Universidad de las Ciencias Informáticas
Facultad 8

Título: Análisis, diseño e implementación de la capa de lógica de negocio del módulo Análisis de Información del SIIPOL.

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

Autores: Abel Méndez Ortega
Duenytz Sierra Aguila

Tutores: Ing. Susel Ruiz Durán
Ing. Irina Cancela Nieto

Ciudad de la Habana
Junio de 2008

DECLARACIÓN DE AUTORÍA

Declaramos ser los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas en general a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ___ días del mes de _____ del año _____.

Abel Méndez Ortega

_____

Ing. Susel Ruiz Durán

_____

Duenytz Sierra Aguila

_____

Ing. Irina Cancela Nieto

_____

DATOS DE CONTACTO

Nombre y Apellidos: Susel Ruiz Durán.
Fecha de nacimiento: 13 de enero de 1984.
País: Cuba.
Ciudadanía: Cubana.
Carnet de Identidad: 84011308819
Pasaporte: C683644
Correo electrónico: sruiz@uci.cu
Situación laboral: Profesora Adiestrada, Departamento de Programación.
Institución: Universidad de las Ciencias Informáticas, Facultad 8.
Dirección: Carretera San Antonio de los Baños, Torrens, Municipio Boyeros, Ciudad de La Habana, Cuba, Código postal: 19370.
Graduada de la Universidad de las Ciencias Informáticas.
Profesora en el propio centro, desde marzo del año 2006. Ha impartido docencia en las disciplinas de Programación e Ingeniería de Software.
Realizó su trabajo de diploma en el área de las aplicaciones web de gestión, rama en la cual ha centrado desde entonces su labor productiva en la universidad, así como la tutoría de tesis. Ha ejercido como oponente y miembro de tribunales en varias tesis de grado.
Cuenta con experiencia práctica en las actividades correspondientes al rol de Analista y Jefe de Sistema de Software. Actualmente desempeña el rol de Segunda Líder de Software en el proyecto CICPC.

Nombre y Apellidos: Irina Cancela Nieto. Matanzas, Cuba. 1982. graduada de Ingeniera Informática en la Universidad de Matanzas “Camilo Cienfuegos”. Profesora Instructora del Departamento de la Ingeniería y Gestión de Software de la Facultad 8, en la Universidad de Ciencias Informáticas. Ha impartido las asignaturas de Programación, Seguridad Informática, Cursos del 2do Perfil, Historia de la Informática, siendo la coordinadora en la Facultad de esta ultima. A cursados varios postgrados entre los que se encuentran los de La Propiedad Intelectual en el Software Educativo y los Diplomados de Docencia Universitaria y de Líderes de Proyecto, matriculada en la maestría de Gestión de Proyectos todos en la Universidad de las Ciencias Informáticas. Tutoreó de 2 tesis en el pasado curso 2006-2007, en estos momentos se desempeña como líder del proyecto CICPC dedicada totalmente a la Producción. Correo electrónico: irinacn@uci.cu

AGRADECIMIENTOS

A Irina, Susel y Lesky por su gran ayuda en el desempeño del presente trabajo, por lograr la calidad necesaria para el mismo, ya que sin ustedes no hubiese sido posible.

A mi mamita querida por apoyarme tanto en mis estudios, y en mi vida en general.
A mi novia Yadira por su contribución incondicional para la realización de este trabajo, por su opinión oportuna e incansable.
A todos aquellos que de una forma u otra contribuyeron a la realización de este trabajo de diploma.

Duenytz Sierra Aguila.

A mis padres porque siempre me lo dieron todo.
A mis abuelos, hermana y a toda mi familia que siempre me apoyó en todo
A todos mis amigos en especial a todo el edificio 89.
A todos los que contribuyeron para que hoy esté optando por este título.

Abel Méndez Ortega.

DEDICATORIA

A mi mamita linda y querida que es lo más grande en la vida para mí, en todo lo que hago siempre te tengo presente, y solo busco enorgullecerte cada día más.

A mi abuelita Julia, que Dios te tenga bien en alto donde tú te mereces estar, siempre pienso en ti, y en cuanto te hubiese gustado estar aquí para ver lo mucho que tu nietecito querido ha crecido.

A mi abuelo querido por su incansable preocupación por mi futuro, por quererme tanto y ayudar a convertirme en una persona de bien.

A Mauro por ser el mejor padre que hubiera podido tener, por la confianza que ha depositado siempre en mi para todo, y por apreciarme y quererme como un hijo.

A mi hermanita linda Madeleine a quien quiero y adoro tanto.

A mi tío Miguel por darme la inspiración de luchar por un futuro profesional.

A mis amigos todos, en especial al Chino por ser como mi hermano menor y por tantos consejos compartidos.

A Yadira, mi noviecita linda y querida, por compartir tantos momentos lindos y hacerme sentir tan amado en las buenas y en las malas.

…de Duenytz.

A mis padres, a mis hermanas, a mis abuelos y a toda mi familia.

…de Abel.

RESUMEN

El Sistema Integrado de Información Policial, aplicación informática que se utiliza actualmente para el procesamiento de toda la información relacionada con los delitos llevados a cabo en la República Bolivariana de Venezuela, está desprovisto de una serie de funcionalidades que afecta la rapidez con que deben ser gestionados los delitos. En el marco de la colaboración Cuba-Venezuela, se desarrolla un sistema que utilizando las nuevas plataformas tecnológicas, permita modernizar los procesos del Cuerpo de Investigaciones Científicas, Penales y Criminalísticas mejorando el nivel de respuesta a las necesidades de seguridad del ciudadano venezolano.

El presente trabajo constituye la propuesta de un prototipo para la lógica de negocio del módulo Análisis de Información del nuevo Sistema de Investigación e Información Policial centrándose en el análisis, diseño e implementación de dicho módulo, para que se adapte a los requisitos funcionales y no funcionales con que debe contar la aplicación.

La puesta en marcha de esta aplicación en conjunto con el resto del software proveerá a los usuarios finales de una vía rápida y de mucha utilidad para la gestión y el procesamiento de toda la información que se gestiona en el Cuerpo de Investigaciones Científicas, Penales y Criminalísticas, dejando atrás muchas de las dificultades que presentaban en el trabajo cotidiano.

ÍNDICE

INTRODUCCIÓN

La República Bolivariana de Venezuela, liderada por Hugo Chávez Frías, ha estado llevando a cabo una exitosa revolución en estos tiempos, la cual ha cambiado en gran medida la antigua Venezuela por una nueva, capaz de atender de mejor forma las necesidades de todos los sectores del pueblo. Actualmente este país atraviesa una situación delictiva de gran escala que se encuentra presente en todo el territorio venezolano y se ha visto incrementada en el transcurso de los últimos años.

La institución encargada de contrarrestar estos males y de traer la justicia a la sociedad naciente, es el Cuerpo de Investigaciones Científicas, Penales y Criminalísticas (CICPC) de la República Bolivariana de Venezuela. El CICPC actualmente facilita parte de su trabajo usando una aplicación informática: Sistema Integrado de Información Policial, el cual posee una técnica obsoleta que afecta la rapidez con que deben ser gestionados los delitos que se cometen diariamente en Venezuela, además está desprovisto de muchas funcionalidades que podrían estar incluidas para lograr mayor eficiencia y agilidad en los procesos.

En el marco de convenio de colaboración Cuba-Venezuela se le ha dado la tarea a la empresa ALBET del Proyecto Modernización y Transformación del CICPC, el cual entre sus componentes posee el desarrollo de una herramienta informática que permita soportar las decisiones estratégicas del Ministerio del Poder Popular para relaciones Interiores y Justicia y la Dirección General del CICPC, que ayude a controlar y organizar el trabajo en las dependencias del CICPC y mejore el nivel de respuesta a las necesidades de seguridad del ciudadano venezolano acelerando el proceso de la investigación.

Teniendo en cuenta los requerimientos del futuro sistema, así como el modelo de sistema obtenido, la aplicación contará con varios módulos, entre los cuales se encuentra el módulo de Análisis de Información que se encarga de la gestión de toda la información que puede ser colectada durante la investigación de un hecho delictivo. Esta información puede estar dada por los datos de las personas, armas, vehículos y objetos en general que se encuentran relacionados de alguna forma con el delito.

El módulo de Análisis de Información del Sistema de Investigación e Información Policial (SIIPOL) cuenta con una arquitectura basada en capas, que se comunican entre sí y se distribuyen las responsabilidades de la aplicación permitiendo al futuro sistema ser más versátil, ágil y poco complejo para las modificaciones posteriores y su mantenimiento, pues en caso de haber un cambio solo se afecta a la capa requerida abstrayéndose del código de las demás. La atención de este trabajo será enfocada en la capa de lógica de negocio, dando respuesta a la necesidad que existe de analizar e

implementar una capa de lógica de negocio para el módulo de Análisis de Información del SIIPOL donde se cumpla con los requisitos funcionales y no funcionales asociados a dicho módulo.

Por la situación anteriormente expuesta se define el **problema científico** en la siguiente interrogante: ¿Cómo garantizar el cumplimiento de los requisitos funcionales y no funcionales asociados al módulo Análisis de Información del SIIPOL desde el punto de vista de la lógica de negocio?

El **objeto de estudio** del presente trabajo lo conforma el módulo de Análisis de Información del SIIPOL.

El **campo de acción** lo constituye la capa de lógica de negocio del módulo de Análisis de Información del SIIPOL.

El **objetivo general** de este trabajo es analizar el módulo Análisis de Información del SIIPOL, diseñar e implementar una capa de lógica de negocio que se adapte a los requisitos funcionales y no funcionales de la aplicación.

Del objetivo general se derivan como **objetivos específicos** los siguientes:

- Realizar un estudio profundo del Modelo de Casos de Uso del Sistema del módulo de Análisis de Información del SIIPOL, para obtener un punto de partida para la realización del flujo análisis y diseño.
- Diseñar e implementar la capa de lógica de negocio del módulo de Análisis de Información del SIIPOL.
- Garantizar el buen rendimiento del sistema, pues este módulo es usado por el resto de la aplicación y las entidades que maneja son muy grandes, lo que conlleva a que exista un riesgo palpable de no cumplir con los requerimientos no funcionales, específicamente aquellos relacionados con los tiempos de respuesta de la aplicación.

La **idea a defender** quedaría formulada de la siguiente forma: “El análisis, diseño e implementación de la capa de lógica de negocio del módulo Análisis de Información, posibilita la integración de la aplicación gestionando la información sobre la cual se trabaja en CICPC y cumpliendo con los requisitos funcionales y no funcionales del SIIPOL”.

