradicional especificación funcional del sistema. Una especificación funcional tradicional se concentra en responder la pregunta: ¿Qué se supone que el sistema debe hacer? La estrategia de casos de uso puede ser definida agregando tres palabras al final de la pregunta: ¿por cada usuario? Estas tres palabras tienen una implicación importante, se fuerza a pensar en términos del valor a los usuarios y no solamente en términos de las funciones que sería bueno que tuviera.

Sin embargo, los casos de uso no son solamente una herramienta para especificar los requerimientos del sistema, también dirigen su diseño, implementación y pruebas, esto es, dirigen el proceso de desarrollo. Aún y cuando los casos de uso dirigen el proceso, no son elegidos de manera aislada. Son desarrollados a la par con la arquitectura del sistema, esto es, los casos de uso dirigen la arquitectura del sistema y la arquitectura del sistema influencia la elección de ellos. Por lo tanto ambos maduran conforme avanza el ciclo de vida.

**Centrado en la arquitectura**

El papel del arquitecto de sistemas es similar en naturaleza al papel que el arquitecto desempeña en la construcción de edificios. El edificio se mira desde diferentes puntos de vista: estructura, servicios, plomería, electricidad, etc. Esto le permite al constructor ver una radiografía completa antes de empezar a construir. Similarmente, la arquitectura en un sistema de software es descrita como diferentes vistas del sistema que está siendo construido. El concepto de arquitectura de software involucra los aspectos estáticos y dinámicos más significativos del sistema. La arquitectura surge de las necesidades de la empresa, tal y como las interpretan los usuarios y otros involucrados, y tal y como están reflejadas en los casos de uso. Sin embargo, también está influenciada por muchos otros factores, como la plataforma de software en la que se ejecutará, la disponibilidad de componentes reutilizables, consideraciones de instalación, sistemas legados, requerimientos no funcionales.

La arquitectura es la vista del diseño completo con las características más importantes hechas más visibles y dejando los detalles de lado. Ya que lo importante depende en parte del criterio, el cual a su vez viene con la experiencia, el valor de la arquitectura depende del personal asignado a esta tarea. Sin embargo, el proceso ayuda al arquitecto a enfocarse en las metas correctas, como claridad y flexibilidad en los cambios futuros. ¿Cómo se relacionan los casos de uso con la arquitectura? Cada producto tiene función y forma. Uno sólo de los dos no es suficiente. Estas dos fuerzas deben estar balanceadas para obtener un producto exitoso. En este caso, función corresponde a los casos de uso y forma a la arquitectura. Existe la necesidad de intercalar entre casos de uso y arquitectura. Por una parte, los casos de uso deben, cuando son realizados, acomodarse en la arquitectura. Por otra parte, la arquitectura debe proveer espacio para la realización de todos los casos de uso, hoy y en el futuro. En la realidad, ambos, arquitectura y casos de uso deben evolucionar en paralelo.

**Iterativo e Incremental**

Desarrollar un producto de software comercial es una tarea enorme que puede continuar por varios meses o años. Es práctico dividir el trabajo en pequeñas partes o mini-proyectos. Cada mini-proyecto es una iteración que finaliza en un incremento. Las iteraciones se refieren a pasos en el flujo de trabajo, los incrementos se refieren a crecimiento en el producto. Para ser más efectivo, las iteraciones deben estar controladas, esto es, deben ser seleccionadas y llevadas a cabo de una manera planeada.

Los desarrolladores basan su selección de qué van a implementar en una iteración en dos factores. Primero, la iteración trata con un grupo de casos de uso que en conjunto extienden la usabilidad del producto. Segundo, la iteración trata con los riesgos más importantes. Las iteraciones sucesivas construyen los artefactos del desarrollo a partir del estado en el que fueron dejados en la iteración anterior. En cada iteración, los desarrolladores identifican y especifican los casos de uso relevantes, crean el diseño usando la arquitectura como guía, implementan el diseño en componentes y verifican que los componentes satisfacen los casos de uso. Si una iteración cumple sus metas (y usualmente lo hace) el desarrollo continúa con la siguiente iteración. Cuando la iteración no cumple con sus metas, los desarrolladores deben revisar sus decisiones previas y probar un nuevo enfoque.

La metodología RUP es más apropiada para proyectos grandes (aunque también para pequeños), dado que requiere un equipo de trabajo capaz de administrar un proceso complejo en varias etapas. En proyectos pequeños, es posible que no se puedan cubrir los costos de dedicación del equipo de profesionales necesarios. (3)

### 1.5.2. PLATAFORMA DE DESARROLLO. JAVA 2 ENTERPRISE EDITION.

Java 2 Enterprise Edition (J2EE) es una plataforma de programación para desarrollar y ejecutar *softwares* de aplicaciones en Lenguaje de programación Java con arquitectura de N niveles distribuidos, basándose ampliamente en componentes de *software* modulares ejecutándose sobre un servidor de aplicaciones. La plataforma J2EE está definida por una especificación. Similar a otras especificaciones de la Comunidad de procesos Java, J2EE es también considerada informalmente como un estándar debido a que los suministradores deben cumplir ciertos requisitos de conformidad para declarar que sus productos son conformes a J2EE; no obstante sin un estándar de ISO o ECMA.

Esta plataforma contiene numerosas especificaciones de API, tales como JDBC, RMI, correo electrónico, JMS, Servicios Web, XML, etc. y define cómo coordinarlos. Java EE también configura algunas especificaciones únicas para J2EE para componentes. Estas incluyen Enterprise JavaBeans, *servlets*, *portlets* (siguiendo la especificación de Portlets Java), Java Server Pages (Páginas de servidor java) y varias tecnologías de servicios *web*. Esto permite al desarrollador crear una Aplicación de Empresa portable entre plataformas y escalable, a la vez que integrable con tecnologías anteriores.

Otras de sus ventajas son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados, lo cual implica que los desarrolladores no tienen que preocuparse por funciones de bajo nivel y pueden concentrarse más en las funciones asociadas con el negocio de la aplicación.

Uno de los beneficios de Java EE como plataforma es que es posible empezar con poco o ningún coste. La implementación Java EE de Sun Microsystems puede ser descargada gratuitamente, y hay muchas herramientas de código abierto disponible para extender la plataforma o para simplificar el desarrollo.

Ejemplos de herramientas de desarrollo Java de código abierto de terceras partes son:

- NetBeans IDE, un IDE basado en Java.
- La plataforma Eclipse, un IDE basado en Java.
- Expand, un *plugin* (módulo) de Eclipse, para desarrollo rápido.
- Jedit, de código abierto, un IDE basado en Java.
- Apache Software Foundation Apache Ant, una herramienta de construcción automática.
- Apache Software Foundation Apache Maven, una herramienta de construcción automática y gestión de dependencias.
- JUnit, un *framework* para Pruebas de unidad automatizadas.
- Apache Software Foundation Apache Tomcat, un contenedor *web* de Servlet/JSP.
- Jetty, un servidor web y un contenedor web Servlet/JSP.

- Struts, un framework para desarrollar aplicaciones web EE conforme al modelo MVC.
- OpenXava, un framework de código abierto para desarrollo fácil de aplicaciones de negocio J2EE.
- JDeveloper, un IDE basado en Java y desarrollado por Oracle.
- JBuilder, desarrollado por Borland.
- Java Server Faces, un *framework* para desarrollar aplicaciones *web* EE conforme al modelo MVC, desarrollado por Sun.

Actualmente esta plataforma acapara gran parte de la comunidad de desarrolladores a nivel mundial dada la versatilidad de la misma y las enormes ventajas que su uso ofrece, cuenta con una amplia bibliografía y la cantidad de aplicaciones empresariales desarrolladas con esta tecnología aumenta vertiginosamente. (4)

### 1.5.3. HERRAMIENTA CASE. VISUAL PARADIGM.

Las herramientas CASE (Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y dinero. Estas herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software, en tareas como el proceso de realizar un diseño del proyecto, calculo de costes, implementación de parte del código automáticamente a partir del diseño dado, compilación automática, documentación o detección de errores entre otras.

Visual Paradigm es una herramienta CASE para modelado UML muy potente y de fácil uso. Permite dibujar todo tipo de diagramas UML, revertir código fuente a modelos UML, generar código fuente desde los diagramas UML, y mucho más.

Esta herramienta soporta hasta la fecha UML 3.2 y BPMN, permite realizar ingeniería tanto directa como inversa, a partir de un modelo relacional es capaz de desplegar todas las clases asociadas a las tablas en Java, C++, DotNet Exe/dll, XML, XML Schema, y Corba IDL. Para gestionar la persistencia y el mapeo de estas clases con la base de datos utiliza Hibernate para Java y NHibernate en el caso de un proyecto .Net. Soporta múltiples usuarios trabajando sobre el mismo proyecto; genera la documentación del proyecto

automáticamente en varios formatos como Web o .pdf, permite el control de versiones y ofrece soporte para Rational Rose e integración con Microsoft Visio. Incluye un kit para traducirlo a otros idiomas e integración con BEA WebLogic Workshop. (5)

### 1.5.4. IDE DE DESARROLLO. ECLIPSE.

Eclipse es un IDE (Entorno de Desarrollo Integrado) que, según el proyecto que actualmente lo desarrolla ha sido calificado como “una especie de herramienta universal, un IDE abierto y extensible para todo y nada en particular”. En un principio Eclipse fue desarrollado por IBM para luego ser lanzado a la comunidad de software libre, actualmente existe un proyecto sin ánimos de lucro llamado Fundación Eclipse encargado de su desarrollo el cual está compuesto por más de 20 prestigiosas compañías entre las cuales se encuentran Borland, Adobe, Intel, HP, Oracle, Google, MySQL, Nokia, Motorola, Zend entre otras, todas estas aún lideradas por IBM.

Se caracteriza por ser un IDE de código abierto y multiplataforma, que goza de una popularidad tremenda en la comunidad de desarrollo debido a su versatilidad y posibilidades de ampliación. El hecho de tener un código abierto a los desarrolladores ha facilitado la aparición de las mas disimiles extensiones y la integración de variadas herramientas de desarrollo.

La base para Eclipse es la Plataforma de Cliente Enriquecido (del inglés Rich Client Plataform RCP) constituida por los siguientes componentes:

- Plataforma principal, inicio de Eclipse, ejecución de *plugins*.
- OSGi, plataforma para bundling estándar.
- El Standard Widget Toolkit (SWT), un *widget toolkit* portable.
- JFace, manejo de archivos, manejo de texto, editores de texto.
- El Workbench de Eclipse, vistas, editores, perspectivas, asistentes.

Los *widgets* de Eclipse están implementados por una herramienta de *widget* para Java llamada SWT, a diferencia de la mayoría de las aplicaciones Java, que usan las opciones estándar Abstract Window

Toolkit (AWT) o Swing. La interfaz de usuario de Eclipse también tiene una capa GUI intermedia llamada JFace, la cual simplifica la construcción de aplicaciones basada en SWT.

A diferencia de otros entornos monolíticos donde todas las funcionalidades están ya incluidas, las necesite el usuario o no, el IDE de Eclipse emplea plugins para proporcionar todas sus funcionalidades, contando así con una plataforma ligera para componentes de software. Además es posible extender el IDE para el trabajo con otros lenguajes de programación como son C/C++, Perl, PHP, Phyton y otros. La arquitectura plugin permite escribir cualquier extensión deseada en el ambiente, actualmente Eclipse cuenta con innumerables herramientas de gran utilidad para el desarrollo de software que pueden ser agregadas o eliminadas a conveniencias.

En cuanto a las aplicaciones clientes, eclipse provee al programador con frameworks muy ricos para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de software, aplicaciones web, etc. Por ejemplo, GEF (Graphic Editing Framework - Framework para la edición gráfica) es un plugin de Eclipse para el desarrollo de editores visuales que pueden ir desde procesadores de texto hasta editores de diagramas UML, interfaces gráficas para el usuario (GUI), etc. Dado que los editores realizados con GEF “viven” dentro de Eclipse, además de poder ser usados conjuntamente con otros plugins, hacen uso de su interfaz gráfica personalizable y profesional.

El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código. El IDE también hace uso de un espacio de trabajo, en este caso un grupo de metadato en un espacio para archivos plano, permitiendo modificaciones externas a los archivos en tanto se refresque el espacio de trabajo correspondiente.

Eclipse es soportado además por varios sistemas operativos, Linux, Windows, Solaris (SPARC/GTK 2) y Mac OSX – Mac/Carbon y su última versión, la 3.3.1.1 fue liberada el 23 de octubre de 2007.

### 1.5.5. FRAMEWORK (MARCO DE TRABAJO) PARA LÓGICA DE NEGOCIO. SPRING (PRIMAVERA)

Spring es un *framework* de código abierto concebido para el desarrollo sobre la plataforma Java que interviene en todas las capas arquitectónicas de una aplicación J2EE, brinda soporte a Java Server Faces,

y se integra con varios frameworks de acceso a datos, formando así una poderosa herramienta para el desarrollo de las aplicaciones empresariales.

La primera versión fue escrita por Rod Johnson, quien lo lanzó primero con la publicación de su libro Expert One-on-One Java EE Design and Development (Wrox Press, octubre 2002). También hay una versión para la plataforma .NET, Spring.net. El framework fue lanzado inicialmente bajo Apache 2.0 License en junio de 2003. El primer gran lanzamiento hito fue la versión 1.0, que apareció en marzo de 2004 y fue seguida por otros hitos en septiembre de 2004 y marzo de 2005

A pesar de que Spring Framework 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. 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.

Mientras que las características fundamentales de este framework pueden emplearse en cualquier aplicación hecha en Java, existen muchas extensiones y mejoras para construir aplicaciones basadas en web por encima de la plataforma empresarial de Java (Java Enterprise Plataform).

Spring Framework hizo que aquellas técnicas que resultaban desconocidas para la mayoría de programadores se volvieran populares en un periodo muy corto de tiempo. El ejemplo más notable es la inversión de control. En el año 2004, Spring disfrutó de unas altísimas tasas de adopción y al ofrecer su propio framework de programación orientada a aspectos (aspect oriented programming, AOP) consiguió hacer más popular su paradigma de programación en la comunidad Java

En 2005 Spring superó las tasas de adopción del año anterior como resultado de nuevos lanzamientos y más características fueron añadidas. El foro de la comunidad formada alrededor de Spring Framework (The Spring Forum) que arrancó a finales de 2004 también ayudó a incrementar la popularidad del framework y desde entonces ha crecido hasta llegar a ser la más importante fuente de información y ayuda para sus usuarios.

Promueve el bajo acoplamiento de las clases que conforman la solución a través de la inversión de control (IoC) y está concebido para evitar la dependencia a su código base de las aplicaciones que lo usan, siendo posible la separación del framework de la aplicación sin demasiadas complicaciones. Soporta

además la Programación Orientada a Aspectos (AOP) que complementa la Programación Orientada a Objetos (POO), proponiendo otra manera de pensar sobre la estructura de una aplicación. Mientras esta ultima descompone las aplicaciones en una jerarquía de objetos, la AOP la descompone en aspectos o preocupaciones. Dichas preocupaciones se convierten en servicios del sistema, separados de la lógica de negocio y que se ejecutan de manera transversal a la funcionalidad base, lo que proporciona una definición de las responsabilidades superior. Spring constituye en principio un contenedor que se encarga de gestionar y administrar el ciclo de vida y configuración de las clases de la aplicación.

Los principales módulos de este framework son:

Core: Como indica su nombre constituye en núcleo del framework. En él se implementan las funciones esenciales que permiten su funcionamiento y otras que lo caracterizan como la técnica de Inversión de Control (IoC).

Context: Proporciona herramientas para acceder a los beans y da soporte a propagación de eventos, resource bundles, carga de recursos y creación transparente de contextos por parte de los contenedores.

DAO: Proporciona una capa de abstracción JDBC (Java Database Connectivity, o conectividad a base de datos java) y una forma de administración de las transacciones de la aplicación.

ORM: Provee capas de integración para APIs de mapeo objeto-relacional.

AOP: Proporciona una implementación de programación orientada a aspectos, permitiendo definir puntos de corte e interceptores.