Las **tareas** trazadas para dar solución a los objetivos expuestos anteriormente, consisten en las siguientes:

- Estudio e investigación de las herramientas que se usarán en el desarrollo del presente trabajo, entre ellas, técnicas de programación, lenguajes y frameworks entre otras.
- Estudio del modelo de casos de usos del sistema correspondiente al módulo Análisis de Información realizado por los analistas de sistemas a fin de tener un punto de partida para el presente trabajo.

- Creación de modelos para el análisis y diseño que se adapten a los requisitos funcionales y no funcionales del módulo de Análisis de Información del SIIPOL.
- Implementación de la capa de lógica de negocio del módulo de Análisis de Información del SIIPOL.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

1.1 Introducción.

En este capítulo se hace referencia al concepto de Sistema de Gestión de Información, así como ejemplificar en la actualidad las tendencias de este tipo de sistemas a nivel mundial.
Por otra parte se realiza un análisis comparativo de las principales metodologías existentes, además de hacer un estudio detallado de las tecnologías y herramientas que se utilizarán a lo largo de la presente investigación y su desarrollo.

1.2 Fundamentación Tema.

1.2.1 ¿Qué es un Sistema de Gestión de Información?

Una definición muy acertada para un Sistema de Gestión de Información (SGI) podría consistir en un conjunto de elementos que interactúan entre sí con el fin de apoyar las actividades que se realizan en una organización o para automatizar los procesos de trabajo que se efectúan dentro de la misma (1). Un SGI requiere de las técnicas informáticas para un correcto funcionamiento. La utilización del hardware necesario es de suma importancia para contar con la eficiencia requerida por el SGI. Se debe tener en cuenta además el recurso humano, que es quien interactúa con el sistema y obtiene beneficios del mismo. Los Sistemas de Gestión de Información están caracterizados por reproducir cuatro actividades básicas: entrada, almacenamiento, procesamiento y salida de información.
La entrada de información en un SGI es un proceso donde el sistema recupera datos necesarios para procesar cierta información, esta actividad puede ser de forma manual o automática, dependiendo de su origen puede ser efectuada por un usuario directamente, o por otros sistemas internos o externos a la organización donde está desplegado el sistema en cuestión. El almacenamiento es fundamental en el SGI pues permite persistir toda la información gestionada en el proceso de entrada, haciéndola disponible para otros usuarios y sistemas que tengan privilegios sobre dicha información. El procesamiento de información consiste en la capacidad del SGI para calcular cómo gestionar el flujo de información de acuerdo a restricciones y operaciones establecidas por el lenguaje de programación. Por último el proceso de salida de un SGI es lo que posibilita utilizar la información previamente entrada o almacenada ya sea mostrándola en una interfaz al usuario o sirviendo de entrada a otros sistemas internos o externos (1).

1.2.2 Sistemas de Gestión de Información Policial.

En todos los tiempos la actividad delictiva ha estado presente a nivel mundial en muchas esferas de la vida, en cada etapa en la que le ha tocado vivir la humanidad, se han dispuesto reglas y normas a seguir, ajustándose al momento vivido, para el normal desenvolvimiento de la sociedad. Actualmente la creciente tecnología constituye, para las organizaciones que están en función de la tranquilidad ciudadana, un aporte inmenso a la eficiencia de las mismas, siendo posible por primera vez en la historia, desplegar todo un sistema policial en un país entero, y por tanto solucionar situaciones delictivas de gran envergadura en las distintas localidades en cuestión de minutos. Por otro lado ha existido una fuerte tendencia a integrar informáticamente los diferentes sectores que velan por la seguridad social, así como los que colaboran con estos últimos para hacerlos más eficientes. Existen ejemplos de Sistemas de Gestión de Información en el mundo donde se procura ante todo poder obtener mejoras contribuyentes para los sistemas de seguridad nacional.

Se puede destacar el Proyecto STEGPOL (Sistema Territorial de Emergencias y Gestión Policial) (2), del cual dispone la nación de Chile. Éste tiene como objetivo principal actuar como un Sistema de Información Geográfica con tecnología de punta sobre una Plataforma Nacional Común de Información aplicada al Sistema de Emergencias Nacionales y al Sistema Territorial de Gestión Policial que operan previamente en Chile. La idea principal del Proyecto STEGPOL es integrar a las diferentes entidades como Carabineros, Investigaciones, Ministerio del Interior y Municipios, entre otros, en una Plataforma Nacional Común de Información que permita el intercambio de datos y que sirva de apoyo a la gestión operacional regional o comunal, donde dichas Instituciones estén interconectadas entre sí, en especial con diferentes Divisiones o Departamentos de Carabineros, tales como Jefaturas de Zona, Prefecturas, Comisarías, Sub-Comisarías, Tenencias y por último los Retenes, más el apoyo directo y coordinado entre Carabineros y el área de Seguridad de los Municipios. Hoy en día se han dado todas las condiciones para poder interconectar un gran número de Comisarías, y que todas tengan acceso a una misma Plataforma de Mapas Digitalizados del país, además de las otras Instituciones relacionadas al tema de Seguridad y sus respectivas Unidades Especializadas. Vía la Intranet del Estado o vía comunicación en Banda Ancha – Internet, se puede llegar a acceder al STEGPOL montado en la Web con diferentes claves de acceso restringido tanto para los usuarios operadores encargados de ingresar o modificar datos en línea, como para aquellos que solo tengan acceso a consultar.

Por otro lado en Mérida, España se está llevando a cabo la realización de un SGI para las entidades policiales persiguiendo el objetivo de informatizar todas las tareas que realiza el cuerpo policial, ya que en la actualidad se utiliza la metodología del papel escrito (3). Dentro de unos meses se pretende que los registros de denuncias que hoy se estampan en un libro, sean almacenados en una base de datos

a la que se accederá de forma restringida a través de un ordenador. Además se contará con informes electrónicos sobre cada incidencia, así como también, se dispondrá de un registro informático de los centenares de llamadas telefónicas que se realizan cada año. Muchas policías locales de España ya poseen estos servicios, por lo que este proyecto lograría una integración con el resto del sistema para una mayor eficiencia en el mantenimiento de la seguridad social.

Lo anteriormente expuesto demuestra que mundialmente se ha marcado un hito en las tecnologías asociadas a los sistemas de seguridad policial. La necesidad de informatizar e integrar estos sectores no se hace ajena a ningún país o ciudad dada la demostrada efectividad de estos medios para contrarrestar la creciente actividad delictiva en el mundo.

1.3 Ingeniería de Software.

La ingeniería de software se define según Pressman como una tecnología multicapa en la que se pueden identificar: los métodos, el proceso y las herramientas (4). La ingeniería de software tiene antecedentes en la construcción de software sencillo sin la utilización de guías o metodologías, lo que trajo consigo en el año 1968 la crisis del software. Dicha crisis se debió a que los sistemas no cumplían lo esperado por sus usuarios finales, dado a que poseían errores y colapsaban a menudo. El costo del proceso de producción de software era muy difícil de prever y la modificación de una parte del producto implicaba altos costos sin contar la complejidad y la gran posibilidad de cometer errores, además los recursos no se aprovechaban al máximo y por lo general no se respetaban los plazos de entrega. Lo anterior demostró que hacer un sistema informático era más difícil de lo pensado hasta el momento, asimismo en octubre de ese mismo año se celebra en Garmish, Alemania una conferencia donde se define por primera vez el término Ingeniería de Software que según Fritz Bauer es “Establecer y usar principios de ingeniería orientados a obtener software de manera económica, fiable y que funcione eficientemente sobre máquinas reales.” Desde entonces se han hecho muchos esfuerzos para identificar las causas de los problemas que han existido y para definir formas viables para la producción y mantenimiento del software de manera que se cree con calidad y éste cubra las expectativas de los clientes.

1.3.1 Metodologías existentes. Estudio Comparativo.

La industria del software ha evolucionado en los últimos tiempos de tal manera, que ha sido necesario desarrollar y optimizar a la par modelos y metodologías para sostener la demanda de producción de sistemas cada vez mayores en complejidad y tamaño, logrando su construcción de forma óptima y

eficiente. Como concepto de Metodología se puede citar el siguiente: se encarga de elaborar estrategias de desarrollo de software que promuevan prácticas adoptativas en vez de predictivas; centradas en las personas o los equipos, orientadas hacia la funcionalidad y la entrega, de comunicación intensiva y que requieren implicación directa del cliente (5).

Hoy en día, existen dos grupos dentro de los cuales se pueden clasificar las metodologías. Por una parte existen las metodologías tradicionales, en las cuales lo principal es el control del proceso, a través de una planificación exhaustiva, donde se controlan las actividades que se realizarán, los artefactos que se generarán, además de las herramientas y notaciones que serán usadas. Se caracterizan por comenzar con la obtención y análisis de los requerimientos solicitados por el usuario, luego de una intensa interacción con los usuarios y clientes, se definen los requerimientos funcionales y no funcionales del futuro sistema. Estas metodologías están basadas en la producción de proyectos de larga duración, lo que permite estructurar un amplio equipo de trabajo en roles que cumplirían diferentes funciones dentro de la producción. Dentro de estas metodologías se encuentra Rational Unified Process (RUP), la cual es muy usada en la actualidad pues goza de excelente prestigio dentro del mundo informático por los éxitos alcanzados.

El otro gran grupo en que se clasifican las metodologías sería el de las ágiles, las cuales surgidas por la incapacidad que tienen las tradicionales de operar en condiciones volátiles, se caracterizan por una gran reducción de los tiempos de desarrollo del software, planteando que es más importante la producción de un software funcional que procesar una documentación excesiva. Además no se sigue un plan estricto por lo que se posee una alta capacidad de respuesta a un cambio, Todo esto las hace más flexibles que las anteriores. También se cuenta con el cliente como un trabajador más del equipo de trabajo, pues colabora de forma ininterrumpida con la realización del software. Las metodologías ágiles han presentado éxito en proyectos de poco alcance, donde el equipo de desarrollo no necesita ser amplio y se requiere la entrega inmediata del producto. Una metodología claramente visible en este grupo es eXtreme Programing (XP).

1.3.2 Justificación de la propuesta seleccionada.

El éxito de un proyecto está estrechamente ligado a la metodología que se adopte para su realización, pero resulta difícil en ocasiones determinar cuál es la idónea para el desarrollo exitoso del proyecto en cuestión, por tanto la elección depende mucho de las características en que se deba desarrollar el mismo. En el caso del proyecto de modernización del CICPC y el desarrollo del nuevo SIIPOL, se debe analizar primeramente que su realización será efectuada entre Cuba y Venezuela, países los cuales se

encuentran a una distancia considerable entre sí, de lo que se deduce que el cliente no podrá jugar un papel dedicado en el desarrollo del sistema. Por otra parte es un proyecto de gran magnitud con un alcance bien definido, por lo que se sabría a ciencia cierta su duración, los plazos de entrega, así como el presupuesto estimado del que se debe disponer. Además se cuenta con una gran masa de desarrolladores los cuales jugarían diferentes roles dentro de la producción. Por las características anteriormente citadas se hace factible usar la metodología RUP catalogada dentro de las metodologías tradicionales. A continuación se hace un estudio más detallado de esta metodología.

1.3.3 El Proceso Unificado de Desarrollo de Software.

El Proceso Unificado de Desarrollo de Software junto al Lenguaje Unificado de Modelado (UML), forman la metodología estándar más utilizada para el análisis, diseño, implementación y documentación de sistemas orientados a objetos. Esta metodología constituye una guía rectora que decide quién hace qué, cuándo y cómo lo hace. RUP dirige las actividades que se realizan por rol, les da un orden, especifica qué artefactos deberían ser desarrollados y puede además monitorear y medir los productos y actividades de un proyecto, teniendo en cuenta la calidad del producto final. RUP se caracteriza por dividir el ciclo de vida de la producción del software en 4 fases:

- Inicio o Conceptualización: es donde se determina la visión del proyecto, o sea se comprende el entorno y se determina el alcance del producto.
- Elaboración: en esta etapa se determinan los cimientos de la arquitectura y se analiza el dominio del problema.
- Construcción: en esta fase se obtiene la capacidad operacional inicial del producto.
- Transición: se obtiene el release o liberación del producto y se pone en manos de los usuarios finales.

Además de esto cuenta con 9 flujos de trabajo, 6 principales y 3 de soporte los cuales son: Modelamiento del negocio, Captura de requerimientos del sistema, Análisis y Diseño, Implementación, Prueba, Despliegue, Gestión de configuración y cambios, Gestión de Proyectos y Entorno, donde los 3 últimos constituyen los flujos de soporte. Esta metodología es usada en proyectos de gran envergadura que impliquen elevados tiempos de desarrollo y que posean un equipo de desarrollo grande.

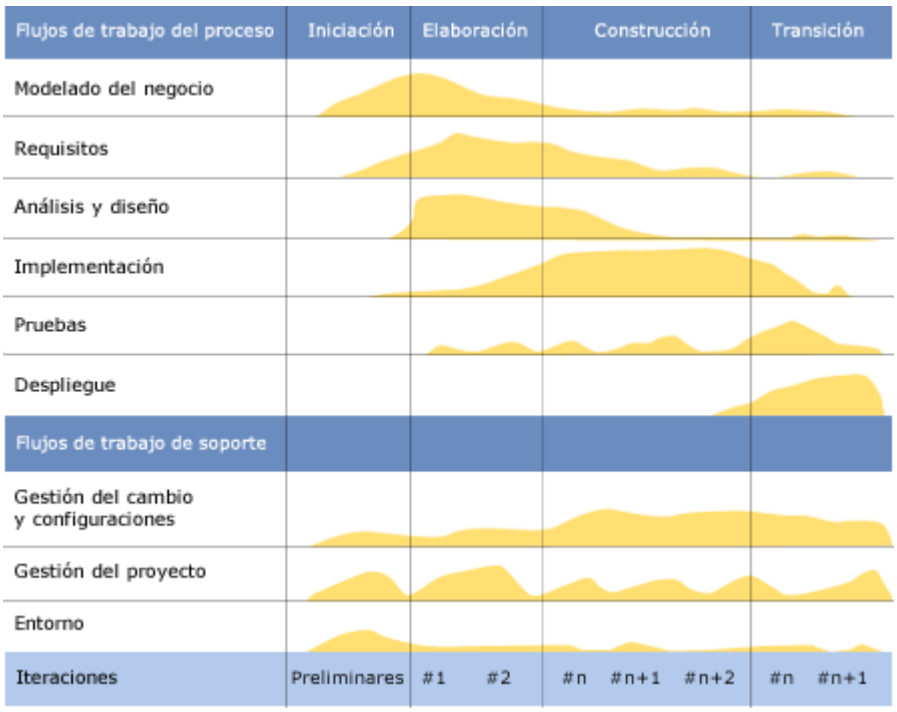

Fig.1: Fases y Flujos de trabajo de RUP

Los elementos característicos del RUP son:

- Actividades: Son los procesos que se llegan a realizar en cada iteración.
- Trabajadores: Son las personas o entidades involucradas en cada proceso.
- Artefactos: Un artefacto puede ser un modelo, o un elemento de modelo, un documento, en fin todo lo que puede ser generado en el proceso.

Sus características principales son:

- Dirigido por casos de uso. Los casos de uso guían el proceso de desarrollo pues los modelos que se obtienen representan la realización de los mismos.
- Centrado en arquitectura. La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo.
- Iterativo e Incremental. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros. En el caso de una iteración de la fase elaboración, se centra la atención en el análisis y diseño, a la vez que se refinan los requerimientos y se obtiene un producto con un determinado nivel, que irá creciendo incrementalmente en cada iteración.

1.3.4 Lenguaje Unificado de Modelado (UML).

El lenguaje unificado de modelado (UML), es un lenguaje de modelado visual que permite visualizar, especificar, construir y documentar los artefactos que se generan en el proceso de modelado y construcción de un software (6). Surge en octubre del año 1994, cuando Rumbaugh se unió a la compañía Rational fundada por Booch. Empezaron a trabajar conjuntamente para unificar el Booch y la OMT (Object Modeling Tool), dos métodos creados anteriormente por ellos. En el año 1995 se une también a la compañía otra personalidad de la investigación en el área de metodologías de software, Jacobson, quien hizo aportes al lenguaje UML. Además el lenguaje se abrió a otras compañías para que hicieran aportes, el resultado fue el surgimiento de la primera versión de dicho lenguaje, la 0.8, esta versión se pone a disposición de un grupo de trabajo, para en el año 1997 convertirse en el estándar del OMG (Object Management Group) posteriormente pasó por una serie de cambios menores llegando hasta la versión 2.0 muy usada y difundida en la actualidad.

Las funciones principales de UML se pueden apreciar como siguen:

- Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.
- Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

1.4 Plataformas para el desarrollo de software empresarial.

El avance que han tenido las aplicaciones empresariales, la contribución tanto entre departamentos como entre empresas y el desarrollo de aplicaciones y servicios web han sufrido un aumento muy importante durante los últimos años. Ante esta nueva demanda surgen dos plataformas distintas para el desarrollo de este tipo de aplicaciones: J2EE de Sun Microsystems y .NET de Microsoft.

1.4.1 La plataforma J2EE.

Para empezar el análisis de J2EE (Java 2 Platform, Enterprise Edition) es necesario tener una breve idea acerca de lo que es Java. El lenguaje Java surge a principios de los años 90 en los laboratorios

de Sun Microsystems. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel como punteros.

El lenguaje Java se creó con cinco objetivos principales:

1. Debería usar la metodología de la programación orientada a objetos.
2. Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos.
3. Debería incluir por defecto soporte para trabajo en red.
4. Debería diseñarse para ejecutar código en sistemas remotos de forma segura.
5. Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

A diferencia de los lenguajes convencionales, Java es compilado a un código intermedio, el cual es interpretado por una máquina virtual de Java. La máquina virtual hace posible que una aplicación que haya sido implementada en Java se ejecute en cualquier sistema operativo con soporte para la máquina virtual. La máquina virtual proporciona un entorno de ejecución que convierte el código neutro de Java al código nativo del ambiente en que está siendo ejecutada. Java es un lenguaje de programación multipropósito que tiene todas las características de un ambiente orientado a objetos: es sencillo, robusto, seguro, cuenta con capacidad de generación de aplicaciones distribuidas de arquitectura neutral, portable, multihilo, dinámico y de alto rendimiento. Además, la API (Application Program Interface, interfaz de programas de aplicación) de Java está formada por un conjunto de paquetes de clases que le proporcionan una extensa funcionalidad. El núcleo de la API viene con cada una de las implementaciones de la máquina virtual: tipos de datos, clases y objetos, manejo de red, seguridad, componentes, etc. Estos componentes son llamados Java Beans, los cuales son código reusable que se pueden desarrollar fácilmente para crear aplicaciones sofisticadas. Con Java, Sun Microsystems introdujo en el mercado la primera plataforma de software universal diseñada desde y para el crecimiento de Internet y de las intranets corporativas. Esta tecnología permite escribir aplicaciones una sola vez y ejecutarlas en cualquier computadora, lo que, desde entonces ha revolucionado el mundo del desarrollo de software por representar un cambio de paradigma.

En el mercado de desarrollo de software surge la necesidad de contar con medios y herramientas que permitan construir aplicaciones corporativas, y entonces en respuesta a esto, se diseñó la plataforma abierta y estándar de Java, mejor conocida como J2EE (Java 2 Enterprise Edition, Java 2 edición empresarial). Se le denomina plataforma porque proporciona técnicas específicas que describen el lenguaje, pero, además, provee las herramientas para implementar productos de software basados en dichas especificaciones.

J2EE ha sido diseñada para aplicaciones distribuidas con base en componentes o unidades funcionales de software que interactúan entre sí para formar parte de una aplicación empresarial J2EE.

Un componente de esta plataforma debe formar parte de una aplicación y ser desplegado en un contenedor, o sea, en la parte del servidor J2EE que le ofrece al componente ciertos servicios de bajo nivel y de sistema, tales como seguridad, manejo de concurrencia, persistencia y transacciones. Como se puede apreciar, J2EE no es solo una plataforma o una tecnología, sino un estándar de desarrollo, construcción y despliegue de aplicaciones.

Uno de los grandes beneficios de J2EE como plataforma es que es posible empezar con poco o ningún coste. Su implementación, de Sun Microsystems, puede ser descargada gratuitamente, y hay muchas herramientas disponibles de código abierto para extender la plataforma o para simplificar el desarrollo.

J2EE incluye varias especificaciones de API, tales como JDBC, RMI, e-mail, JMS, Servicios Web, XML, etcétera, y define cómo coordinarlos. Además configura algunas especificaciones únicas para componentes. Éstas incluyen Enterprise JavaBeans, servlets, portlets (siguiendo la especificación de Portlets Java), JavaServer Pages y varias tecnologías de servicios web, lo que permite al desarrollador crear una aplicación de empresa portable entre plataformas y escalable, a la vez que integrable con tecnologías anteriores. Otros beneficios añadidos son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados, significando que los desarrolladores pueden concentrarse más en la lógica de negocio de los componentes en lugar de tareas de mantenimiento de bajo nivel.