Web: Provee de características de integración orientadas a la web, como funcionalidad multipartes, inicialización de contextos mediante servlet listeners (escuchadores servlet) y un contexto de aplicación orientada a la web. También permite integrar de forma sencilla otros frameworks como Struts, JSF o WebWork.

Spring MVC: Provee una implementación Modelo-Vista-Controlador que permite el uso del resto de funcionalidades del Spring Framework.

Spring es actualmente uno de los frameworks que mejor documentación oficial tiene, además de una gran comunidad de desarrolladores asiduos y un soporte de excelente calidad. Sus características permiten además la posibilidad de agregarle funcionalidades y extender las ya existentes.

### 1.5.6. FRAMEWORK PARA ACCESO A DATOS. HIBERNATE.

Hibernate es una herramienta para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional y el modelo orientado a objetos de una aplicación mediante archivos declarativos (XML) que permiten establecer relaciones. Es una herramienta ORM completa que ha conseguido en un tiempo record una excelente reputación en la comunidad de desarrollo posicionándose claramente como uno de los productos OpenSource (Código Abierto) líderes en este campo gracias a sus prestaciones, buena documentación y estabilidad. Es valorado por muchos como una solución superior a productos comerciales dentro de su mismo enfoque, siendo una muestra de su reputación y soporte la reciente integración dentro del grupo JBoss que seguramente generará iniciativas muy interesantes para el uso de Hibernate dentro de este servidor de aplicaciones, fue creado a finales de 2001 por Gavin King como alternativa de EJB, siendo hoy Gavin y Christian Bauer los principales gestores de su desarrollo.

Esta herramienta parte de una filosofía de mapear objetos Java “normales”, también conocidos en la comunidad como “POJOs” (Plain Old Java Objects), no contempla la posibilidad de automatizar directamente la persistencia de Entity Beans tipo BMP (es decir, generar automáticamente este tipo de objetos), aunque aun así es posible combinar Hibernate con este tipo de beans utilizando los conocidos patrones para la delegación de persistencia en POJOs.

Una característica de la filosofía de diseño de Hibernate ha de ser destacada dada su gran importancia: puede utilizar los objetos Java definidos por el usuario tal cual, es decir, no utiliza técnicas como generación de códigos a partir de descriptores del modelo de datos o manipulación de bytecodes en tiempo de compilación, ni obliga a implementar interfaces determinadas o a heredar de una superclase. Utiliza en vez de ello el mecanismo de reflexión de Java, característica que le permite un modelado iterativo fluido y natural basado en UML, un factor fundamental para lograr un trabajo ágil y productivo.

Hibernate nos proporciona además un lenguaje para el manejo de consultas a la base de datos. Este lenguaje es similar al SQL y es utilizado para obtener objetos de la base de datos según las condiciones especificadas en el HQL. El uso de HQL permite usar un lenguaje intermedio y según la base de datos

que usemos y el dialecto que especifiquemos será traducido al SQL dependiente de cada base de datos de forma automática y transparente, con lo cual Hibernate gana otro punto en la batalla entre los frameworks de persistencia, la portabilidad e independencia del motor de bases de datos con el que se trabaje.

Características del framework Hibernate:

- No intrusivo (estilo POJO).
- Muy buena documentación (fórums para ayuda, libros).
- Comunidad activa con muchos usuarios.
- Transacciones, caché, asociaciones, polimorfismo, herencia, carga perezosa, persistencia transitiva, estrategias de fetching.
- Potente lenguaje de consulta (HQL): subconsultas, outer joins, ordenamiento, proyeccion (report query, o consultas de reportes), paginación.
- Facilita las pruebas.
- No es estándar.

El API de Hibernate presenta una arquitectura de dos capas (Capa de persistencia y Capa de Negocio).

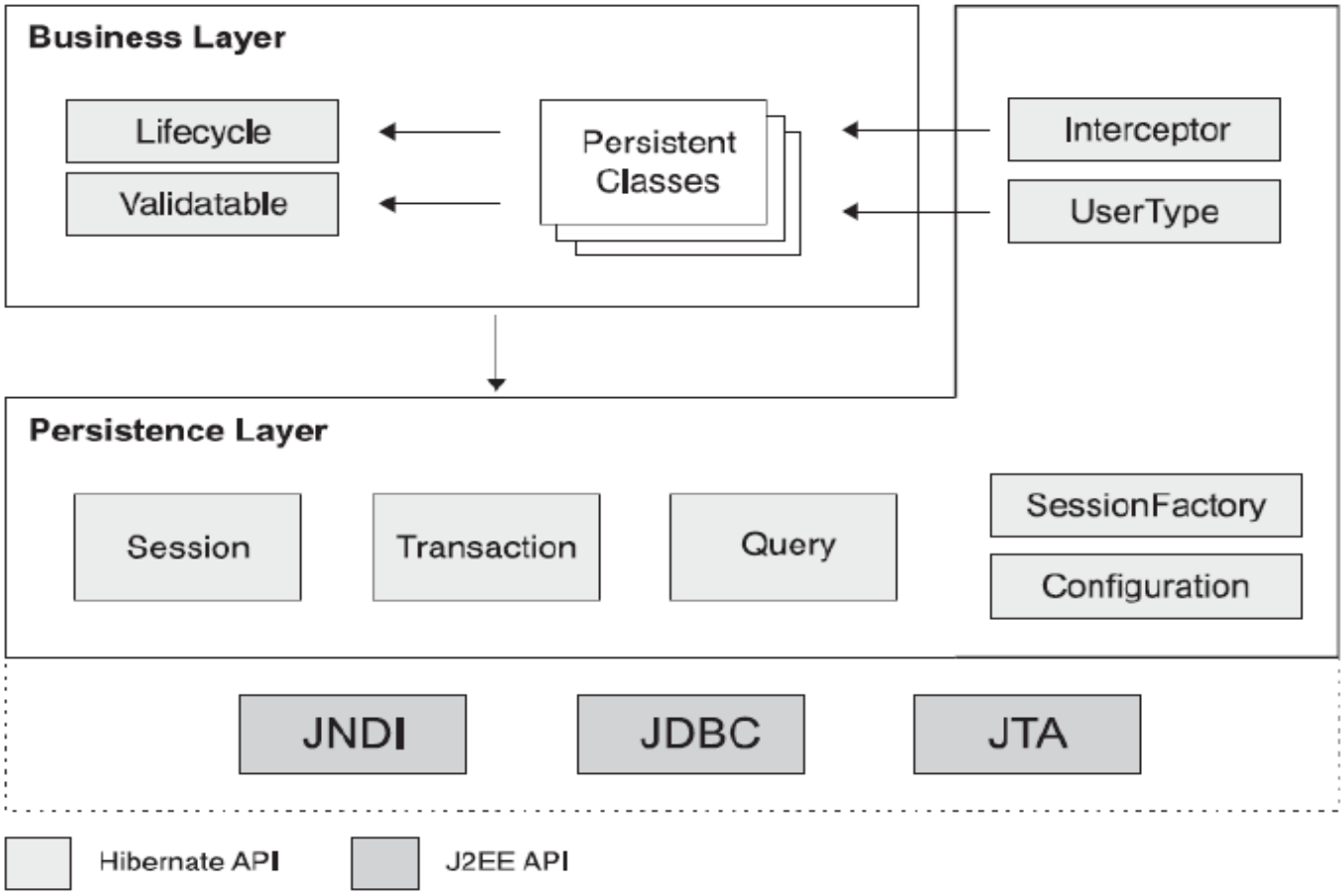



**Fig. 2 Representación de las capas del API.**

Hibernate soporta además la mayoría de los sistemas de bases de datos SQL, ofrece facilidades para recuperación y actualización de datos, control de transacciones, repositorios de conexiones a bases de datos, consultas programáticas y declarativas, y un control de relaciones de entidades declarativas.

Existen diversas herramientas útiles para el uso de Hibernate que de usarse cubren todo el desarrollo de la aplicación hasta la base de datos y viceversa.

Desde herramientas para modelado UML como Poseidón, Rational Rose y Visual Paradigm se pueden generar Modelos Entidad-Relación que son traducidos por AndroMDA a POJO's y mediante XDoclet se generan los ficheros HBM. Todas estas tareas también pueden ser automatizadas mediante el uso de ANT, además el paquete de HibernateTools proporcionado también por el equipo de desarrollo de

Hibernate ofrece estas funcionalidades y otras, todas enmarcadas en el ambiente de desarrollo como un módulo agregado. La comunidad de desarrollo de este framework realiza constantes mejoras al mismo, y cuenta con un muy bien ganado prestigio por la documentación y soporte de innegable calidad que ofrece. La última versión hasta el momento de Hibernate es la 3.3.1 liberada el 11 de septiembre de 2008.

## 1.6. ARQUITECTURA TÉCNICA

En una aplicación web los usuarios acceden a un servidor web a través de Internet o de una intranet mediante un navegador. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores web, en la que se confía la ejecución al navegador, es ligera y no necesita ser instalada.

SIIPOL será un sistema con una aplicación web basada en tecnología Java, con un gestor de base de datos Oracle, un servidor de aplicaciones Apache Tomcat 5.5, y una arquitectura en tres capas.

El sistema está organizado en diferentes módulos, en el módulo de Investigación Forense se gestiona toda la información asociada a las investigaciones forenses, y específicamente, en el Submódulo Cadáveres, el proceso de tratamiento y control de la información de los occisos.

La arquitectura para el Módulo de Investigación Forense quedaría de la siguiente forma:

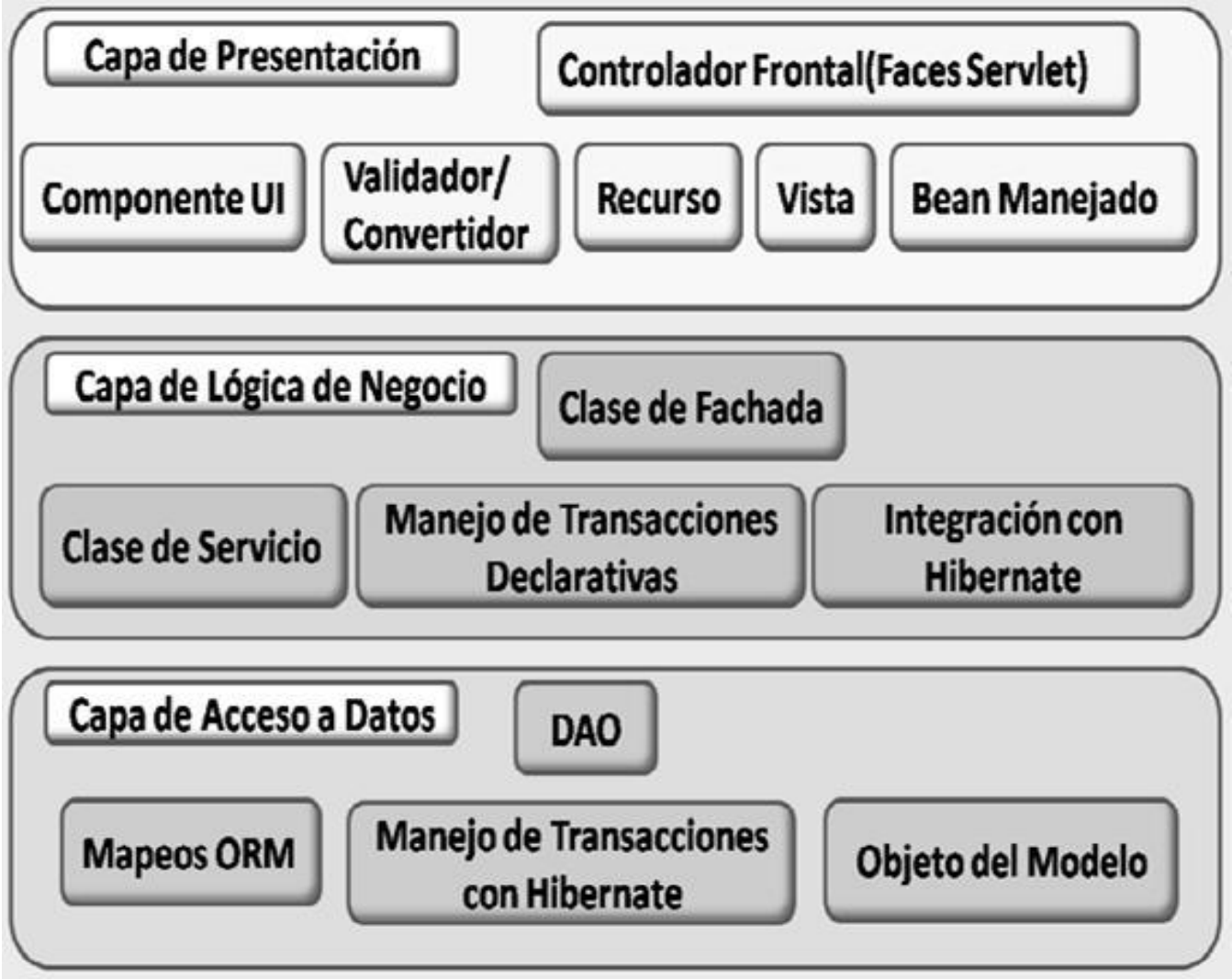



**Fig. 3 Representación de la arquitectura de tres capas.**

En esta arquitectura se pueden distinguir las capas de Presentación, Lógica de Negocio y Acceso a datos. Son en las dos últimas capas para el submódulo Cadáveres donde se va a centrar la investigación que consiste en el Análisis, Diseño e Implementación de las mismas.

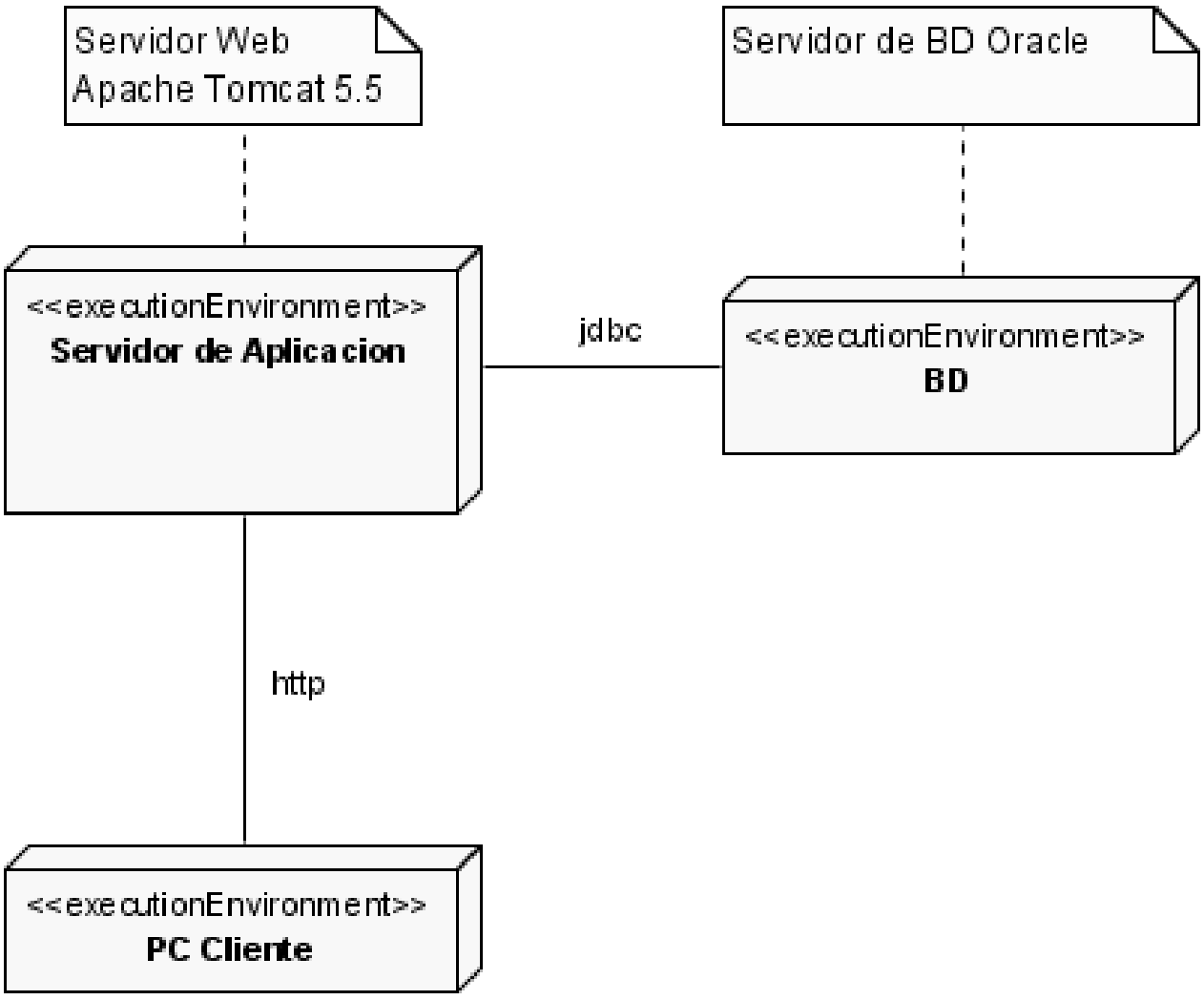



**Fig. 4 Diagrama de despliegue.**

## 1.7. CONCLUSIONES

En este capítulo se investigó si existía alguna solución similar, se estudiaron las encontradas y se dio una breve descripción de las mismas. Se trató la metodología a usar así como la plataforma sobre la que se va a desarrollar la aplicación. También se presentaron las ventajas de las herramientas y frameworks que se van a utilizar en la elaboración del sistema, y por último se presentó la propuesta de cómo quedarán las capas de Lógica de Negocio y Acceso a Datos del submódulo Cadáveres según la arquitectura establecida.