En el 2005 se calcula en 4,5 millones el número de desarrolladores y 2.500 millones de dispositivos habilitados con tecnología Java. Entre noviembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre (aunque la biblioteca de clases de Sun que se requiere para ejecutar los programas Java todavía no es software libre).

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
- 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, un IDE desarrollado por Borland

1.4.2 El modelo de desarrollo J2EE.

La plataforma J2EE define un modelo de programación encaminado a la creación de aplicaciones basadas en n-capas. La lógica de la aplicación se divide en componentes de diferentes funciones que componen una aplicación J2EE y que están distribuidos en dependencia de la capa en el ambiente multicapas J2EE al cual la aplicación pertenece. Aunque puede variar, típicamente una aplicación suele tener cinco capas diferentes:

- Capa cliente: representa la interfaz de usuario que maneja el cliente.
- Capa de presentación: representa el conjunto de componentes que generan la información que se mostrará en la interfaz de usuario del cliente.
- Capa de lógica de negocio: contiene los componentes de negocio reutilizables.
- Capa de integración: aquí se encuentran los componentes que permiten hacer más transparente el acceso a la capa de sistemas de información. Este es el lugar idóneo para implementar la lógica de objetos de acceso a datos.
- Capa de recursos: engloba los sistemas en los cuales la información se almacena físicamente como bases de datos relacionales, bases de datos orientadas a objetos, bancos de ficheros de datos, etc.

Las ventajas de un modelo como este son muy importantes. Al tener las capas separadas existe poco acoplamiento entre las mismas, de modo que es mucho más simple hacer modificaciones en ellas sin que afecten a las demás. Todo esto causa la obtención de mejoras en cuanto a mantenibilidad, extensibilidad y reutilización de componentes. Otra ventaja que se obtiene es la de promover la heterogeneidad de los clientes, ya que añadir nuevos tipos de clientes se reduce a añadir nuevas capas de interfaz de usuario y presentación, sin necesidad de modificar el resto de las capas.

El modelo de desarrollo con J2EE está basado en componentes reutilizables, con el objetivo de aumentar la reusabilidad de las aplicaciones. Estos componentes gracias a las especificaciones, son

intercambiables entre servidores de aplicaciones, por lo que la portabilidad de las aplicaciones es máxima.

1.4.3 La plataforma J2EE en el mundo corporativo.

La plataforma J2EE es un conjunto de herramientas que crean un escenario ideal para el desarrollo y despliegue de aplicaciones escalables en la Web, por todo esto resulta una propuesta atractiva, interesante y de vanguardia que responde, de manera natural, a la demanda actual para el desarrollo de software bajo el concepto de arquitectura en capas.

Para una compañía que cuenta con una diversidad de equipos de automatización, con sus respectivos sistemas operativos, una diversidad de ambientes de trabajo en las distintas áreas y una diversidad de entornos de desarrollos, los sistemas y soluciones informáticas con que se cuenta en cada lugar, la propuesta de J2EE para unificar el desarrollo de aplicaciones que puedan utilizarse en la empresa resulta viable y con posibilidades de amplia aceptación.

1.4.4 La plataforma .NET.

.NET es un proyecto de Microsoft para crear una nueva plataforma de desarrollo de software con énfasis en transparencia de redes, con independencia de plataforma de hardware y que permita un rápido desarrollo de aplicaciones. Basado en ella, la empresa intenta desarrollar una estrategia horizontal que integre todos sus productos, desde el sistema operativo hasta las herramientas de mercado.

.NET podría considerarse una respuesta de Microsoft al creciente mercado de los negocios en entornos Web, como competencia a la plataforma Java de Sun Microsystems. Su propuesta es ofrecer una manera rápida y económica, a la vez que segura y robusta, de desarrollar aplicaciones permitiendo una integración más rápida y ágil entre empresas y un acceso más simple y universal a todo tipo de información desde cualquier tipo de dispositivo.

.NET es una plataforma de software que conecta información, sistemas, personas y dispositivos. La plataforma .NET conecta una grande variedad de tecnologías de uso personal y de negocios, de teléfonos celulares a servidores corporativos, permitiendo el acceso a información importante, donde y cuando se necesite.

Desarrollado con base en los estándares de Servicios Web XML, .NET permite que los sistemas y aplicaciones, ya sea nuevos o existentes, conecten sus datos y transacciones independientemente del

sistema operativo, tipo de computadora o dispositivo móvil que se utilice, o del lenguaje de programación empleados para crearlo.

La idea fundamental de Microsoft .NET es un cambio de enfoque en lo que es la informática, pasando de un mundo de aplicaciones, sitios Web y dispositivos aislados a una infinidad de computadoras, dispositivos, transacciones y servicios que se conectan directamente y trabajan en conjunto para ofrecer soluciones más amplias y ricas en contenido.

Los Servicios Web son la más innovadora tecnología para los negocios en la Web. Los Servicios Web XML utilizan tecnologías programables y reutilizables que aprovechan la flexibilidad de Internet. Con ellos es posible tener una infinidad de aplicaciones conectados en red, ya sea que se ejecuten en diferentes plataformas, proporcionando información a todos s sus clientes, socios de negocios y empleados. Los Servicios Web tienen como base un conjunto de estándares abiertos, incluyendo XML, SOAP, WSDL y UDDI, los cuales son controlados por el World Wide Web Consortium (W3C). Desde el punto de vista de un desarrollador, .NET facilita la escritura de sistemas capaces de conectar entre sí utilizando Microsoft Visual Studio .NET, .NET Framework y los XML Web Services.

1.4.5 Comparación entre J2EE y .NET.

J2EE es una iniciativa de Sun Microsystems secundada por muchas más empresas como IBM, HP, BEA, ORACLE (en la actualidad la apoyan más de 400 empresas) y creada en 1996, mientras que .NET es la alternativa creada por el gigante Microsoft cuatro años más tarde. Para ello, ha creado una plataforma de desarrollo cerrada dentro de su paquete “Visual Studio .NET”, que incluye todo lo necesario para crear aplicaciones de este tipo. Sin embargo, J2EE es un conjunto de especificaciones definidas como estándar, que deben ser seguidas para desarrollar el producto, para ello es necesario adquirir una serie de recursos que, en su conjunto, permitirán desarrollar las aplicaciones: el JRE, el JDK (librerías), servidores web y de aplicaciones como IBM WebSphere, BEA Weblogic, Oracle9iAS, Sun ONE, JBoss, Apache Tomcat, Jetty, Resin, Apache Gerónimo, entornos de programación, etc. Por tanto, se muestra la primera diferencia entre estas dos plataformas es precisamente esa: .NET es un producto, mientras que J2EE es un conjunto de especificaciones que definen un estándar.

.NET cuenta con un ambiente de desarrollo llamado Visual Studio .NET, mientras que J2EE no es un producto de ninguna empresa, se trata en cambio de un estándar, por lo que no cuenta con un entorno de desarrollo tipo “Visual Studio”. Existen como alternativa en el mercado múltiples productos que ofrecen entornos de desarrollo adecuados, tales como NetBeans de Sun Microsystems, Visual Café de WebGain, Visual Age for Java de IBM, IBM WebSphere, Eclipse, entre otros. Es bien sabido que la

mayoría de estos entornos de desarrollo no ofrecen las prestaciones y las posibilidades que brinda Visual Studio .NET.

Tanto .NET como J2EE tienen un fin común pero metodologías básicamente diferentes a la hora de abordar problemas como la seguridad, portabilidad, etc. En el aspecto funcional ambas plataformas guardan varias similitudes; se programa en un lenguaje que luego se compila a un código intermedio ("Intermediate Language" en el caso de Microsoft .NET y "Bytecodes" en el caso de Java). Este código se ejecutará en un "entorno de ejecución" que transformará el lenguaje intermedio a código propio de la máquina en la que se corre la aplicación, Common Language Runtime (CLR) en Microsoft .NET y Java Runtime Environment (JRE) en J2EE.

Una de las principales características de la plataforma .NET consiste en la posibilidad de programar los distintos componentes de una aplicación empleando distintos lenguajes. Es posible programar en una gran cantidad de lenguajes como C# (su lenguaje principal), Visual Basic, C++, entre otros. Pero .NET va más allá de soportar estos lenguajes; también ofrece plena interoperabilidad entre ellos, todo esto permite que sea posible construir un componente en un lenguaje, introducirlo en una aplicación escrita en otro distinto e incluso heredarlo y añadir nuevas características en un tercero. J2EE, por su parte, el único lenguaje que soporta es Java y es el que se tendrá que utilizar para desarrollo de todos los componentes. Existen sólo dos formas oficiales para acceder a la plataforma J2EE con otros lenguajes, la primera es a través de JNI (Java Native Interface) y la segunda es a través de la interoperabilidad que ofrece CORBA.

El rendimiento es uno de los temas más controvertidos a la hora de comparar estas dos plataformas. En aspectos fundamentales del entorno de ejecución, como son el rendimiento, la escalabilidad y la seguridad, J2EE sigue teniendo fama de estar por delante de .NET, y debido a esto podemos decir que en los entornos en los que éstos son los aspectos fundamentales, como pueden ser los entornos transaccionales de las grandes multinacionales, J2EE suele ser la plataforma escogida. La gran experiencia y madurez de la plataforma juega a favor de J2EE, ya que salió al mercado algunos años antes que .NET, y en todo este tiempo se han ido desarrollando multitud de productos y servicios al tiempo que se han ido corrigiendo errores y cubriendo las carencias y necesidades detectadas, por lo que actualmente cuenta con una gama de productos altamente consolidados mientras que .NET tiene menos experiencia a sus espaldas. J2EE presentaba cierta desventaja por el hecho de restringir la programación a un único lenguaje, Java. Aunque también tiene una gran ventana respecto a su rival: la portabilidad, o la posibilidad de ejecutar las aplicaciones desarrolladas en cualquier sistema operativo y máquina del mercado. Los productos J2EE ofrecen mucha más portabilidad que .NET, que sólo está preparada para ejecutarse sobre plataformas Microsoft (Windows). También hay que señalar que,

como era de suponer, la plataforma de Microsoft está en vías de salvar esta circunstancia gracias a proyectos como MONO, un proyecto de código abierto para crear un grupo de herramientas libres, basadas en GNU/Linux y compatibles con .NET según lo especificado por el ECMA.

La seguridad es, sin lugar a dudas, uno de los aspectos más importantes a la hora de evaluar las dos plataformas. J2EE y .NET proporcionan servicios de seguridad sencillos, aunque con enfoques diferentes.

Ambas plataformas usan conceptos similares para manipular el acceso a los recursos por usuario y por código, basándose ambos en permisos. Además, se usa el concepto de perfiles en ambos. Mientras J2EE usa el concepto de "Perfiles Organizacionales" para delimitar responsabilidades a varios niveles del proceso de desarrollo y explotación (Product Provider, Application Component Provider, Application Assembler, Deployer y System's Administrador, por defecto), .NET no define la jerarquía tan claramente.

Luego de un análisis de ambas plataformas de desarrollo y teniendo en cuenta la necesidad de trabajar con una de ellas, el cliente seleccionó la plataforma J2EE, pues resultó mostrar mayor compatibilidad con parte de los productos que normalmente operan en Venezuela actualmente y además por ser una tecnología que opera bajo licencia abierta.

1.5 Herramientas para el desarrollo de aplicaciones J2EE.

Para el desarrollo de aplicaciones J2EE se realizó un estudio de las posibles herramientas a utilizar. Teniéndose en cuenta la tendencia actual y las novedades de cada una de ellas.

1.5.1 Visual Paradigm.

Una guía de cómo usar UML es la metodología RUP, Rational es una herramienta propuesta por IBM que une la metodología con la representación visual. Actualmente la versión de Rational 7.0 permite integración con herramientas IDE y la generación de código para el lenguaje Java pero en el momento de la selección de estas herramientas esta versión no estaba disponible en el mercado; sin embargo, y a pesar de que es una de las más conocidas en el mundo, no es la única.

Visual Paradigm (VP) es una herramienta CASE que usa UML como lenguaje de modelado y además de su interfaz amigable se integra con herramientas como Eclipse para Java.

Visual Paradigm ofrece distintas funcionalidades como:

- Entorno de creación de diagramas para UML 2.0

- Diseño centrado en casos de uso y enfocado al negocio generando un software de mayor calidad
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa en su versión profesional, e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo
- Disponibilidad de múltiples versiones, para cada necesidad.
- Disponibilidad de integrarse en los principales IDE.
- Disponibilidad en múltiples plataformas

1.5.2 Diagramas en Visual Paradigm.

Un diagrama es una representación gráfica de una colección de elementos de modelado. Existen diferentes tipos de diagrama que permiten ver el sistema desde diferentes perspectivas. Con VP se pueden incluir los siguientes diagramas:

- Componentes: Describen la organización de los elementos físicos que implementan sistema.
- Despliegue: Describen la configuración del entorno de máquinas y redes sobre el que se distribuyen componentes y procesos del sistema.
- Secuencia: Describen la interacción entre elementos del sistema en el tiempo.
- Casos de Uso: Representan la funcionalidad del sistema.
- Clase: Describen la estructura (estática) del sistema.
- Actividad: Describen cómo se desarrolla un flujo de actividades entre elementos del sistema o del dominio.
- Estado: Describen el estado, condiciones y respuesta de los elementos del sistema.

1.5.3 Eclipse IDE.

Eclipse comenzó como un proyecto de IBM Canadá. Fue desarrollado por OTI (Object Technology International). Esta plataforma, típicamente ha sido usada para desarrollar un Entorno Integrado de Desarrollo (En Inglés: IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se embarca como parte de Eclipse y que son usados también para desarrollar el mismo Eclipse. Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. En noviembre del 2001, se formó un consorcio para el desarrollo futuro de Eclipse como código abierto. En 2003, la fundación independiente de IBM fue creada. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que

fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

La versión actual de Eclipse dispone de las siguientes características:

- Editor de texto.
- Resaltado de sintaxis.
- Compilación en tiempo real.
- Pruebas unitarias con JUnit.
- Control de versiones con CVS.
- Integración con Ant.
- Asistentes (wizards): para creación de proyectos, clases, tests, etc.
- Refactorización.

Asimismo, a través de "plugins" libremente disponibles es posible añadir:

- Control de versiones con Subversion, vía Subclipse.
- Integración con Spring, vía Spring Framework.
- Integración con Hibernate, vía Hibernate Tools.

Además del Eclipse se estudió la posibilidad de emplear NetBeans como entorno de desarrollo. NetBeans, a pesar de ser de código abierto y libre no tiene las facilidades brindadas por Eclipse para integrar elementos que faciliten el desarrollo de aplicaciones web. Es por esto que el equipo de desarrollo eligió Eclipse, debido a que es un entorno de desarrollo de código abierto, es adaptable dado a su posible integración con cualquier plugin, además es muy usado actualmente por lo que goza de una experiencia enorme. Permite integrarse con Visual Paradigm para lograr la generación de código desde el diseño UML lo cual ahorraría tiempo de desarrollo en la producción.

1.6 Principales Frameworks para Capa de Lógica de Negocio.

Para abordar el tema que concierne a esta sección una precondición importante es el conocimiento del significado del término Framework.

Framework es una estructura de soporte definida en la cual un proyecto de software puede ser organizado y desarrollado. Básicamente, 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.

La selección de un conjunto de frameworks para integrarlos y dar soporte a la arquitectura del proyecto, que se va del alcance de los autores, constituye un paso esencial al concebir un proyecto de las características del nuevo SIIPOL, una vez más conscientes de la importancia de esta tarea, es

necesario justificar su uso. A continuación se presentan los principales frameworks para la capa de lógica de negocio.

1.6.1 Spring Framework.

Spring es un Framework de código abierto para el desarrollo de aplicaciones para la plataforma Java, que interviene en todas las capas arquitectónicas de una aplicación J2EE, brinda soporte a varios frameworks de presentación, entre ellos Java Server Faces (JSF), brinda soporte a Hibernate integrándose con ellos para formar una poderosa herramienta para el desarrollo de aplicaciones empresariales.

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. Por su diseño el Framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar, prácticas comunes en la industria. La simplificación del desarrollo de aplicaciones y de sus respectivas pruebas es una de las claves de su éxito. Spring puede organizar de forma efectiva los objetos de la capa central y manejar sus conexiones. Además puede eliminar la proliferación de solitarios y facilita unas buenas prácticas de programación orientada a objetos, por ejemplo utilizando interfaces. Este Framework se sustenta en dos características básicas en su núcleo: Inversión de Control (IoC) y la Programación Orientada a Aspectos (AOP) (7).

La inversión de control permite inyectar las dependencias en un bean al momento de su creación usando un manejador externo. El bean sólo necesita definir la propiedad requerida en su código así como el método de establecimiento (set() method). La fuente primaria de la inyección de dependencias es un archivo de configuración en formato XML, lo que promueve el bajo acoplamiento de las clases.

La programación orientada a aspectos permite implementar la mayoría de los servicios comunes, como son el manejo de transacciones, seguridad, logging, entre otros, que pueden ser aplicados en múltiples componentes. Spring utiliza AOP además para ofrecer manejo de transacciones declarativo sin utilizar un contenedor EJB. El control transaccional de Spring no está atado a JTA (Java Transaction API) y puede funcionar con diferentes estrategias de transacción. AOP permite además a los desarrolladores la facilidad de implementar sus propios aspectos personalizados (8).

Spring básicamente es un contenedor, que se encarga de gestionar y administrar el ciclo de vida y configuración de las clases de la aplicación.

Se debe destacar la magnífica integración que Spring posee con marcos de trabajo de mapeo O/R, especialmente con Hibernate, además de disponer de una potente abstracción a JDBC (Java

Database Connectivity). Spring ofrece un manejo seguro y eficiente de sesiones Hibernate, maneja la configuración de la SessionFactory de Hibernate y las fuentes de datos JDBC en el contexto de la aplicación, lo que hace que la aplicación sea más fácil de probar (9).
Spring es un entorno diseñado para aumentar la productividad, liberando al desarrollador de tareas repetitivas, ayudándolo a hacer diseños más consistentes y limpios. Es maduro y muy amplio, es de destacar que las grandes compañías lo consideran el Framework líder.

1.6.2 JBoss Seam.

Seam es un potente Framework de java que unifica la integración de diversas tecnologías, tales como: AJAX, JSF, Enterprise Java Beans (EJB 3.0), Web Services y administración de procesos de negocio (BPM). Surgió con el objetivo de eliminar la complejidad de la arquitectura y del API, habilitando a los desarrolladores para que puedan desarrollar complejos sistemas con simples anotaciones en los POJOs y poco código XML. Soporta Rich-Faces y ICE–Faces, dos soluciones open source de JSF basado en AJAX. Se inclina más por las anotaciones que por el XML; con las anotaciones, EJB 3.0 se encarga de brindarle la información necesaria al contexto, no tanto JSF que aun depende del XML para definir sus reglas de navegación.

1.6.3 Justificación de la Propuesta Seleccionada.

La selección de uno u otro Framework de esta categoría, es arbitraria, cualquiera de los frameworks se integraría correctamente a este proyecto, aunque los responsables de la arquitectura del sistema se basaron en la experiencia de otras producciones de software similares que acreditan su uso, para seleccionar Spring.

1.7 Conclusión.

En el capítulo se realizó un estudio sobre los Sistemas de Gestión de Información Policial a nivel mundial y sus tendencias para el futuro. Se hizo un estudio conceptual y comparativo de muchas de las metodologías, tecnologías y herramientas existentes en el mundo informático actual, dentro de las cuales, por convenios con el cliente y características del sistema a automatizar, se eligieron los candidatos más apropiados para guiar e implementar el proceso productivo en cuestión.

CAPÍTULO 2: ANÁLISIS Y DISEÑO DEL SISTEMA

2.1 Introducción.

En el presente capítulo se realiza un breve estudio del modelo de Casos de Usos del Sistema (CUS) del módulo de Análisis de Información del SIIPOL, listando y resumiendo los CUS significativos dentro del campo de acción en el cual se enfoca este trabajo, pudiendo constituir la base para proponer la solución al problema planteado.

Además se abordan temas específicos sobre los modelos de análisis y diseño de la capa de lógica de negocio del módulo de Análisis de Información del SIIPOL y su construcción como la propone RUP para el desarrollo de software, en pos de lograr una aplicación robusta y de gran calidad.

2.2 Modelo de Sistema.

2.2.1 Descripción de los Casos de Uso del Sistema.

Estructuralmente el proyecto de modernización del CICPC consta de ciclos de desarrollo bien definidos, de los cuales en el primero es donde se centrará la atención puesto que es el que está dentro del alcance del presente trabajo. En su primera iteración, el proyecto de modernización del SIIPOL en cuanto al módulo de Análisis de Información, consta de veintiún CUS a desarrollar de los cuales se determinó que para la realización de los CUS Consultar Datos de Persona en SAIME y CUS Consultar Datos de Vehículo en INTTT, no se necesita implementar las funcionalidades necesarias desde la capa de lógica de negocio del módulo de Análisis de Información, por lo que este trabajo solo abarca los restantes diecinueve CUS.