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

## 2.1. INTRODUCCIÓN

En el siguiente capítulo se muestran los modelos de análisis, diseño e implementación de las capas de lógica de negocio y acceso a datos del submódulo cadáveres. Los artefactos que se presentan son los correspondientes a los casos de uso Gestionar Entrevista Antropológica, Gestionar Diagrama Corporal Antropológico y Registrar Levantamiento de Cadáver, ya que en estos se generalizan todas las funcionalidades del submódulo.

## 2.2. MODELO DE ANÁLISIS

El análisis consiste en obtener una visión que se preocupa de ver que hace el sistema de software a desarrollar, por tal motivo este se interesa en los requerimientos funcionales. Constituye una abstracción del Modelo de Diseño. Sus principales componentes son las clases del análisis que tienen tres estereotipos, denominados Interfaz, Controladora y Entidad. En dependencia de la función de una clase identificada en el dominio del problema, esta adoptará uno de los tres estereotipos. El resultado de este flujo es menos específico que el de Diseño, pero ayuda a afianzar los conocimientos de algunos conceptos y relaciones entre las entidades del sistema a implementar.

### 2.2.1. DAGRAMAS DE CLASES DE ANÁLISIS.

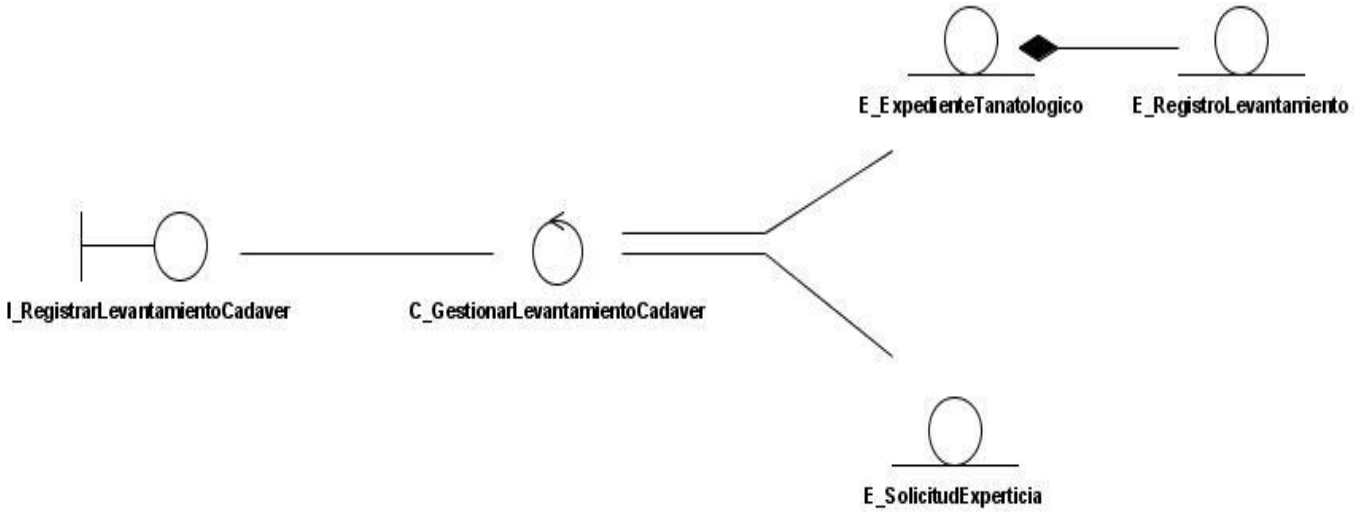



**Fig. 5 Diagrama de Clases de Análisis del CU Registrar Levantamiento de Cadáver.**

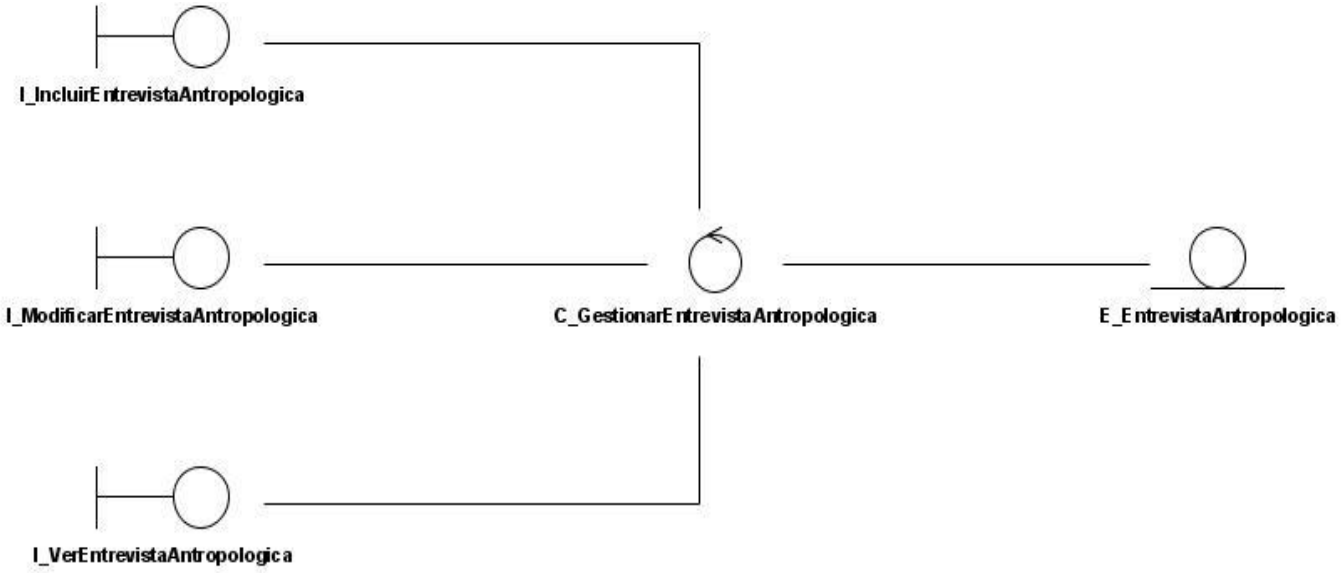



**Fig. 6 Diagrama de Clases de Análisis CU Gestionar Entrevista Antropológica.**

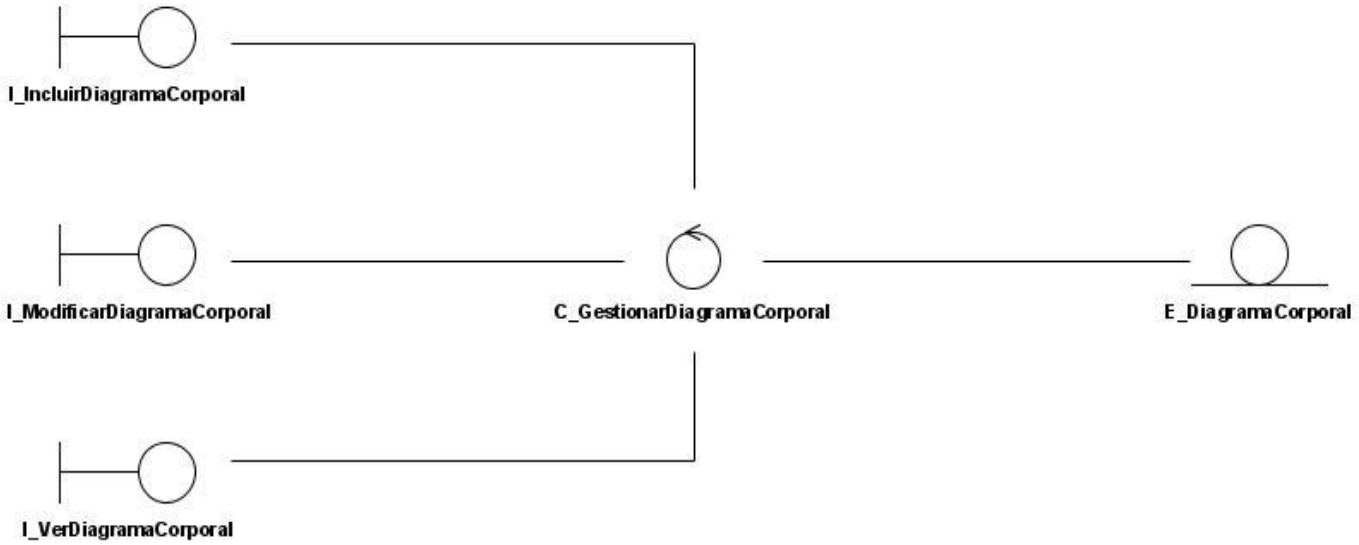



**Fig. 7 Diagrama de Clases de Análisis CU Gestionar Diagrama Corporal Antropológico.**

### 2.2.2. REALIZACIÓN DE CASOS DE USO SIGNIFICATIVOS.

Para ver los diagramas de colaboración de los casos de uso dirigirse al Anexo 2.

## 2.3. MODELO DE DISEÑO

Es una abstracción del Modelo de Implementación y su código fuente, el cual fundamentalmente se emplea para representar y documentar su diseño. Es usado como entrada esencial en las actividades relacionadas a implementación. Representa a los casos de uso en el dominio de la solución. El Modelo de Diseño puede contener: los diagramas, las clases, paquetes, subsistemas, capsulas, protocolos, interfaces, relaciones, colaboraciones, atributos, las realizaciones de los casos de uso, entre otros que se puedan considerar para el sistema en desarrollo.

### 2.3.1. DIAGRAMA DE PAQUETES

Un diagrama de paquetes muestra como un sistema está dividido en agrupaciones lógicas, mostrando las dependencias entre ellas. 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 ellos. Con estas líneas maestras sobre la mesa, los paquetes son buenos elementos de gestión. Cada paquete puede asignarse a un individuo o a un equipo, y las dependencias entre ellos pueden indicar el orden de desarrollo requerido.

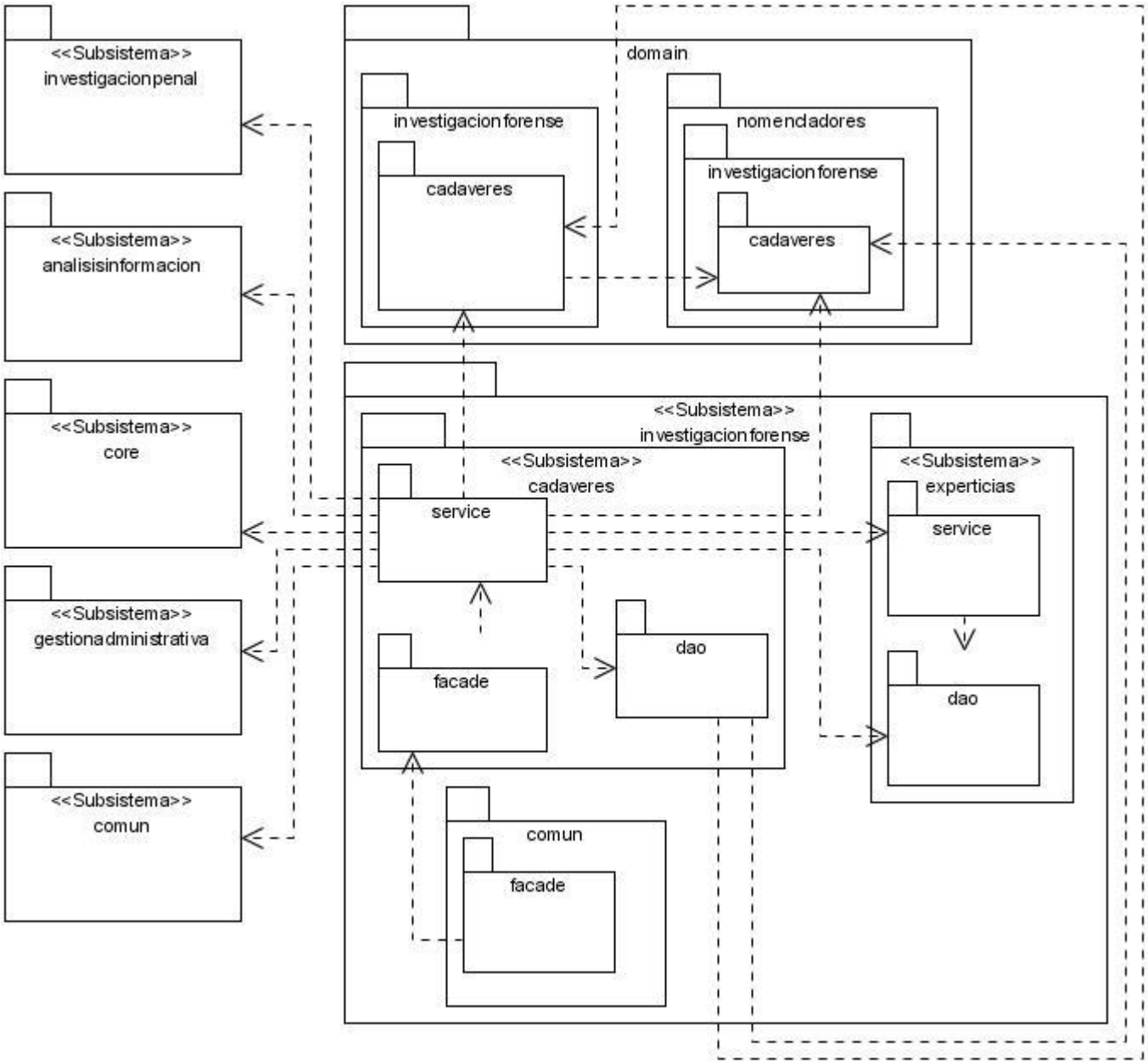



**Fig. 8 Diagrama de Paquetes.**

### 2.3.2. DIAGRAMA DE CLASES DEL DISEÑO

A continuación se visualiza el diagrama de clases de diseño correspondiente a los casos de uso que se eligieron al comienzo del capítulo, con él se dará una vista de cómo interactúan entre ellas. Para ver este diagrama dirigirse al Anexo 3.

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.

Para la capa de lógica de negocio se utilizó el patrón **Fachada**, el mismo provee una interfaz única (*CadaveresFacade*) a un conjunto de interfaces, estas brindan diversas funcionalidades ya que representan las clases de servicio. Otro patrón que se tuvo en cuenta fue el patrón de asignación de responsabilidad **Bajo Acoplamiento,** el cual se encarga de asignar las responsabilidades de modo que se mantenga un bajo acoplamiento, o sea el modo de dar soporte a poca dependencia y a una mayor reutilización.

En la capa de acceso a datos se utilizó el patrón **DAO** (**D**ata **A**ccess **O**bject u **O**bjeto de **A**cceso a **D**atos) el cual abstrae y encapsula todos los accesos a la fuente de datos, oculta completamente los detalles de implementación de la fuente de datos a sus clientes, la interfaz expuesta por DAO no cambia cuando cambia la implementación de la fuente de datos subyacente (diferentes esquemas de almacenamiento). (6)

Para comprender mejor el funcionamiento del submódulo cadáveres se seleccionaron las clases más significativas de las capas de lógica de negocio y acceso a datos para mostrar sus características.

| | |
|---|---|
| Nombre: CadaveresFacade | |
| Tipo de clase: Interfaz | |
| Descripción: Fachada para el submódulo Cadaveres. | |
| Para cada responsabilidad: | |
| Nombre: | obtenerExpedienteTanatologico(Integer idExpTanatologico) |
| Descripción: | Este método devuelve un ExpedienteTanatologico. |
| Nombre: | salvarExpTanatologico(ExpedienteTanatológico ExpedienteTanatologico) |
| Descripción: | Este método salva el ExpedienteTanatológico que se le pasa por parámetros |
| Nombre: | actualizarExpedienteTanatologico(ExpedienteTanatologico ExpedienteTanatológico) |
| Descripción: | Este método actualiza un ExpedienteTanatológico. |
| Nombre: | actualizarRegistroLevantamiento(RegistroLevantamientoCadaveres registro) |
| Descripción: | Este método actualiza un RegistroLevantamientoCadaveres. |
| Nombre: | salvarEntrevistaAntropológica(EntrevistaAntropológicaFamiliar entrevista, boolean flag) |
| Descripción: | Este método salva una EntrevistaAntropológicaFamiliar. |

| | |
|---|---|
| Nombre: | actualizarEntrevistaAntropologica(EntrevistaAntropologicaFamiliar entrevista, boolean flag) |
| Descripción: | Este método actualiza una EntrevistaAntropológicaFamiliar. |
| Nombre: | obtenerEntrevistaAntropologicaFamiliarPorId(Integer id) |
| Descripción: | Este método obtiene una EntrevistaAntropológicaFamiliar dado su id. |
| Nombre: | salvarDiagramaCorporal(DiagramaCorporal diagrama) |
| Descripción: | Este método es el encargado de salvar un DiagramaCorporal. |
| Nombre: | actualizarDiagramaCorporal(DiagramaCorporal diagrama) |
| Descripción: | Este método es el encargado de actualizar un DiagramaCorporal. |
| Nombre: | obtenerDiagramaCorporalPorId(Integer id) |
| Descripción: | Este método obtiene un DiagramaCorporal. |

| | |
|---|---|
| Nombre: EntrevistaAntropológicaFamiliarService | |
| Tipo de clase: Interfaz | |
| Descripción: Interfaz para los servicios de la EntrevistaAntropológicaFamiliar | |
| Para cada responsabilidad: | |
| Nombre: | salvarEntrevistaAntropológica(EntrevistaAntropológicaFamiliar entrevista, boolean flag) |
| Descripción: | Este método es el encargado de salvar una EntrevistaAntropológicaFamiliar. |
| Nombre: | actualizarEntrevistaAntropologica(EntrevistaAntropologicaFamiliar entrevista, boolean flag) |
| Descripción: | Este método es el encargado de actualizar una EntrevistaAntropológicaFamiliar. |
| Nombre: | obtenerEntrevistaAntropologicaFamiliarPorId(Integer id) |
| Descripción: | Este método obtiene una EntrevistaAntropológicaFamiliar. |

| | |
|---|---|
| Nombre: EntrevistaAntropológicaFamiliarServiceImpl | |
| Tipo de clase: Clase de Implementación | |
| Descripción: Implementa los métodos de la Interfaz de servicio de EntrevistaAntropológicaFamiliar | |
| Atributo | Tipo |
| carga | Carga |
| entrevistaAntropologicaFamiliarDao | EntrevistaAntropologicaFamiliarDao |
| nomencladorComunFacade | NomencladorComunFacade |
| informeComunService | InformeComunService |
| nomencladorCadaveresService | NomencladorCadaveresService |
| diagramaCorporalService | DiagramaCorporalService |
| Para cada responsabilidad: | |
| Nombre: | salvarEntrevistaAntropológica(EntrevistaAntropológicaFamiliar entrevista) |
| Descripción: | Este método es el encargado de salvar una EntrevistaAntropológicaFamiliar. |
| Nombre: | actualizarEntrevistaAntropologica(EntrevistaAntropologicaFamiliar entrevista) |

| | |
|---|---|
| Descripción: | Este método es el encargado de actualizar una EntrevistaAntropológicaFamiliar. |
| Nombre: | obtenerEntrevistaAntropologicaFamiliarPorId(Integer id) |
| Descripción: | Este método obtiene una EntrevistaAntropológicaFamiliar. |

| | |
|---|---|
| Nombre: DiagramaCorporalDao | |
| Tipo de clase: Interfaz | |
| Descripción: Interfaz para los métodos de acceso a datos del DiagramaCorporal. | |
| Para cada responsabilidad: | |
| Nombre: | salvar(DiagramaCorporal diagrama) |
| Descripción: | Este método es el encargado de salvar un DiagramaCorporal. |
| Nombre: | actualizar(DiagramaCorporal diagrama) |
| Descripción: | Este método es el encargado de actualizar un DiagramaCorporal. |
| Nombre: | obtenerDiagramaCorporalPorId(Integer id) |
| Descripción: | Este método obtiene un DiagramaCorporal. |

| | |
|---|---|
| Nombre: DiagramaCorporalDaoImpl | |
| Tipo de clase: Clase de Implementación. | |
| Descripción: Implementa los métodos de DiagramaCorporalDao. | |
| Para cada responsabilidad: | |
| Nombre: | salvar(DiagramaCorporal diagrama) |
| Descripción: | Este método es el encargado de salvar un DiagramaCorporal. |
| Nombre: | actualizar(DiagramaCorporal diagrama) |
| Descripción: | Este método es el encargado de actualizar un DiagramaCorporal. |
| Nombre: | obtenerDiagramaCorporalPorId(Integer id) |
| Descripción: | Este método obtiene un DiagramaCorporal. |

| Nombre: RegistroLevantamientoCadaveres | |
|---|---|
| Tipo de clase: Entidad Persistente | |
| Descripción: Almacena los datos de un RegistroLevantamientoCadaveres. | |
| Atributo | Tipo |
| noRegistroLevantamiento | String |
| autoridad | NAutoridad |
| forense | Funcionario |
| entidad | Entidad |
| enfriamiento | String |
| livideces | String |
| rigidez | String |
| posicion | String |
| lugarSuceso | String |
| fechaSuceso | Date |

| | |
|---|---|
| fechaMuerte | Date |
| lugarLevantamiento | String |
| fechaLevantamiento | Date |
| diasMuerte | Integer |
| horasMuerte | Integer |
| tipoMuerte | NTipoMuerte |
| autopsiaSolicitada | Boolean |
| causasMuerteCertificadas | String |
| dentadura | String |
| procedencia | String |
| dependenciaInternaAntropologia | Entidad |
| dependenciaInternaOdontologia | Entidad |
| ingresoVivo | Boolean |
| noHistoriaClinica | String |

| | |
|---|---|
| resumenHistClinica | String |
| fechaIngreso | Date |
| muerteViolentaPresuntiva | NMuerteViolentaPresuntiva |
| comisionDelHecho | NComisionDelHecho |
| otro | String |
| tipoHeridaProyectil | NTipoHeridaProyectil |
| numeroDisparos | Integer |
| tatuaje | NTatuaje |
| orificioSalida | Boolean |
| quedaronProyCad | Boolean |
| observaciones | String |
| numeroHeridas | Integer |
| traumatismosOHeridas | Set<TraumatismoHeridas> |
| tipoHechoTransito | NTipoHechoTransito |

| | |
|---|---|
| tipoVehiculo | NTipoVehiculoForense |
| tipoAccidente | NTipoAccidente |
| tipoAsfixia | NTipoAsfixiaMecanica |
| tipoQuemaduras | Set<NTipoQuemaduras> |
| superficieCorporal | Integer |
| observacionesFinales | String |
| fotos | Set<Foto> |
| solExamenTox | Boolean |
| toxicoInvestigar | String |
| clinicaHospital | String |
| medicoTratante | String |
| indocumentado | Boolean |
| letraCedula | String |
| noCedula | String |

| | |
|---|---|
| pnombreCadaver | String |
| snombreCadaver | String |
| papellidoCadaver | String |
| sapellidoCadaver | String |
| pasaporte | String |

| Nombre: RegistroLevantamientoCadaveres | |
|---|---|
| Tipo de clase: Entidad Persistente | |
| Descripción: Almacena los datos de un RegistroLevantamientoCadaveres. | |
| Atributo | Tipo |
| occiso | Persona |
| registroLevantamiento | RegistroLevantamientoCadaveres |
| registroAutopsia | RegistroAutopsia |
| registroEntrega | RegistroEntrega |

| | |
|---|---|
| registroEntierro | RegistroEntierro |
| solicitudes | Set<SolicitudExperticia> |
| evidencias | Set<Evidencia> |
| estado | NEstadoTanatologico |
| situacionCadaver | NSituacionCadaver |
| funcionarioActaProcesal | Funcionario |
| actaProcesal | ActaProcesal |

### 2.3.3. REALIZACIONES DE CASOS DE USO

Para una mejor comprensión de cómo interactúan los paquetes, pero sin adentrarse en el flujo de eventos que ocurre dentro de los mismos cuando se realiza una petición, se utilizan los diagramas de contrato entre paquetes, los cuales se construyen mediante diagramas de secuencia mostrando el paso de los mensajes entre cada paquete sin llegar a un alto nivel de detalle, esto con el objetivo de ahorra tiempo de diseño.

Como los casos de uso del submódulo Cadáveres tiene un funcionamiento estándar ya que todos cuentan con las funcionalidades: Incluir, Actualizar y un Ver, se decidió presentar solo los diagramas de contrato entre paquetes de los casos de uso más significativos ya que en ellos se generalizan estas funcionalidades. Para ver los diagramas de Contrato entre paquetes dirigirse al Anexo 4.

## 2.4. MODELO DE DATOS

Un modelo de datos es una serie de conceptos que puede utilizarse para describir un conjunto de datos y las operaciones para manipularlos. Hay dos tipos de modelos de datos: los modelos conceptuales y los modelos lógicos. Los modelos conceptuales se utilizan para representar la realidad a un alto nivel de abstracción. Mediante los modelos conceptuales se puede construir una descripción de la realidad fácil de entender. En los modelos lógicos, las descripciones de los datos tienen una correspondencia sencilla con la estructura física de la base de datos.

### 2.4.1. DIAGRAMAS DE CLASES PERSISTENTES

En estos diagramas se muestran las interacciones de las clases persistentes de los casos de uso más significativos del submódulo Cadáveres.

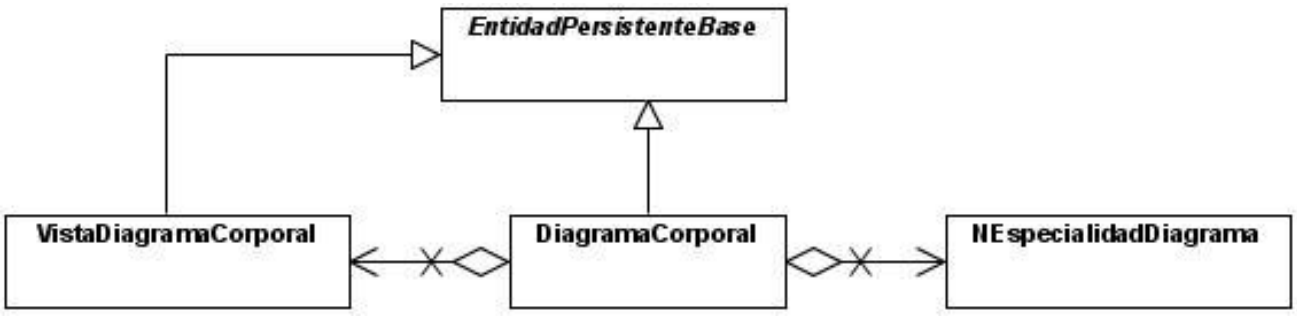



**Fig. 9 Diagrama de Clases Persistentes Diagrama Corporal.**

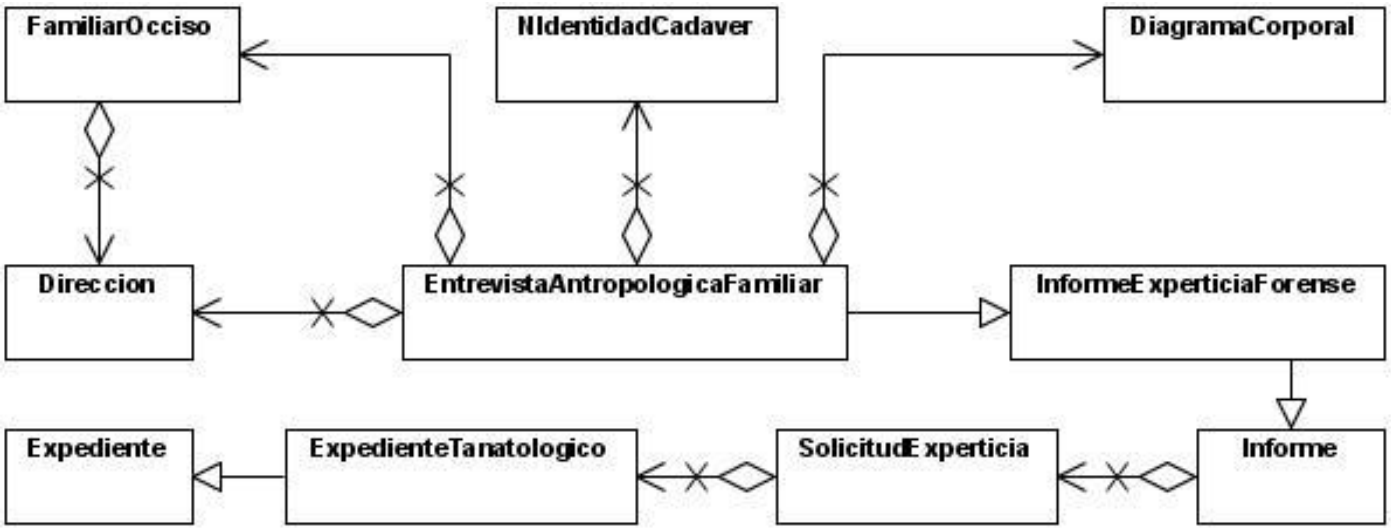



**Fig. 10 Diagrama de Clases Persistentes Entrevista Antropológica a Familiar.**

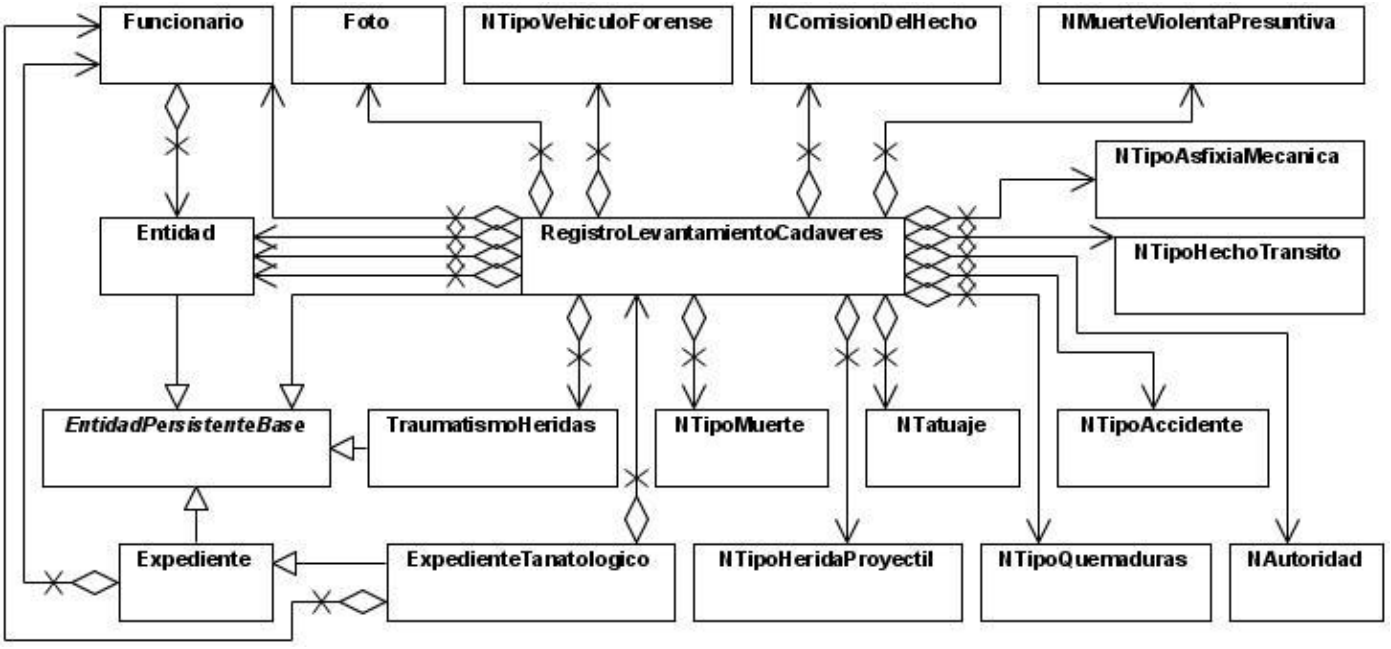



**Fig. 11 Diagrama de Clases Persistentes Registrar Levantamiento de Cadáver.**

### 2.4.2. DIAGRAMA DE TABLAS DEL MODELO RELACIONAL

En este modelo todos los datos son almacenados en relaciones, y como cada relación es un conjunto de datos, el orden en el que estos se almacenen no tiene mayor relevancia (a diferencia de otros modelos como el jerárquico y el de red). Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar por un usuario no experto. La información puede ser recuperada o almacenada por medio de consultas que ofrecen una amplia flexibilidad y poder para administrar la información.