Teniendo en cuenta que el módulo Análisis de Información tiene como propósito manipular la entrada y salida de información como pueden ser datos de personas, funcionarios, armas, vehículos así como objetos en general, se partirá, como base para el desarrollo del proceso, por citar los CUS y un resumen de cada uno para lograr un mejor entendimiento de las acciones que deben realizar los mismos.

Nombre del CU	Gestionar Persona
Actor (es)	Gestor de Información (Inicia), SAIME.
Descripción	El caso de uso se inicia cuando el CU Base solicita realizar una operación sobre una Persona. En caso de crear una Persona el sistema brinda la posibilidad de introducir los datos conocidos de la Persona, permitiendo validar con SAIME la información introducida. En caso de modificar los datos de la Persona, el sistema muestra todos los datos conocidos de la misma y brinda la posibilidad de añadir nuevos datos o modificar los ya existentes, actualizando el registro en el sistema y llevando un registro histórico de los cambios ocurridos. En caso de ver los datos de la Persona, el sistema muestra todos los datos conocidos del mismo.
Referencia	1. RF: Ver datos de una Persona. 2. RF: Crear Persona. 3. RF: Modificar datos de una Persona. 4. RF: Consultar coincidencias de Personas registradas previamente en el sistema. 5. RF: Validar la integridad de los datos introducidos por el usuario. 6. RF: Mantener informado al usuario del resultado de las operaciones. 7. RF: Consultar un servicio de SAIME para obtener información de una persona. 8. RNF: En la sección: “Modificar datos de Persona” la información no es eliminada, solo es ocultada para que no sea accesible. 9. RNF: Si el servicio Web de SAIME no está activo el sistema realizará la búsqueda en la copia de la base de datos que existirá en el entorno local al sistema. 10. RNF: El sistema debe ser capaz de convertir el valor que devuelve SAIME en el dato color de la piel, tamaño, país original, estado civil, objeción, fecha de nacimiento, a la nomenclatura correspondiente en el sistema. 11. RNF: Si el usuario indica la cédula y el pasaporte la consulta en SAIME se realizará con la cédula. 12. RNF: Si se indica la fecha de nacimiento, el sistema debe mostrar la edad en los campos respectivos a la misma.

Tabla 1. Resumen CUS Gestionar Persona.

Nombre del CU	Consultar Persona
Actor (es)	Consultor (inicia).
Descripción	El caso de uso inicia cuando el Consultor accede a la opción de consultar una Persona. El sistema muestra diferentes criterios para la búsqueda de la Persona, el Consultor introduce los criterios deseados y el sistema devuelve las coincidencias encontradas. El caso de uso termina.
Referencia	1. RF: Buscar Persona dado criterios. 2. RF: Mostrar un listado de Persona ordenada por un criterio. 3. RF: Imprimir o Exportar a PDF un listado de Persona. 4. RF: Mantener informado al usuario del resultado de las operaciones. 5. RNF: Las búsquedas en el sistema de campos de tipo texto deben ser de tipo “fonéticas” (ejemplo: nombres y apellidos). Ejemplo: si se indica en el campo primer nombre Lili, el sistema debe mostrar todas las combinaciones posibles como Liliana, Lilianne, etc.

	6. RNF: Si se indica la fecha de nacimiento, el sistema debe mostrar la edad en los campos respectivos a la misma.

Tabla 2. Resumen CUS Consultar Persona.

Nombre del CU	Consultar Hoja de Vida de Persona
Actor (es)	Consultor (inicia), SIGEP, SIGEPOL.
Descripción	El caso de uso inicia cuando el Consultor solicita consultar la Hoja de Vida de la Persona. El sistema muestra todos los datos correspondientes a los antecedentes criminales. Antecedentes Penales y Antecedentes Policiales de la Persona. El sistema permite imprimir o exportar toda la información.
Referencia	1. RF: Consultar servicio de SIGEP *http://XXX*. 2. RF: Consultar servicio de SIGEPOL *http://XXX*. 3. RF: Buscar Actas Procesales de Investigación Penal relacionadas a la persona como “Imputado” o “Presunto Imputado”. 4. RF: Mostrar datos personales. 5. RF: Mostrar Antecedentes Penales. 6. RF: Mostrar Antecedentes Policiales. 7. RF: Mostrar antecedentes Criminales. 8. RF: Mantener informado al usuario del resultado de las operaciones.

Tabla 3. Resumen CUS Consultar Hoja de Vida de Persona.

Nombre del CU	Ver Relaciones de Persona
Actor (es)	Consultor (inicia).
Descripción	El caso de uso inicia cuando el consultor indica ver las relaciones de la Persona con las Actas Procesales y Notificaciones, el sistema debe mostrar las relaciones que tiene registradas en el sistema de esa Persona con las Actas Procesales, Notificaciones, Registro de Detenciones, Órdenes Legales y las relaciones Persona - Persona; además el sistema permite exportar o imprimir la información a formato PDF.
Referencia	1. RF: Ver datos de la Persona 2. RF: Ver las relaciones de la Persona con Actas Procesales, Notificaciones, Órdenes Legales y Registro de Detenciones.

Tabla 4. Resumen CUS Ver Relaciones de Persona.

Nombre del CU	Gestionar Funcionario
Actor (es)	Gestor de Información (Inicia), SIGEF IRRHH.

Descripción	El caso de uso se inicia cuando el Gestor de Información indica realizar una acción sobre un Funcionario, la misma puede ser crear un Funcionario, modificar los datos del Funcionario o ver los datos del Funcionario. Para crear un Funcionario, el sistema brinda la posibilidad de crear una Persona o modificar los datos de una Persona ya existente en el sistema y agregarle a la misma los datos propios del Funcionario. Para modificar un Funcionario, el sistema brinda la posibilidad de modificar los datos de la Persona y los datos propios del Funcionario.
Referencia	1. RF: Ver datos de un Funcionario. 2. RF: Crear un Funcionario. 3. RF: Modificar datos de un Funcionario. 4. RF: Encuestar servicio de SIGEFIRRHH para validar la información del Funcionario. 5. RF: Validar la integridad de los datos introducidos por el usuario. 6. RF: Mantener informado al usuario del resultado de las operaciones.

Tabla 5. Resumen CUS Gestionar Funcionario.

Nombre del CU	Consultar Funcionario
Actor (es)	Consultor (Inicia).
Descripción	El caso de uso inicia cuando el actor introduce los criterios por los que desea realizar la búsqueda. El sistema realiza la búsqueda y muestra un listado de coincidencias con los criterios especificados, permitiendo imprimir o exportar dicha información o realizar nuevamente la búsqueda.
Referencia	1. RF: Consultar Funcionario. 2. RF: Buscar un Funcionario dado criterios. 3. RF: Mostrar un listado de Funcionarios ordenados por un criterio. 4. RF: Ordenar un listado de Funcionarios dado un criterio. 5. RF: Imprimir la información. 6. RF: Exportar la información. 7. RF: Validar la integridad de los datos introducidos por el usuario. 8. RF: Mantener informado al usuario del resultado de las operaciones.

Tabla 6. Resumen CUS Consultar Funcionario.

Nombre del CU	Gestionar Arma
Actor (es)	CU Base (inicia), SAIME.
Descripción	El caso de uso se inicia cuando dese un CU Base se accede a la opción de realizar una operación sobre un Arma. En caso de crear un Arma el sistema brinda la posibilidad de introducir los datos conocidos del Arma. En caso de modificar los datos del Arma, el sistema muestra todos los datos conocidos de la misma y brinda la posibilidad de añadir nuevos datos o modificar los ya existentes, actualizando el registro en el sistema. En caso de ver los datos del Arma, el sistema muestra todos los datos conocidos de la misma.
Referencia	1. RF: Ver datos de un Arma. 2. RF: Crear un Arma. 3. RF: Modificar datos de un Arma. 4. RF: Ver datos de un Arma Orgánica.

	5. RF: Crear un Arma Orgánica. 6. RF: Modificar datos de un Arma Orgánica. 7. RF: Encuestar servicio de SAIME para validar la información del propietario. 8. RF: Mostrar datos del propietario validados por SAIME. 9. RF: Listar coincidencias de Armas. 10. RF: Validar la integridad de los datos introducidos por el usuario. 11. RF: Mantener informado al usuario del resultado de las operaciones. 12. RNF: En la sección 2: "Modificar datos de Arma", los datos no se eliminan, se ocultan para que no sean accesibles desde el sistema. 13. RNF: Si se modifica la condición del Arma, se debe registrar también como parte del Histórico del Arma en el campo Información lo siguiente: • Si dejó de ser Orgánica: Organismo, Cédula, Nombre y Apellidos del Funcionario que tenía asignada el Arma. • Si pasó a ser Orgánica: "Cambio de Condición". 14. RNF: Las búsquedas por serial del Arma, debe ser contra los tres seriales que puede tener un Arma, o sea que si se introduce el serial primario, debe buscarse coincidencias con Todos los seriales registrados (no importa que no sea el correspondiente al primario).

Tabla 7. Resumen CUS Gestionar Arma.

Nombre del CU	Consultar Arma
Actor (es)	Consultor (Inicia).
Descripción	El caso de uso inicia cuando el actor introduce los criterios por los que desea realizar la búsqueda. El sistema realiza la búsqueda y devuelve un listado de coincidencias con los criterios especificados, permitiendo imprimir o exportar dicha información o realizar nuevamente la búsqueda.
Referencia	1. RF: Consultar Arma 2. RF: Buscar Arma dado criterios 3. RF: Mostrar un listado de Arma ordenada por un criterio 4. RF: Ordenar un listado de Arma dado un criterio 5. RF: Imprimir o Exportar la información 6. RF: Validar la integridad de los datos introducidos por el usuario. 7. RF: Mantener informado al usuario del resultado de las operaciones. 8. RNF: Al realizar la búsqueda por Arma Orgánica y/o Arma Orgánica CICPC se mostrarán de igual manera los resultados obtenidos para Arma.

Tabla 8. Resumen CUS Consultar Arma.

Nombre del CU	Registrar Recuperación de Arma
Actor (es)	Gestor de Información (Inicia), SAIME, SIGEFIRRHH.
Descripción	El caso de uso inicia cuando el Gestor de Información indica registrar la recuperación de un Arma. El sistema guarda los datos referentes a la recuperación y que han sido introducidos por el Gestor de Información y cambia el estado del Arma a "Recuperado" o "Localizado". El sistema genera y envía una Notificación Interna a los Funcionarios que tienen asignadas las Actas Procesales con las que está relacionada el Arma indicando que ha sido recuperada o localizada. El caso de uso termina.