Este modelo considera la base de datos como una colección de relaciones. De manera simple, una relación representa una tabla que no es más que un conjunto de filas, cada fila es un conjunto de campos y cada campo representa un valor que interpretado describe el mundo real. Cada fila también se puede denominar tupla o registro y a cada columna también se le puede llamar campo o atributo.

Para ver la representación del Diagrama de tablas del modelo relacional dirigirse al Anexo 5.

## 2.5. MODELO DE IMPLEMENTACIÓN

El Modelo de Implementación es comprendido por un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema. Entre los componentes podemos encontrar datos, archivos, ejecutables, código fuente y los directorios. Fundamentalmente, se describe la relación que existe desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos. Este artefacto describe cómo se implementan los componentes, congregándolos en subsistemas organizados en capas y jerarquías, señalando las dependencias entre éstos. Para representar los diagramas del Modelo de Implementación se puede emplear el diagrama de UML de Componentes.

### 2.5.1. DIAGRAMA DE SUBSISTEMA DE IMPLEMENTACIÓN

Los subsistemas de implementación incluyen dependencias y otras informaciones. También podrían incluir modelos claves del subsistema como el diagrama de componentes. Además un subsistema puede implementar las interfaces que representan la funcionalidad que exportan en forma de operaciones.

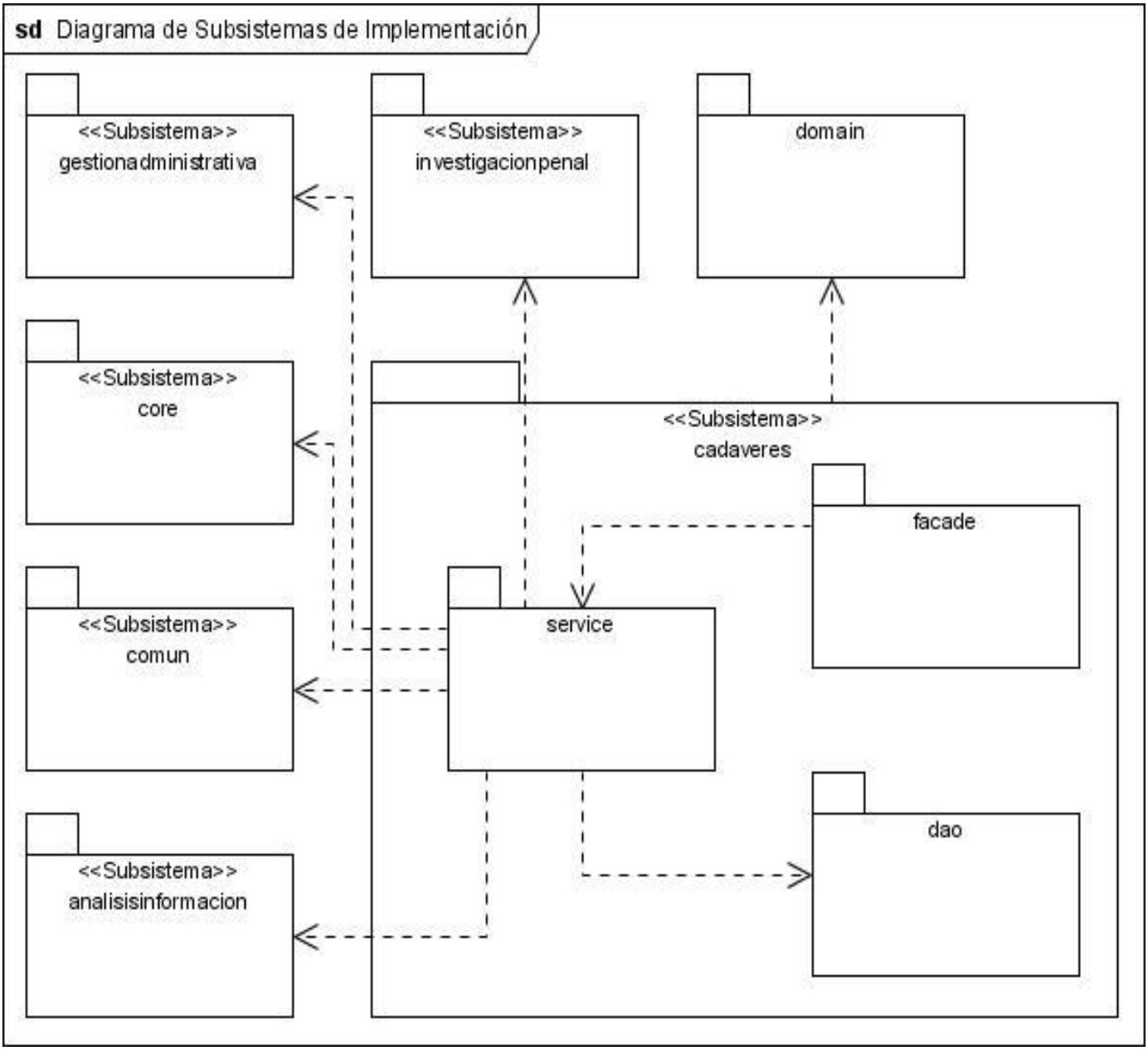



**Fig. 12 Diagrama de Subsistemas de Implementación.**

### 2.5.2. DIAGRAMA DE COMPONENTES

Un diagrama de componentes 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. Los diagramas de Componentes prevalecen en el campo de la arquitectura de software, pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema.

Debido a que estos son más parecidos a los diagramas de casos de usos,  es que son utilizados para modelar la vista estática de un sistema. Muestra la organización y las dependencias entre un conjunto de componentes. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes. Cada diagrama describe un apartado del sistema.

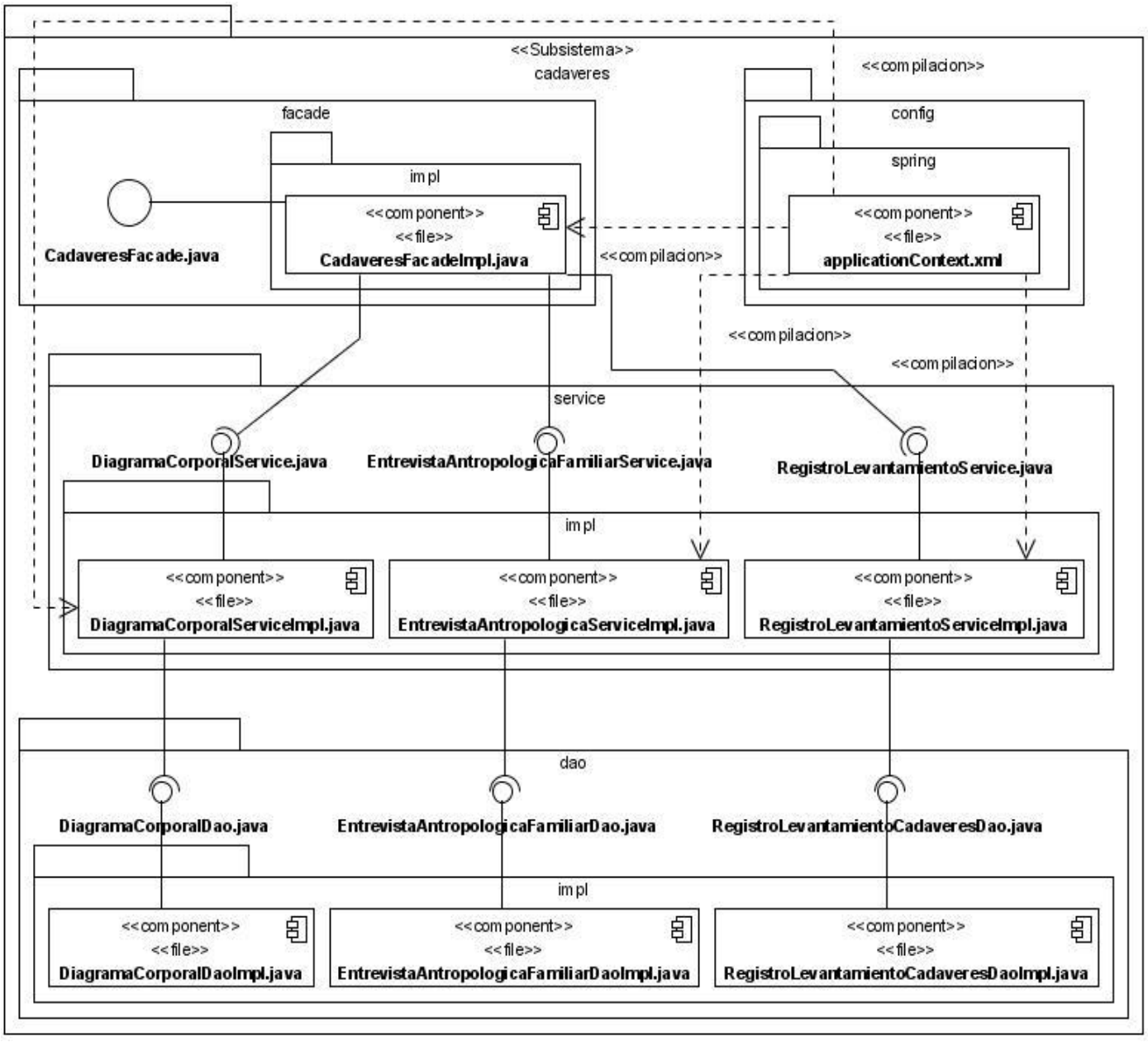



**Fig. 13 Diagrama de Componentes papa el submódulo Cadáveres.**

### 2.5.3. ESTANDAR DE CODIFICACIÓN

Durante la implementación del submódulo Cadáveres se utilizó el estándar de codificación del proyecto CICPC definido por la arquitectura del mismo. Este estándar se basa en el Java Code Conventions (Convenciones de Código Java). Para mayor detalle consultar la Guía de Estilo de Código que se encuentra en la documentación del proyecto CICPC.

## 2.6. CONCLUSIONES

A través de este capítulo se han mostrado los artefactos creados en el análisis, diseño e implementación del submódulo cadáveres. Para ello se han generalizado las funcionalidades en los casos de uso más significativos, para que se tenga una mejor apreciación de cómo funciona el subsistema y cómo interactúan todos sus diversos componentes.

# CAPÍTULO 3. VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

## 3.1. INTRODUCCIÓN

Con el objetivo de demostrar la hipótesis planteada, se realizaron un conjunto de pruebas para verificar el cumplimiento de los requisitos funcionales y no funcionales del submódulo Cadáveres. En el presente capítulo se expondrán los resultados de las pruebas antes mencionadas así como una breve descripción de las mismas.

## 3.2. TIPOS DE PRUEBAS

El objetivo de la fase de pruebas de un programa es el de detectar todo posible malfuncionamiento antes de ser entregado. Este proceso requiere mucho tiempo y esfuerzo, se estima que la mitad del esfuerzo de desarrollo de un programa se invierte en esta fase. Un error detectado puede ser costoso de reparar; pero siempre es peor que el error sea descubierto por el usuario al final. Para comprobar el correcto funcionamiento del submódulo Cadáveres se aplicaron dos tipos de pruebas: Caja Negra y Caja Blanca.

**Pruebas de Caja Negra.**

Las pruebas de caja negra se centran en lo que se espera de un módulo, es decir, intentan encontrar casos en que el módulo no se atiene a su especificación. Por ello se denominan pruebas funcionales, y el probador se limita a suministrarle datos como entrada y estudiar la salida, sin preocuparse de lo que pueda estar haciendo el sistema internamente. Las pruebas de caja negra se apoyan en la especificación de requisitos del módulo.

El problema con las pruebas de caja negra no suele estar en el número de funciones proporcionadas por el módulo (que siempre es un número muy limitado en diseños razonables); sino en los datos de entrada a estas funciones. El conjunto de datos posibles suele ser muy amplio.

A la vista de los requisitos de un módulo, se sigue una técnica algebraica conocida como "clases de equivalencia". Esta técnica trata cada parámetro como un modelo algebraico donde unos datos son equivalentes a otros. Si logramos partir un rango excesivamente amplio de posibles valores reales a un conjunto reducido de clases de equivalencia, entonces es suficiente probar un caso de cada clase, pues

los demás datos de la misma clase son equivalentes. Durante la lectura de los requisitos del sistema, se encuentran una serie de valores singulares, que marcan diferencias de comportamiento. Estos valores son claros candidatos a marcar clases de equivalencia: baja y alta.

Una vez identificadas las clases de equivalencia significativas en el módulo, se procede a seleccionar un valor de cada clase, que no esté justamente al límite de la clase. Este valor aleatorio, hará las veces de cualquier valor normal que se pueda usar en la ejecución real.

La experiencia muestra que un buen número de errores aparecen en torno a los puntos de cambio de clase de equivalencia. Hay una serie de valores denominados "frontera" (o valores límite) que conviene probar. Usualmente se necesitan 2 valores por frontera, uno justo abajo y otro justo encima.

Lograr una buena cobertura con pruebas de caja negra es un objetivo deseable; pero no suficiente a todos los efectos. Un programa puede pasar con holgura millones de pruebas y sin embargo tener defectos internos que surgen en el momento más inoportuno.

Las pruebas de caja negra aseguran que un programa hace lo que se quiere; pero que haga (además) otras cosas menos aceptables. (7)

**Pruebas de Caja Blanca.**

En estas pruebas se está siempre observando el código que las pruebas se dedican a ejecutar con ánimo de "probarlo todo". Esta noción de prueba total se formaliza en lo que se llama "cobertura" y no es sino una medida porcentual de ¿cuánto código se cubrió?

Hay diferentes variantes para definir la cobertura. Todas ellas intentan sobrevivir al hecho de que el número posible de ejecuciones de cualquier programa no trivial es (a todos los efectos prácticos) infinito. Pero si el 100% de cobertura es infinito, ningún conjunto real de pruebas pasaría de un infinitésimo de cobertura.

Existen cuatro tipos diferentes de coberturas:

- Cobertura de segmentos.
- Cobertura de ramas.

- Cobertura de condición/decisión.
- Cobertura de bucles.

En la práctica se procura alcanzar una cobertura cercana al 100% de segmentos. Es muy recomendable (aunque cuesta más) conseguir una buena cobertura de ramas.

Una buena cobertura depende de lo crítico que sea el programa. Hay que valorar el riesgo (o coste) que implica un fallo si éste se descubre durante la aplicación del programa. La cobertura requerida suele ir creciendo con el ámbito previsto de distribución. Si un programa se distribuye y falla en algo grave puede ser necesario redistribuirlo de nuevo y urgentemente.

La ejecución de pruebas de caja blanca puede llevarse a cabo con un depurador (que permite le ejecución paso a paso).

Lograr una buena cobertura con pruebas de caja blanca es un objetivo deseable, pero no suficiente a todos los efectos. Un programa puede estar perfecto en todos sus términos, y sin embargo no servir a la función que se pretende. Las pruebas de caja blanca confirman que un programa funciona correctamente; pero no de que haga lo que necesitamos. (7)

## 3.3. RESULTADOS DE LAS PRUEBAS

Al aplicar los dos tipos de pruebas descritos anteriormente se detectaron no conformidades (NC), estas son todos aquellos requisitos funcionales y no funcionales que no fueron cumplidos al desarrollar el submódulo Cadáveres. Los resultaron quedaron como sigue:

| Pruebas de Aceptación 2 | | | NC | | | |
|---|---|---|---|---|---|---|
| | **CU** | **PC** | **Baja** | **Media** | **Alta** | **Total** |
| **SIIPOL** | 299 | 122 | 12 | 8 | 3 | 23 |
| **Cadáveres** | 25 | 16 | 0 | 0 | 0 | 0 |

| Aceptación 2.02 | | | NC | | | |
|---|---|---|---|---|---|---|
| | **CU** | **PC** | **Baja** | **Media** | **Alta** | **Total** |
| **SIIPOL** | 299 | 23 | 1 | 14 | 12 | 27 |
| **Cadáveres** | 25 | 4 | 0 | 0 | 1 | 1 |

| Pruebas Piloto | | | NC | | | |
|---|---|---|---|---|---|---|
| | **CU** | **PC** | **Baja** | **Media** | **Alta** | **Total** |
| **SIIPOL** | 299 | 116 | 22 | 14 | 42 | 78 |
| **Cadáveres** | 25 | 2 | 0 | 1 | 0 | 1 |

## 3.4. CONCLUSIONES

En este capítulo se describieron los dos tipos de pruebas, pruebas de Caja Blanca y de Caja Negra, a las que fue sometido el submódulo Cadáveres. También se presentaron los resultados de las mismas para el subsistema y para el sistema en general.

# CONCLUSIONES

Esta investigación muestra como se analizó, diseñó e implementó las capas Lógica de Negocio y Acceso a datos del submódulo Cadáveres, para dar solución a la situación por la que atraviesa la Coordinación Nacional de Ciencias Forenses de la hermana república de Venezuela.

Los objetivos propuestos se fueron cumpliendo de acuerdo a la planificación y en el tiempo estimado llevando a cabo las tareas propuestas. Para lograrlo se estudiaron los procesos de la CNCF y la documentación del proyecto CICPC, así como el funcionamiento del módulo Investigación Forense y las herramientas definidas por la arquitectura para desarrollar el sistema. Haciendo uso del conocimiento obtenido y usando una metodología de desarrollo adecuada y buenas prácticas de diseño orientado a objetos se generó un modelo de análisis, un modelo de diseño y un modelo de Implementación con la documentación referente a los mismos.

El subsistema fue sometido a un conjunto de pruebas de calidad mediante las cuales se identificaron y depuraron los errores que presentaba el subsistema, los resultados de las mismas y la conformidad del cliente con el producto final demuestran que se obtuvo una capa de Lógica de Negocio y una capa de Acceso a Datos para el submódulo Cadáveres perteneciente al módulo Investigación Forense del Sistema de Investigación e Información Policial dando por cumplidos los requisitos funcionales y no funcionales definidos para el mismo.

# RECOMENDACIONES

- Realizar una refactorización de la implementación de cada caso de uso para optimizar el rendimiento del mismo.
- Estudiar la posibilidad de automatizar los procesos del instituto de Medicina Legal en Cuba.
- Usar este documento como ayuda para capacitar nuevos desarrolladores para el proyecto CICPC.
- Consultar la documentación del proyecto CICPC para una mejor comprensión de este trabajo.

# BIBLIOGRAFÍA

1. Biometría. [En línea] Ministerio de Justicia, Seguridad y Derechos Humanos, 2008. [Citado el: 2 de Febrero de 2009.] www.biometria.gov.ar.

2. POLICIA DE NEUQUEN. [En línea] Gobierno de la Provincia del Neuquén, 2008. [Citado el: 2 de Marzo de 2009.] www.policiadelneuquen.gov.ar.

3. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* Madrid : Addison Wesley, 2000.

4. Overview. SUN MicroSystems. *Java 2 Platform, Enterprise Edition(J2EE).* [En línea] Enero de 2009. http://java.sun.com/j2ee/overview.html.

5. Why Visual Paradigm for UML? . *UML CASE Tools - Free for Learning UML, Cost-Effective for Business Solutions.* [En línea] Enero de 2009. http://www.visual-paradigm.com/product/vpuml/index.jsp.

6. **Larman, Craig.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* Mexico : Prentice Hall, 1999.

7. **Mañas, José A.** DIT. [En línea] Universidad Politécnica de Madrid, 16 de Marzo de 1994. [Citado el: 4 de Marzo de 2008.] http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/testing.htm.

# GLOSARIO DE TÉRMINOS

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

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

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

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

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

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

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

**Framework**: Es una estructura de soporte definida en la cual otro proyecto de *software* puede ser organizado y desarrollado. Típicamente, un *framework* puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros *software* para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

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

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

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

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

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

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

**JSF**: *framework* de la capa de presentación para Java.

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

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

**NetBeans**: Entorno Integrado de Desarrollo para crear aplicaciones en lenguaje Java. **Servlet**: Es un objeto que se ejecuta en un servidor o contenedor JEE, fue especialmente diseñado para ofrecer contenido dinámico desde un servidor web.

**SIIPOL**: Sistema de Investigación e Información Policial.

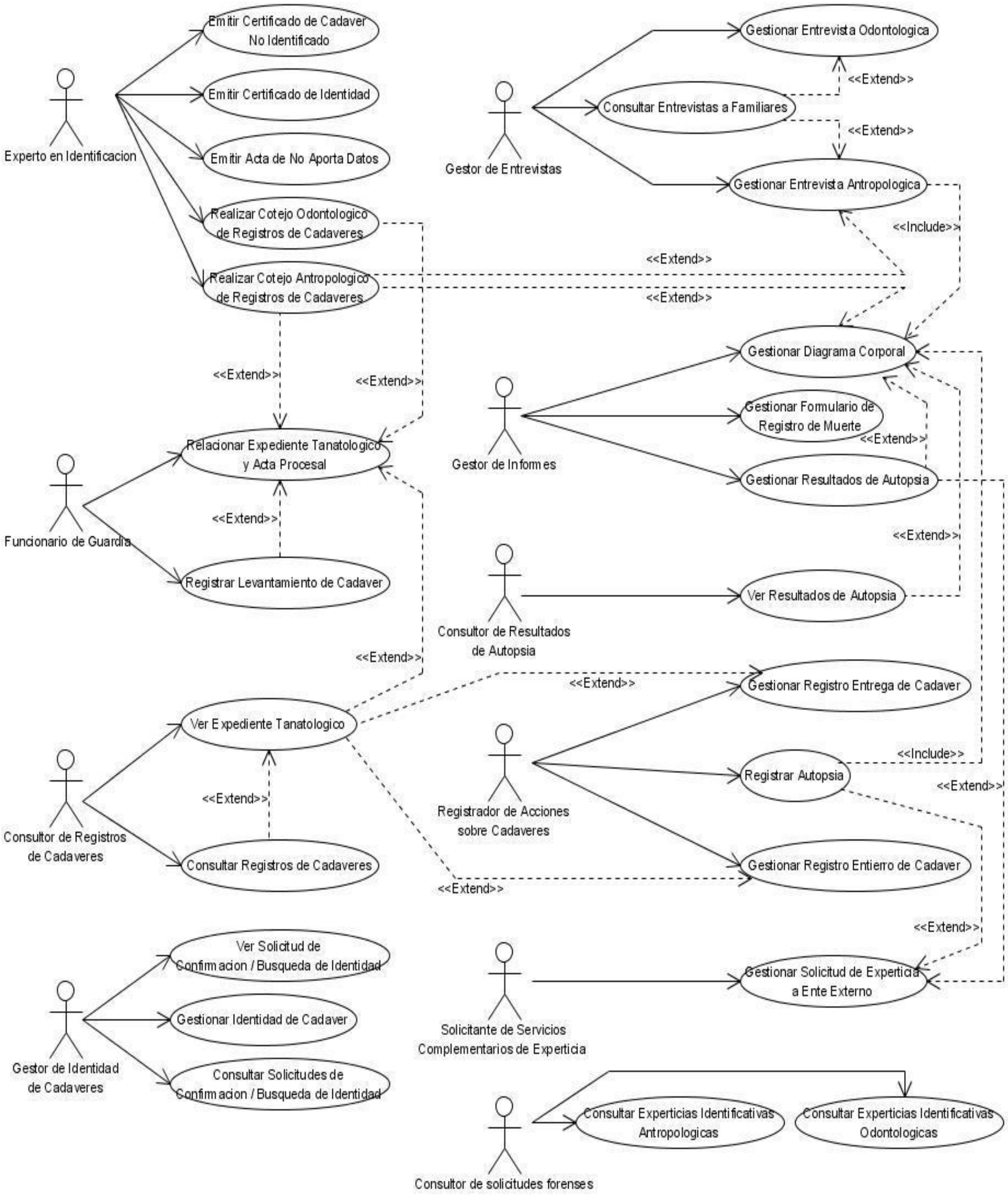
**Spring**: Framework de la capa de lógica de negocio para Java.

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

# ANEXOS

## ANEXO 1

Diagrama de casos de uso del subsistema Cadáveres.



## ANEXO 2

Diagrama de Colaboración CU Registrar Levantamiento de Cadáver.

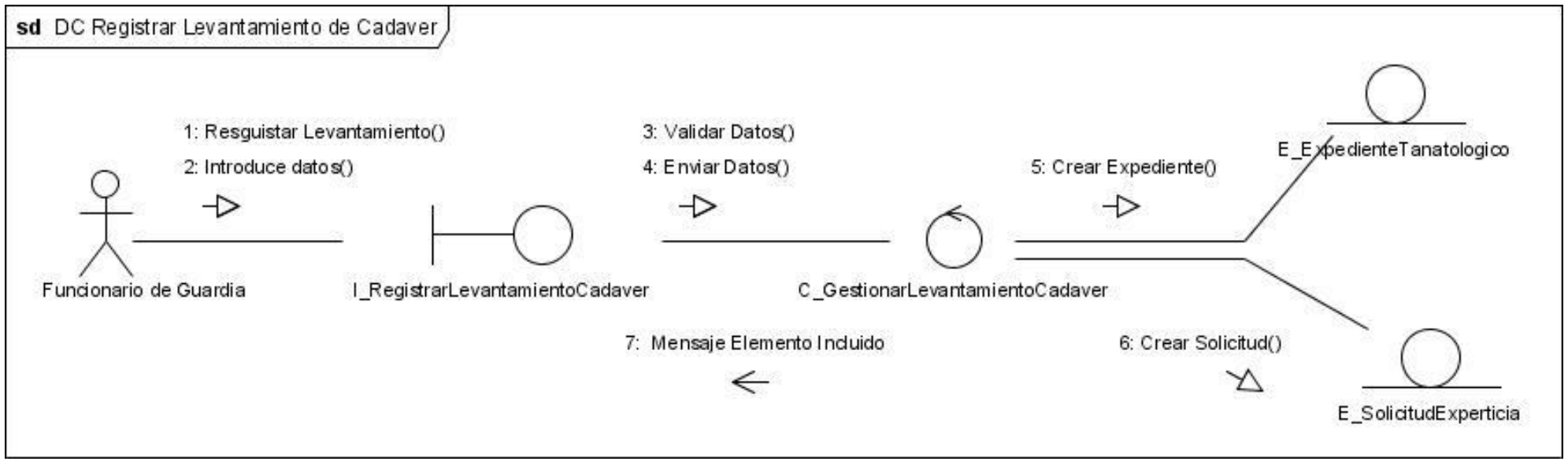



Diagrama de Colaboración CU Gestionar Entrevista Antropológica (Incluir)

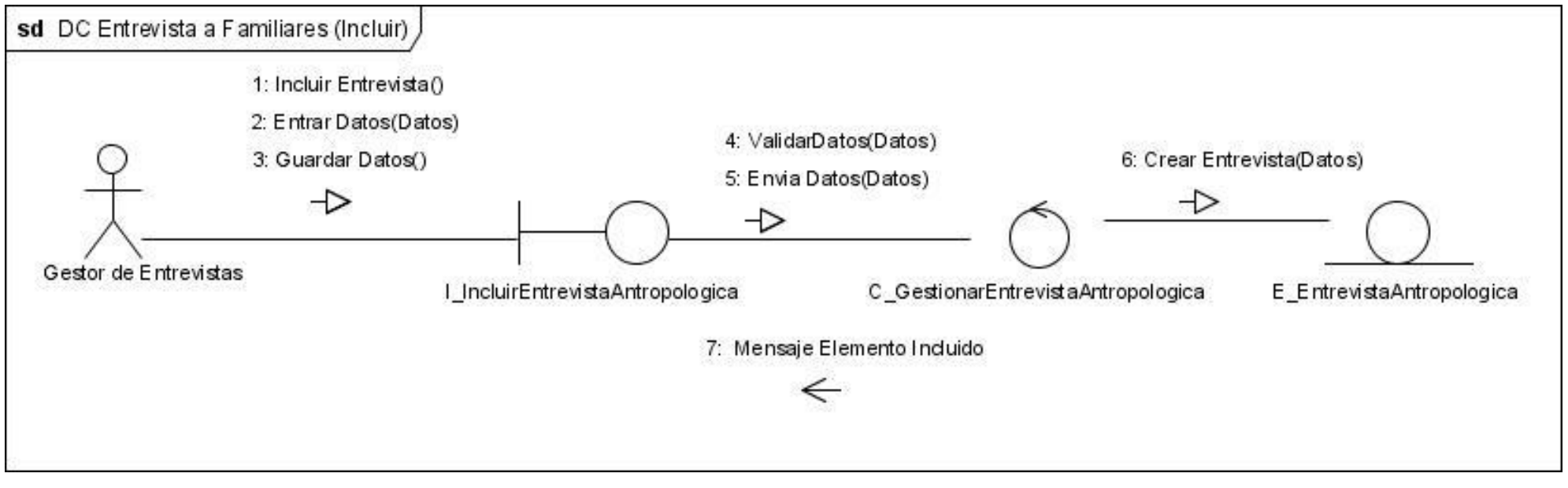



Diagrama de Colaboración CU Gestionar Entrevista Antropológica (Modificar)