Referencia	1. RF: Incluir un Registro de Recuperación. 2. RF: Notificar a los Funcionarios correspondientes la recuperación del Arma. 3. RF: Modificar el estado del Arma a “Recuperado” o “Localizado”. 4. RF: Validar la cédula de la Persona en SAIME. 5. RF: Validar la credencial del funcionario en SIGEFIRRHH. 6. RF: Validar la integridad de los datos introducidos por el usuario. 7. RF: Mantener informado al usuario del resultado de las operaciones

Tabla 9. Resumen CUS Registrar Recuperación de Arma.

Nombre del CU	Ver Relaciones de Arma
Actor (es)	Consultor (Inicia).
Descripción	El caso de uso inicia cuando el consultor indica ver las relaciones del Arma con las Actas Procesales y Notificaciones con los que está relacionada. El sistema muestra las relaciones que tiene asociada el Arma. El sistema permite exportar a PDF o imprimir la información.
Referencia	1. RF: Ver las relaciones del Arma con Actas Procesales. 2. RF: Mantener informado al usuario del resultado de las operaciones.

Tabla 10. Resumen CUS Ver Relaciones de Arma.

Nombre del CU	Gestionar Vehículo
Actor (es)	CU Base (Inicia), INTTT.
Descripción	El caso de uso se inicia cuando el CU Base solicita realizar una operación sobre un Vehículo. En caso de crear un Vehículo el sistema brinda la posibilidad de introducir los datos conocidos del Vehículo, permitiendo validar con INTTT la información introducida. En caso de modificar los datos del Vehículo, el sistema muestra todos los datos conocidos del mismo y brinda la posibilidad de añadir nuevos datos o modificar los ya existentes, actualizando el registro en el sistema y llevando un registro histórico de los cambios ocurridos. En caso de ver los datos del Vehículo, el sistema muestra todos los datos conocidos del mismo.
Referencia	1. RF: Ver datos de Vehículo. 2. RF: Incluir un Vehículo. 3. RF: Modificar datos de Vehículo. 4. RF: Validar la integridad de los datos introducidos por el usuario. 5. RF: Consultar y mostrar coincidencias de Vehículos registrados en el sistema. 6. RF: Mantener informado al usuario del resultado de las operaciones. 7. RF: Consultar servicio de INTTT. 8. RNF: Si el servicio Web de INTTT no está activo el sistema realizará la búsqueda en la copia de la base de datos que existirá en el entorno local al sistema. 9. RNF: Si se especifica por el usuario el número de la placa y el número del serial de carrocería, el sistema debe consultar en INTTT con el serial de carrocería.

Tabla 11. Resumen CUS Gestionar Vehículo.

Nombre del CU	Consultar Vehículo
Actor (es)	Consultor (Inicia).

Descripción	El caso de uso inicia cuando el actor accede a la opción de consultar un Vehículo. El sistema muestra diferentes criterios para la búsqueda del Vehículo, el Consultor introduce los criterios deseados y el sistema devuelve las coincidencias encontradas. El caso de uso termina.
Referencia	1. RF: Consultar Vehículo. 2. RF: Buscar Vehículo dado criterios. 3. RF: Mostrar un listado de Vehículo ordenado por un criterio. 4. RF: Ordenar un listado de Vehículo dado un criterio. 5. RF: Validar la integridad de los datos introducidos por el usuario. 6. RF: Mantener informado al usuario del resultado de las operaciones. 7. RNF: La búsqueda por defecto deberé ser sobre Ambos, en caso de no especificarse si la consulta se desea realizar sobre Motos, o sobre Otros Vehículos o sobre Otros vehículos y Motos.

Tabla 12. Resumen CUS Consultar Vehículo.

Nombre del CU	Registrar Recuperación de Vehículo
Actor (es)	Gestor de Información (Inicia), SAIME, SIGEFIRRHH.
Descripción	El caso de uso inicia cuando el Gestor de Información indica registrar la recuperación de un Vehículo. El sistema guarda los datos referentes a la recuperación y que han sido introducidos por el Gestor de Información y cambia el estado del Vehículo a “Recuperado” o “Localizado”. El sistema genera y envía una Notificación Interna a los Funcionarios que tienen asignadas las Actas Procesales con las que está relacionado el Vehículo indicando que ha sido recuperado o localizado. El caso de uso termina.
Referencia	1. RF: Incluir un Registro de Recuperación. 2. RF: Notificar a los Funcionarios correspondientes la recuperación del Vehículo. 3. RF: Modificar el estado del Vehículo a “Recuperado” o “Localizado”. 4. RF: Validar la cédula de la Persona en SAIME. 5. RF: Validar la credencial del funcionario en SIGEFIRRHH. 6. RF: Validar la integridad de los datos introducidos por el usuario. 7. RF: Mantener informado al usuario del resultado de las operaciones

Tabla 13. Resumen CUS Registrar Recuperación de Vehículo.

Nombre del CU	Validar Serial de Carrocería
Actor (es)	Consultor (Inicia).
Descripción	El caso de uso inicia cuando el consultor accede a la opción de validar la estructura del serial de carrocería.
Referencia	1. RF: Validar estructura de serial de carrocería. 2. RF: Validar la integridad de los datos introducidos por el usuario. 3. RF: Mantener informado al usuario del resultado de las operaciones.

Tabla 14. Resumen CUS Validar Serial de Carrocería.

Nombre del CU	Ver Relaciones de Vehículo
Actor (es)	Consultor (Inicia).

Descripción	El caso de uso inicia cuando el consultor indica ver las relaciones del Vehículo con las Actas Procesales y Notificaciones con las que está relacionado. El sistema muestra las relaciones que tiene asociada el Vehículo con las Actas Procesales y las Notificaciones registradas en el sistema. El sistema permite exportar o imprimir la información.
Referencia	1. RF: Ver las relaciones del Vehículo con Notificaciones y Actas Procesales. 2. RF: Mantener informado al usuario del resultado de las operaciones.

Tabla 15. Resumen CUS Ver Relaciones de Vehículo.

Nombre del CU	Gestionar Objeto
Actor (es)	Gestor de Información (Inicia).
Descripción	El caso de uso se inicia cuando el Gestor de Información accede a la opción de realizar una operación sobre un Objeto. En caso de crear un Objeto el sistema brinda la posibilidad de introducir los datos conocidos del Objeto. En caso de modificar los datos del Objeto, el sistema muestra todos los datos conocidos del mismo y brinda la posibilidad de añadir nuevos datos o modificar los ya existentes, actualizando el registro en el sistema. En caso de ver los datos del Objeto, el sistema muestra todos los datos del Objeto.
Referencia	1. RF: Ver datos de Objeto. 2. RF: Crear Objeto. 3. RF: Modificar datos de Objeto. 4. RF: Imprimir o exportar a PDF los datos del Objeto. 5. RF: Validar la integridad de los datos introducidos por el usuario. 6. RF: Mantener informado al usuario del resultado de las operaciones. 7. RNF: En la sección "Modificar Datos de Objetos", los datos eliminados de los listados no se eliminan del sistema, solo se ocultan. 8. RNF: En la sección "Ver Datos de Objetos" la acción Registrar Recuperación, debe estar habilitada solo si el Objeto tiene estado "Solicitado".

Tabla 16. Resumen CUS Gestionar Objeto.

Nombre del CU	Consultar Objeto
Actor (es)	Consultor (Inicia).
Descripción	El caso de uso inicia cuando el actor selecciona la opción de consultar los datos de Objetos. El Consultor introduce uno o varios criterios para realizar la búsqueda. El sistema realiza la búsqueda con los criterios indicados y muestra un listado con el resumen de los datos de los Objetos coincidentes, permitiendo imprimir o exportar el resumen de la información.
Referencia	1. RF: Consultar Objeto. 2. RF: Buscar Objeto dado criterios. 3. RF: Mostrar un listado de Objeto ordenado por un criterio. 4. RF: Ordenar un listado de Objeto dado un criterio. 5. RF: Validar la integridad de los datos introducidos por el usuario. 6. RF: Mantener informado al usuario del resultado de las operaciones.

Tabla 17. Resumen CUS Consultar Objeto.

Nombre del CU	Registrar Recuperación de Objeto
Actor (es)	Gestor de Información (Inicia), SAIME, SIGEFIRRHH.
Descripción	El caso de uso inicia cuando el Gestor de Información indica registrar la recuperación de un Objeto. El sistema guarda los datos referentes a la recuperación y que han sido introducidos por el Gestor de Información y cambia el estado del Objeto a “Recuperado” o “Localizado”. El sistema genera y envía una Notificación Interna a los Funcionarios que tienen asignadas las Actas Procesales con las que está relacionado el Objeto indicando que ha sido recuperado o localizado. El caso de uso termina.
Referencia	1. RF: Incluir un Registro de Recuperación. 2. RF: Notificar a los Funcionarios correspondientes la recuperación del Objeto. 3. RF: Modificar el estado del Objeto a “Recuperado” o “Localizado”. 4. RF: Validar la cédula de la Persona en SAIME. 5. RF: Validar la credencial del funcionario en SIGEFIRRHH. 6. RF: Validar la integridad de los datos introducidos por el usuario. 7. RF: Mantener informado al usuario del resultado de las operaciones.

Tabla 18. Resumen CUS Registrar Recuperación de Objeto.

Nombre del CU	Ver Relaciones de Objeto
Actor (es)	Consultor (Inicia).
Descripción	El caso de uso inicia cuando el consultor indica ver las relaciones de un Objeto con las Actas Procesales y Notificaciones, el sistema debe mostrar las relaciones que tiene registradas en el sistema de ese Objeto con las Actas Procesales y las Notificaciones; además el sistema permite exportar o imprimir la información a formato PDF.
Referencia	1. RF: Ver las relaciones de la Persona con Notificaciones y Actas Procesales 2. RF: Mantener informado al usuario del resultado de las operaciones.

Tabla 19. Resumen CUS Ver Relaciones de Objeto.

2.3 Modelo de Análisis.

2.3.1 Definición del Modelo de Análisis.

El Modelo de análisis contiene clases de análisis y sus objetos organizados en paquetes que colaboran. “Durante el análisis, analizamos los requisitos que se describen en la captura de requerimientos, refinándolos y estructurándolos. El objetivo de hacerlo es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que nos ayude a estructurar el sistema entero, incluyendo su arquitectura” (10).