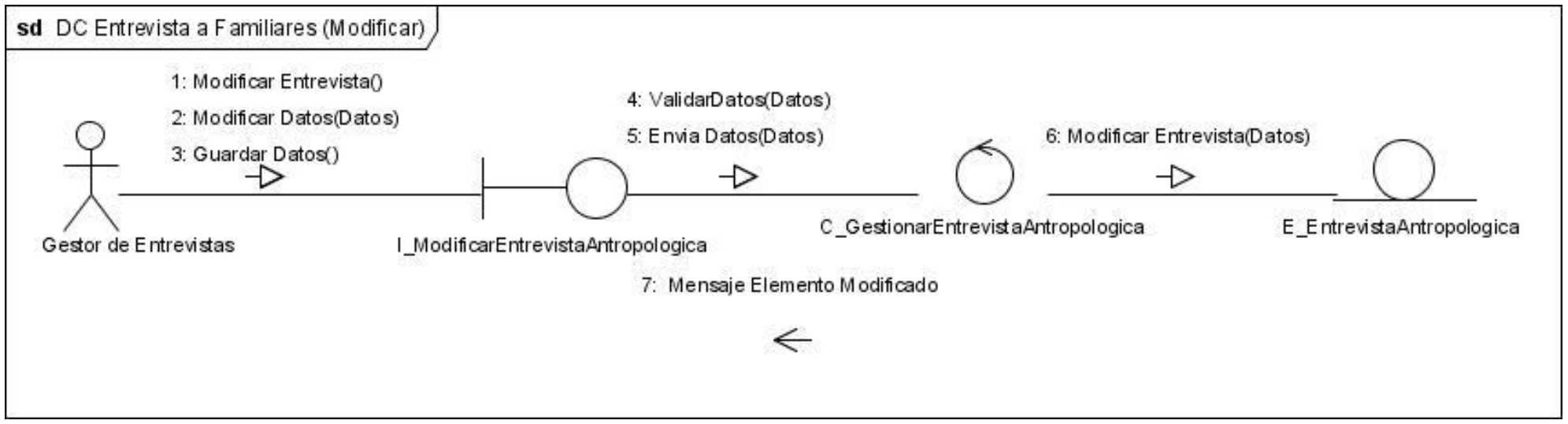
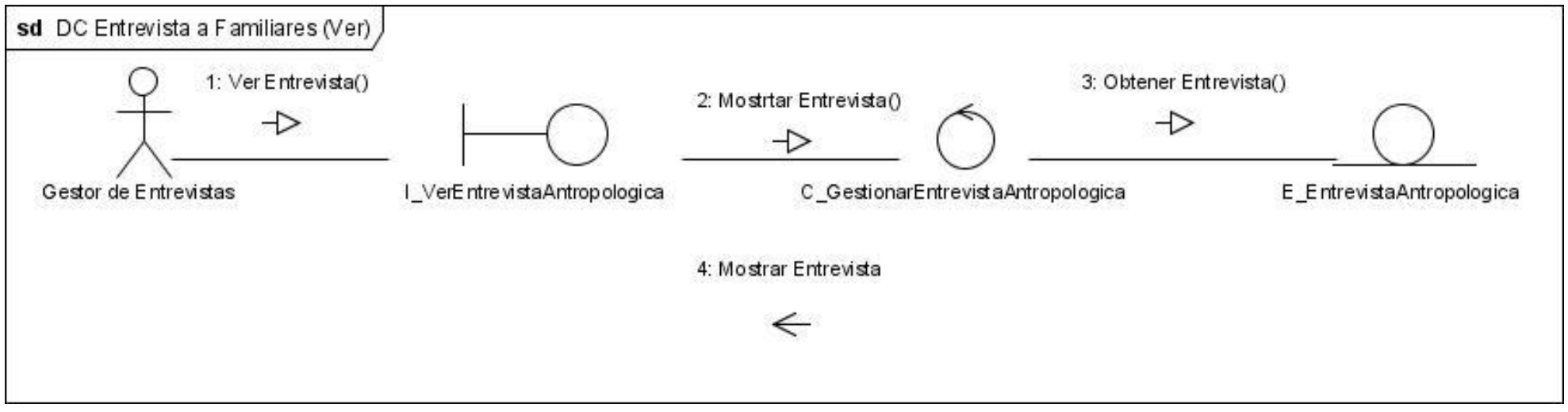
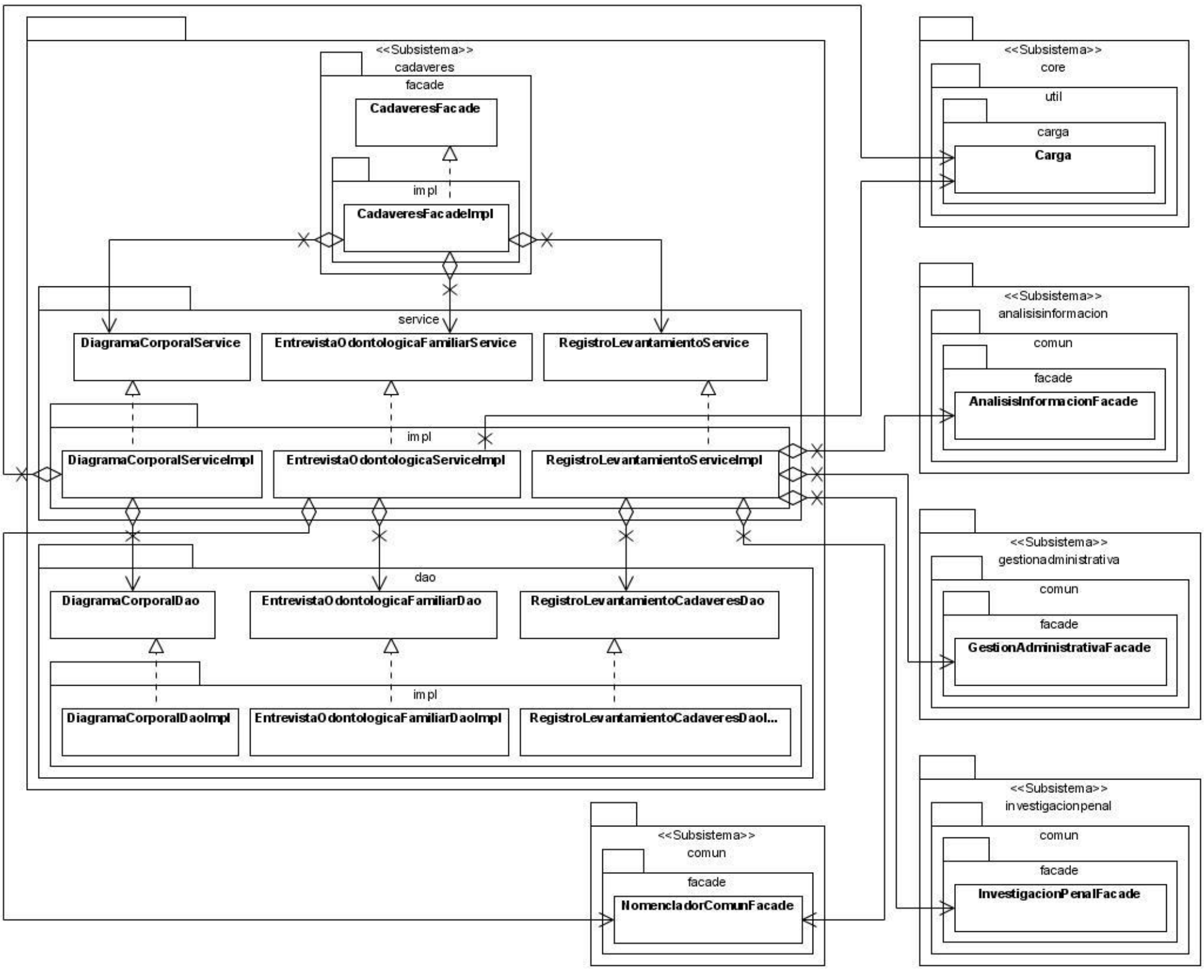


Diagrama de Colaboración CU Gestionar Entrevista Antropológica (Ver)



## ANEXO 3

Diagrama de clases de Diseño



## ANEXO 4

Diagrama de contrato entre paquetes CU Registrar Levantamiento de cadáver

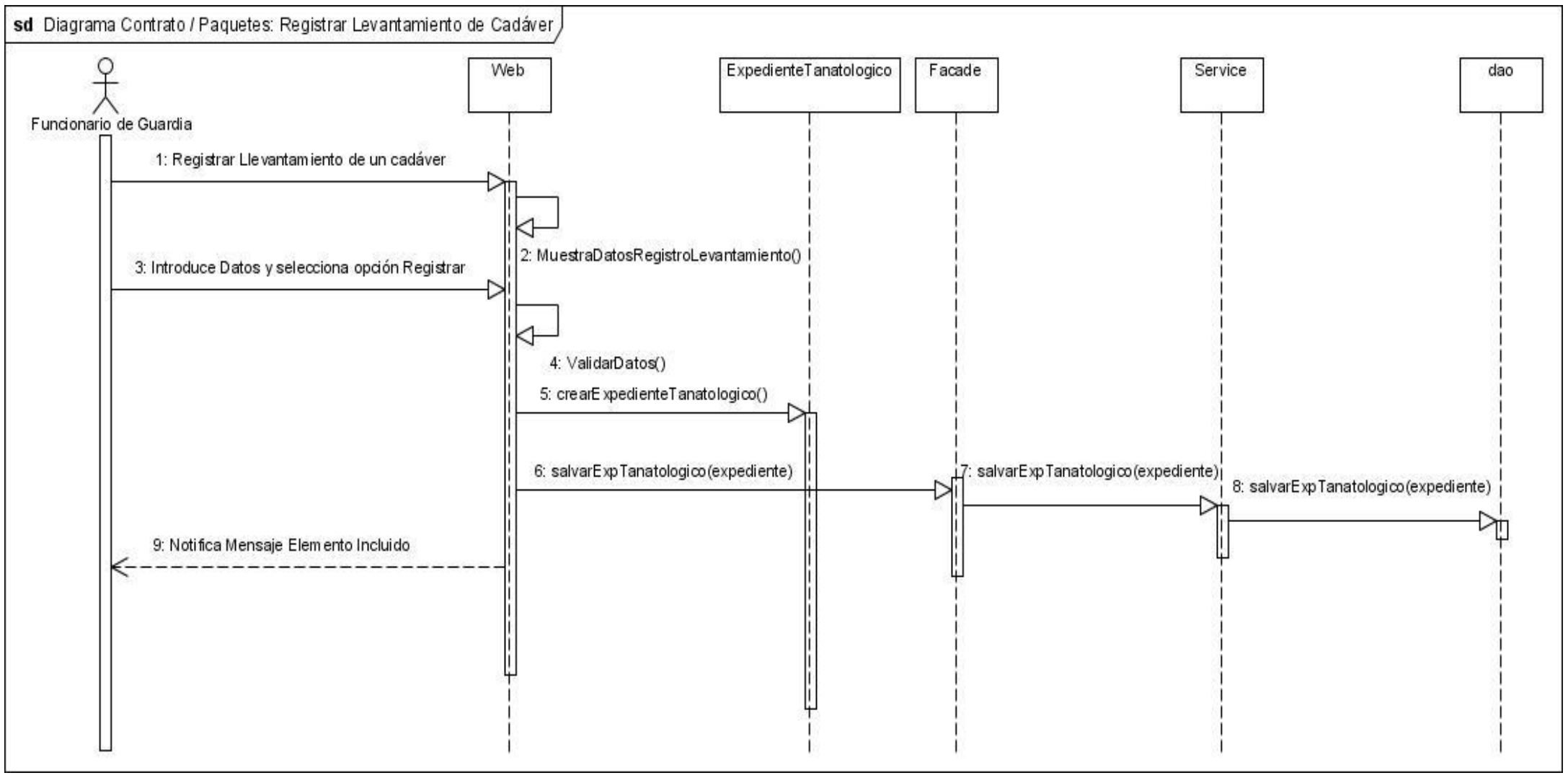



Diagrama de contrato entre paquetes CU Gestionar Diagrama Corporal (Incluir)

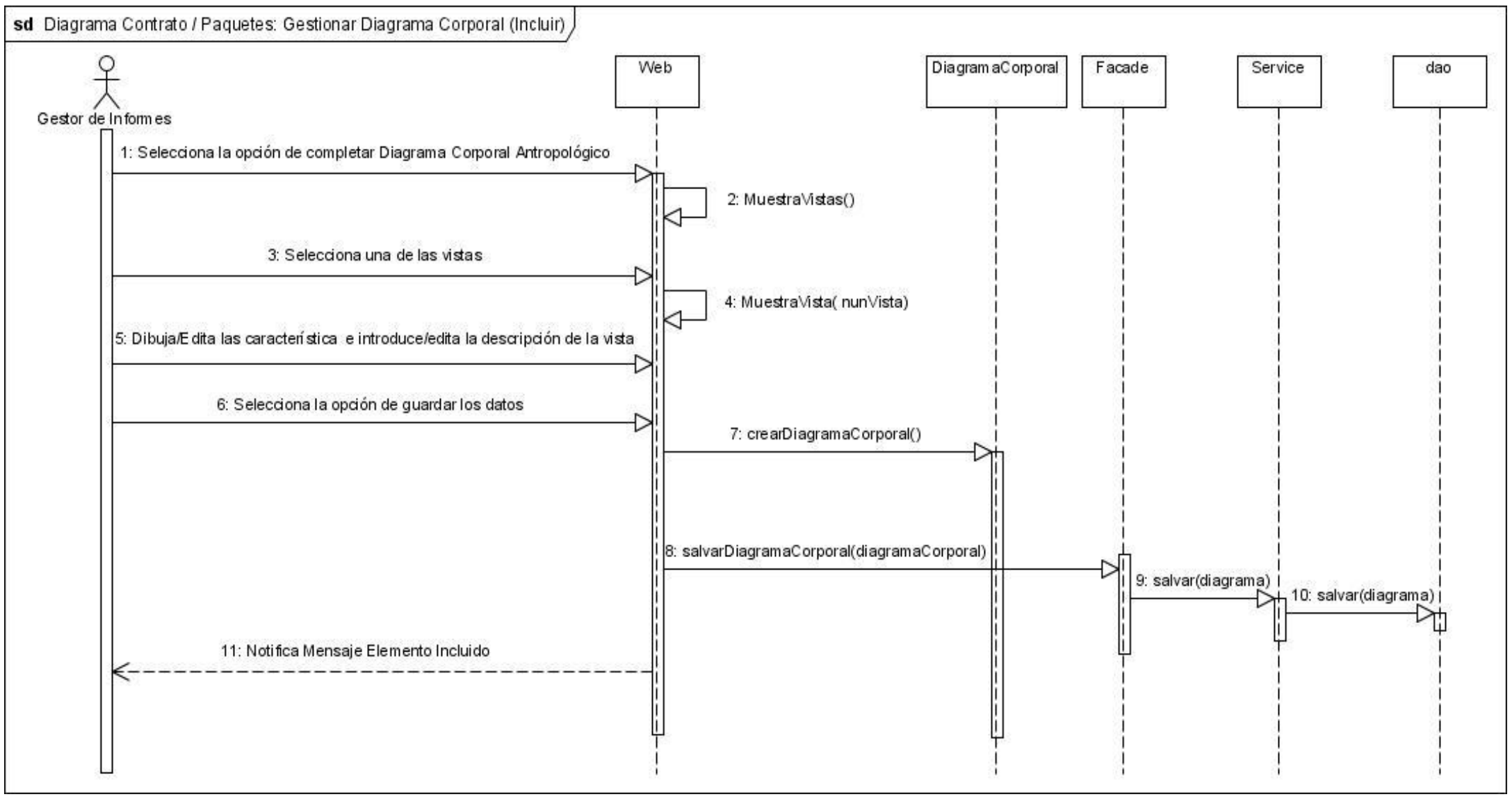


Diagrama de contrato entre paquetes CU Gestionar Diagrama Corporal (Modificar)

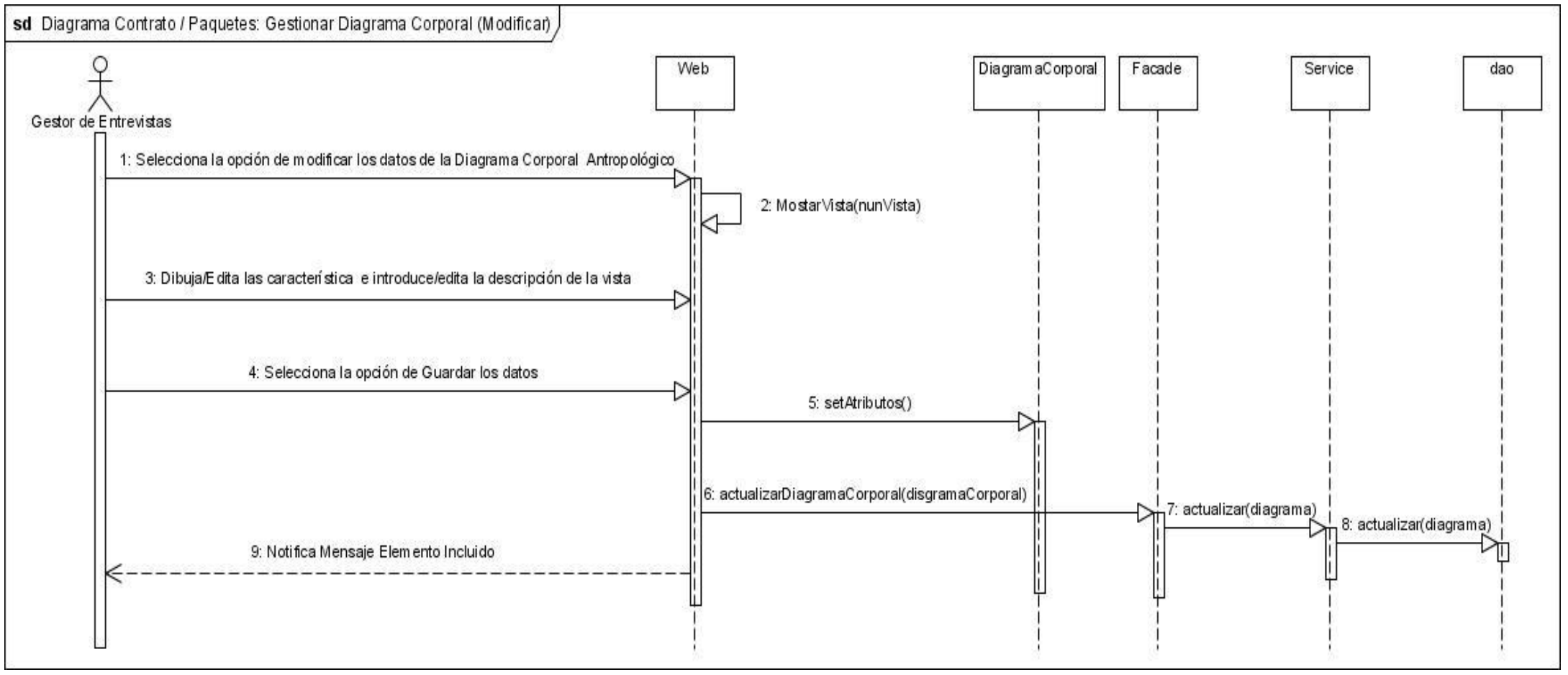



Diagrama de contrato entre paquetes CU Gestionar Diagrama Corporal (Ver)

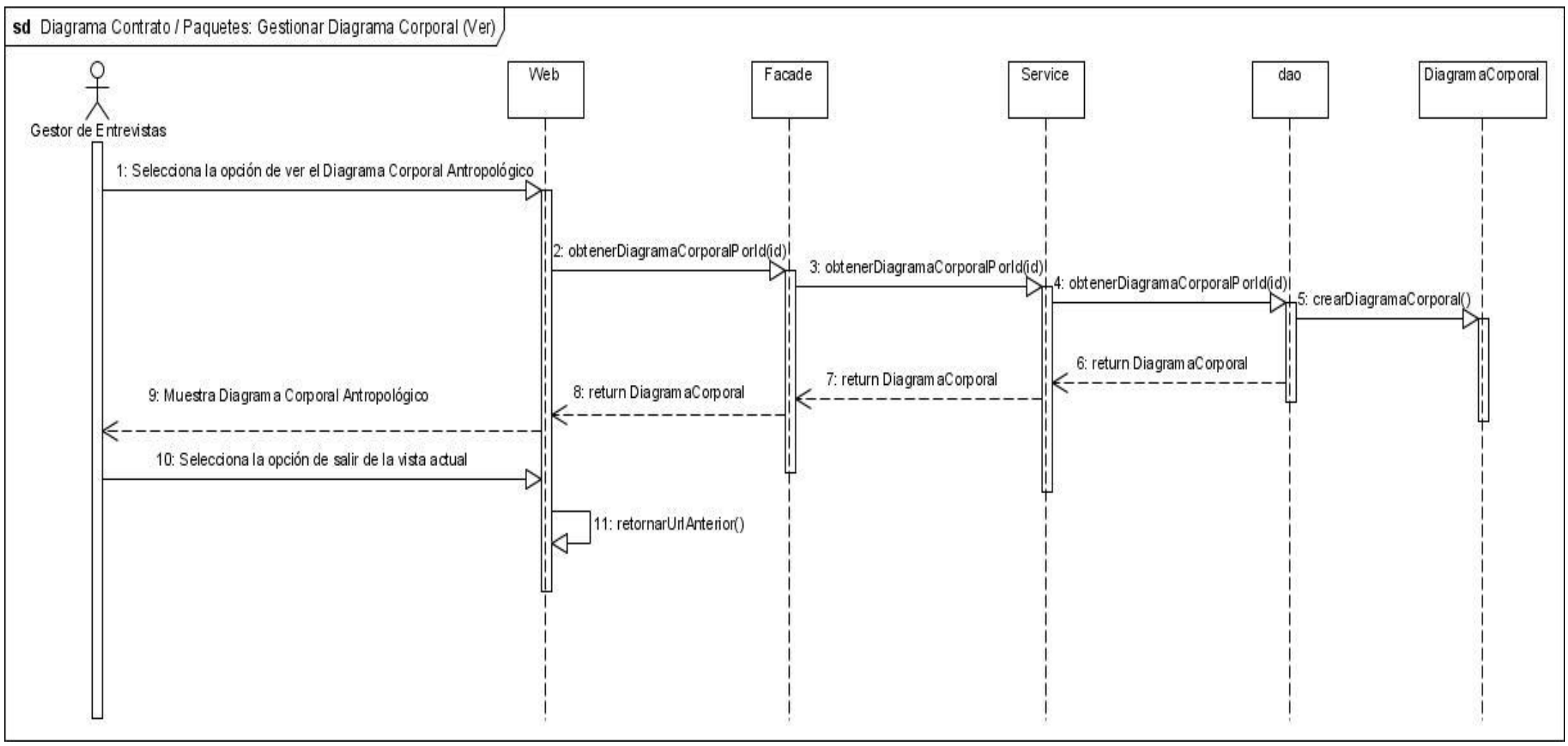


Diagrama de contrato entre paquetes CU Gestionar Entrevista Antropológica a Familiar (Incluir)

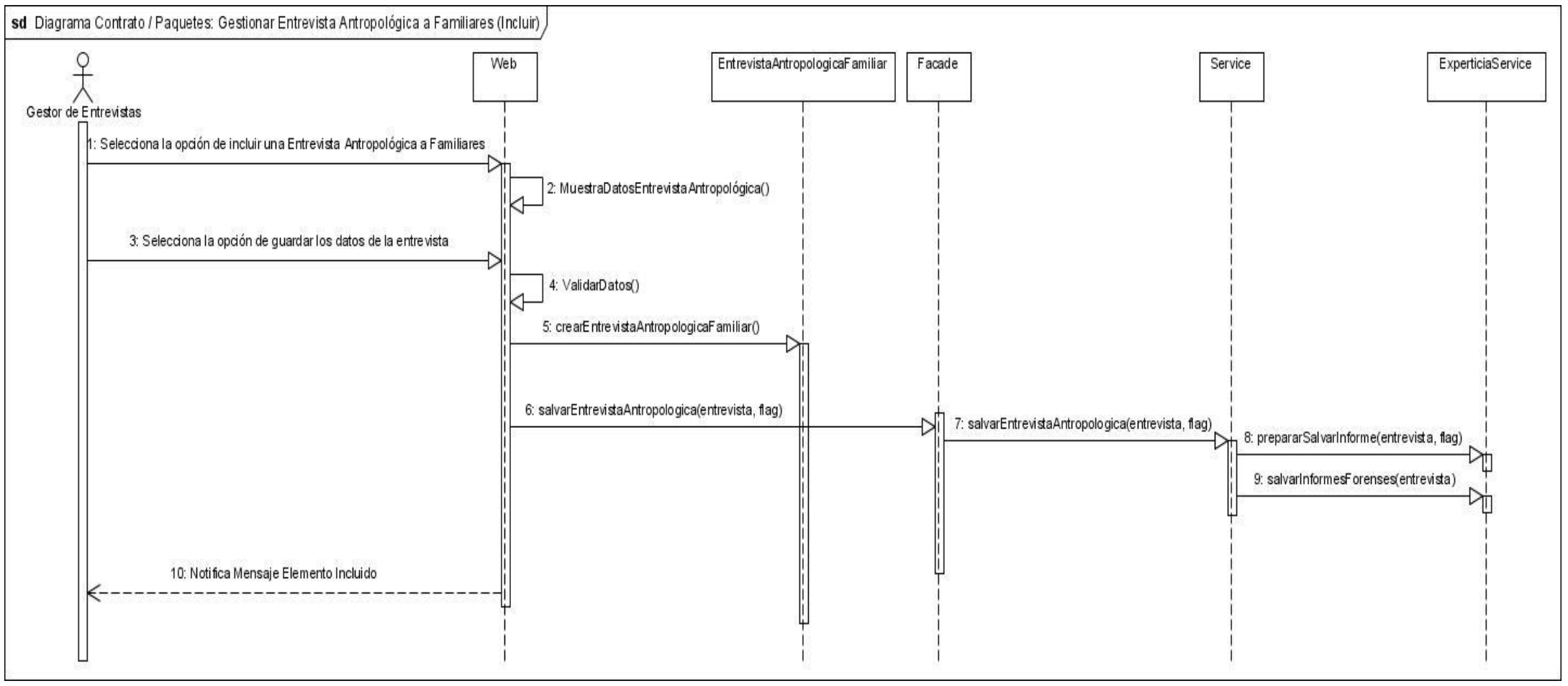



Diagrama de contrato entre paquetes CU Gestionar Entrevista Antropológica a Familiar (Modificar)

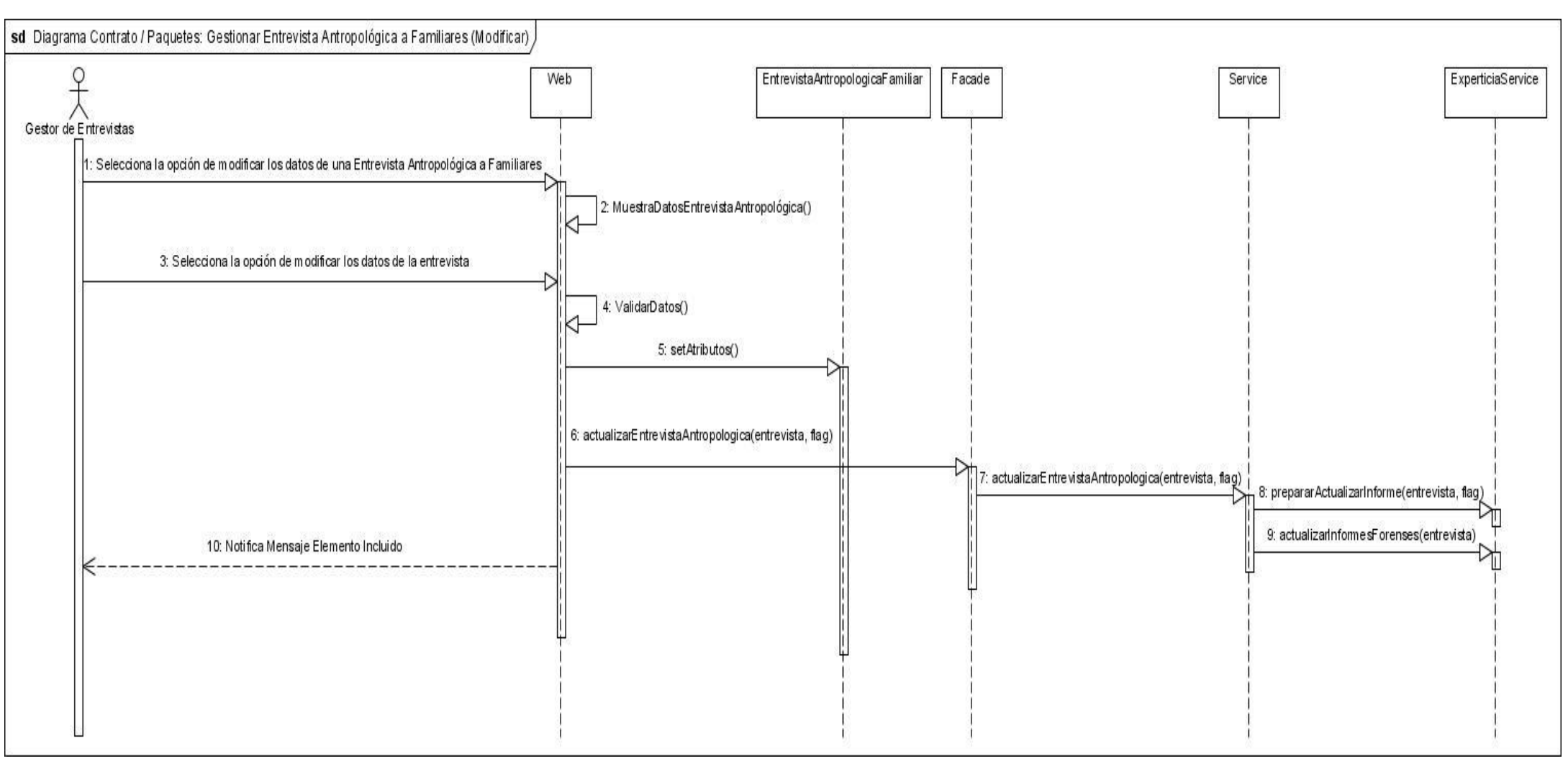
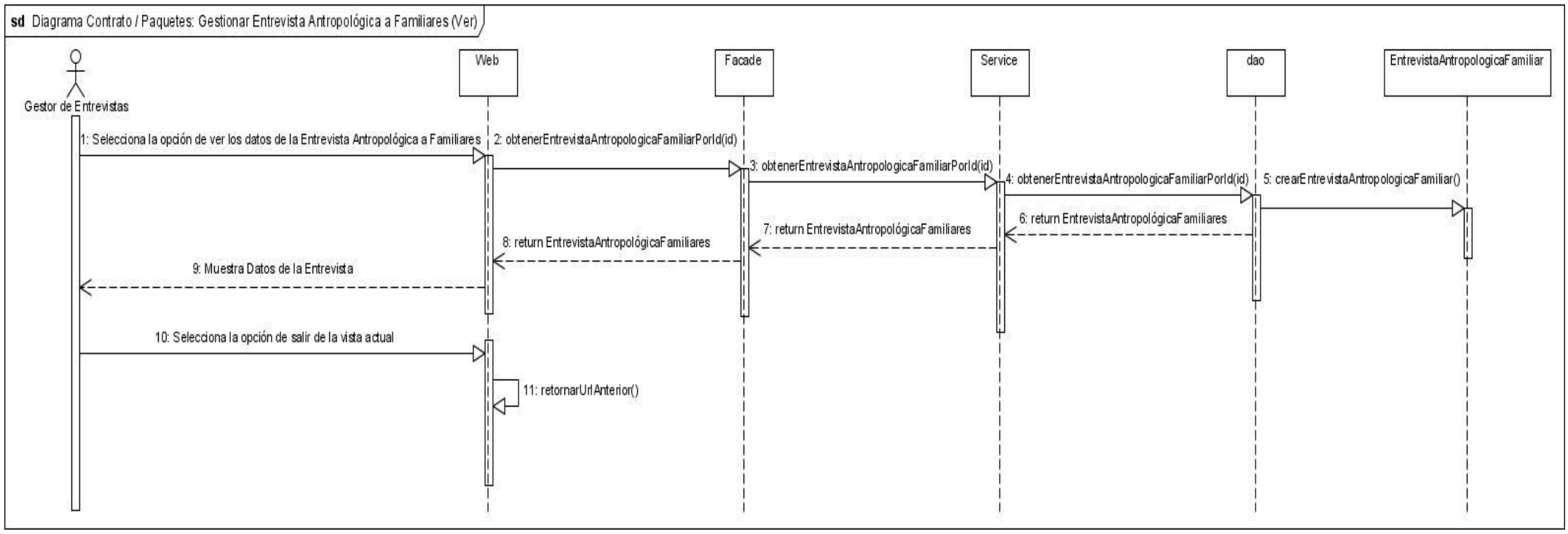


Diagrama de contrato entre paquetes CU Gestionar Entrevista Antropológica a Familiar (Ver)



## ANEXO 5

Diagrama de tablas del modelo relacional