Las clases de análisis se centran en los requisitos funcionales y son evidentes en el dominio del problema porque representan conceptos y relaciones del dominio. Tienen atributos y entre ellas se establecen relaciones de asociación, agregación / composición, generalización especialización y tipos

asociativos. En la construcción del modelo de análisis se tienen que identificar las clases que describen la realización de los casos de uso, los atributos y las relaciones entre ellas. Con esta información se construye el Diagrama de clases del análisis, que por lo general se descompone para agrupar las clases en paquetes. Esta descomposición tiene impacto por lo general en el diseño e implementación de la solución.

RUP propone clasificar las clases en:

- *Clases de Interfaz*: Modelan la interacción entre el sistema y sus actores.
- *Clases de control*: Coordinan la realización de uno o unos pocos casos de uso coordinando las actividades de los objetos que implementan la funcionalidad del caso de uso.
- *Clases entidad*: Modelan información que posee larga vida y que es a menudo persistente.

2.3.2 Diagramas de Clases de Análisis.

El objetivo principal del flujo de trabajo del análisis son los diagramas de clases de análisis, los cuales muestran qué clases participan en las realizaciones de los distintos casos de usos. Para cada CUS se realizó un diagrama de clases del análisis, a continuación se presentan los vinculados a la realización del sub-módulo Persona. Los restantes diagramas son parte de los anexos. (Ver Anexo 1)

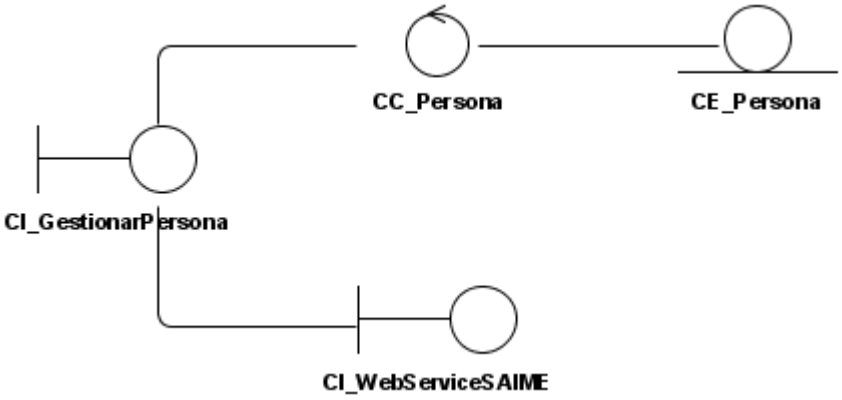

Fig 2. CUS Gestionar Persona.

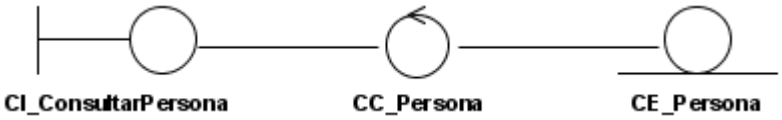

Fig 3. CUS Consultar Persona.

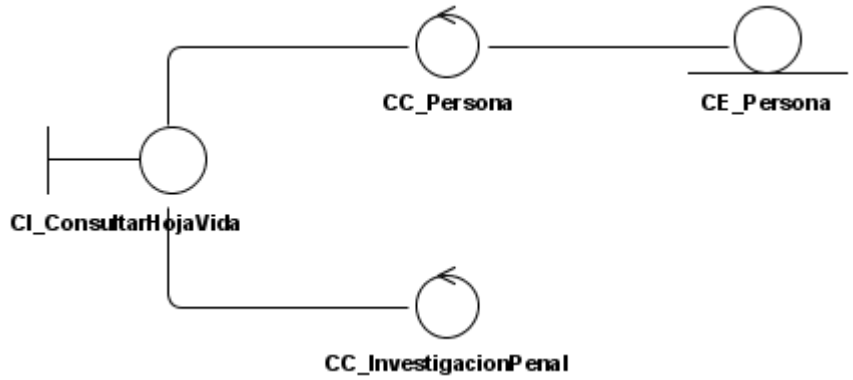

Fig 4. CUS Consultar Hoja de Vida de Persona.

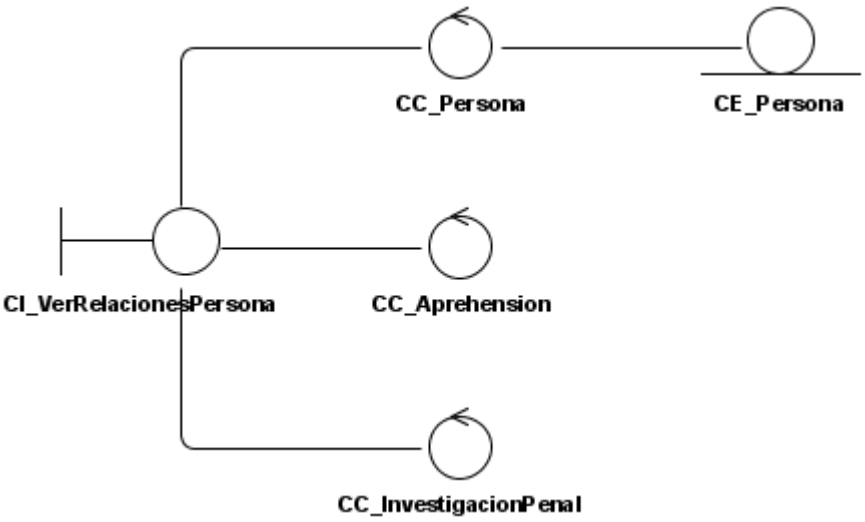

Fig 5. CUS Ver Relaciones de Persona.

2.4 Modelo de Diseño.

2.4.1 Definición del Modelo de Diseño.

En el flujo de trabajo de diseño, se modela el sistema de manera que soporte todos los requerimientos, incluyendo a los requerimientos no funcionales. Este modelo se puede utilizar para visualizar la implementación y para soportar las técnicas de programación gráfica de la aplicación.

“El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Además, el modelo de diseño sirve de abstracción de la implementación del sistema y es, de ese modo, utilizada como una entrada fundamental de las actividades de implementación” (10).

2.4.2 Arquitectura del Módulo Análisis de Información.

El módulo de Análisis de Información del SIIPOL cuenta con una estructura de paquetes como se muestra en la figura 6, donde se agrupan sus diferentes sub-módulos, **Arma**, **Funcionario**, **Objeto**, **Persona** y **Vehículo**. Se puede apreciar la estructura interna de paquetes del sub-módulo Arma, la cual comparten los restantes sub-módulos, al igual que el paquete **común.** Éste último se encarga de agrupar todas las funcionalidades que por alguna razón resultan comunes entre algunos o el resto de los sub-módulos citados anteriormente.

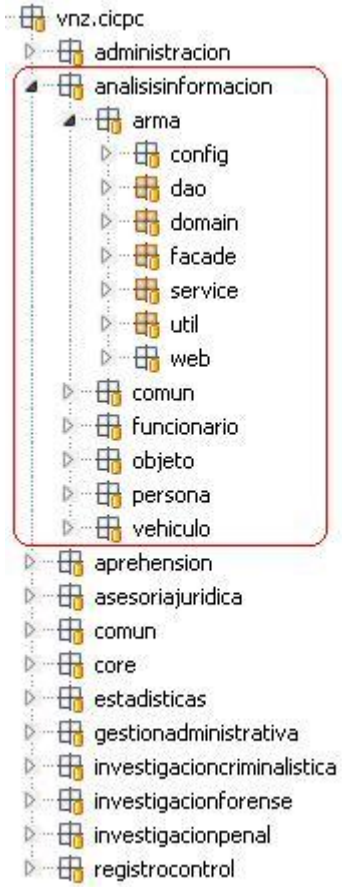

Fig 6. Estructura de paquetes del módulo Análisis de Información del SIIPOL.

2.4.3 El Manejo de Nomencladores.

El trabajo con nomencladores es una funcionalidad muy importante que desde sus inicios se decidió llevar a cabo en el desarrollo de la aplicación, así sería posible acotar el dominio de datos disponibles para cada entidad de información lo que permite eliminar redundancias en la base de datos, haciendo más objetivo y eficiente el manejo de datos. Cada módulo del SIIPOL trata de forma independiente sus nomencladores, en el caso de la capa de lógica de negocio del módulo de Análisis de Información, este tratamiento consiste fundamentalmente en brindar las listas de valores posibles de cada nomenclador en cuanto la capa de presentación las solicite. En los diferentes sub-módulos de Análisis de Información existen los nomencladores como características de las entidades persistentes, por lo que se pueden agrupar en un mismo servicio todo el trabajo con los nomencladores del módulo y ubicar el mismo en el paquete **común**. El diseño necesario para proveer a la capa de presentación de los servicios referentes a los nomencladores, no son parte de las realizaciones de los CUS que estudia la solución, pero aun así se decidió abarcarlos pues forman parte de la lógica de negocio del módulo Análisis de Información del SIIPOL. Para una mejor comprensión del diseño del manejo de los nomencladores del módulo dirigirse a los Anexos. (Ver Anexo 6)

2.4.4 Diagramas de Clases de Diseño.

A través del flujo de diseño, uno de los artefactos más importantes a obtener son los diagramas de clases de diseño, donde se exponen las clases que intervienen en las realizaciones de los casos de uso del sistema. En este tipo de diagrama se representa un nivel de detalle más alto que los diagramas de clases del análisis, relacionándose con el lenguaje de programación del cual se hará uso en la implementación del sistema. Además se especifican los parámetros entrantes en las operaciones así como sus tipos de retorno, dejándole una clara visión arquitectónica al programador de cómo debe ser estructurado el código a implementar.

El modelo de diagrama de clases a utilizar modela una propuesta con los siguientes tipos de clases, clasificados según su objetivo:

- Clases de Fachada.
- Clases de Servicios.
- Clases de Acceso a Datos.
- Entidades.

Las clases fachada tienen como tarea principal la de manejar transacciones y conexiones e implementar la seguridad hacia adentro de la capa de negocio, estas clases están desacopladas de su implementación usando los mecanismos que brinda Spring Framework, a través de interfaces y archivos XML de mapeo, son además una implementación del patrón Fachada**.**

Los servicios encapsulan la funcionalidad que tiene que ver con la lógica de negocio, y las clases de acceso a datos le brindan un medio para acceder a los datos almacenados a un nivel de abstracción alto, utilizando para ello las instancias de las clases entidades.

De los diagramas de clases del diseño generados como solución a los CUS propuestos se citan a continuación los relacionados al sub-módulo Persona, el resto es parte de los anexos. (Ver Anexo 2)

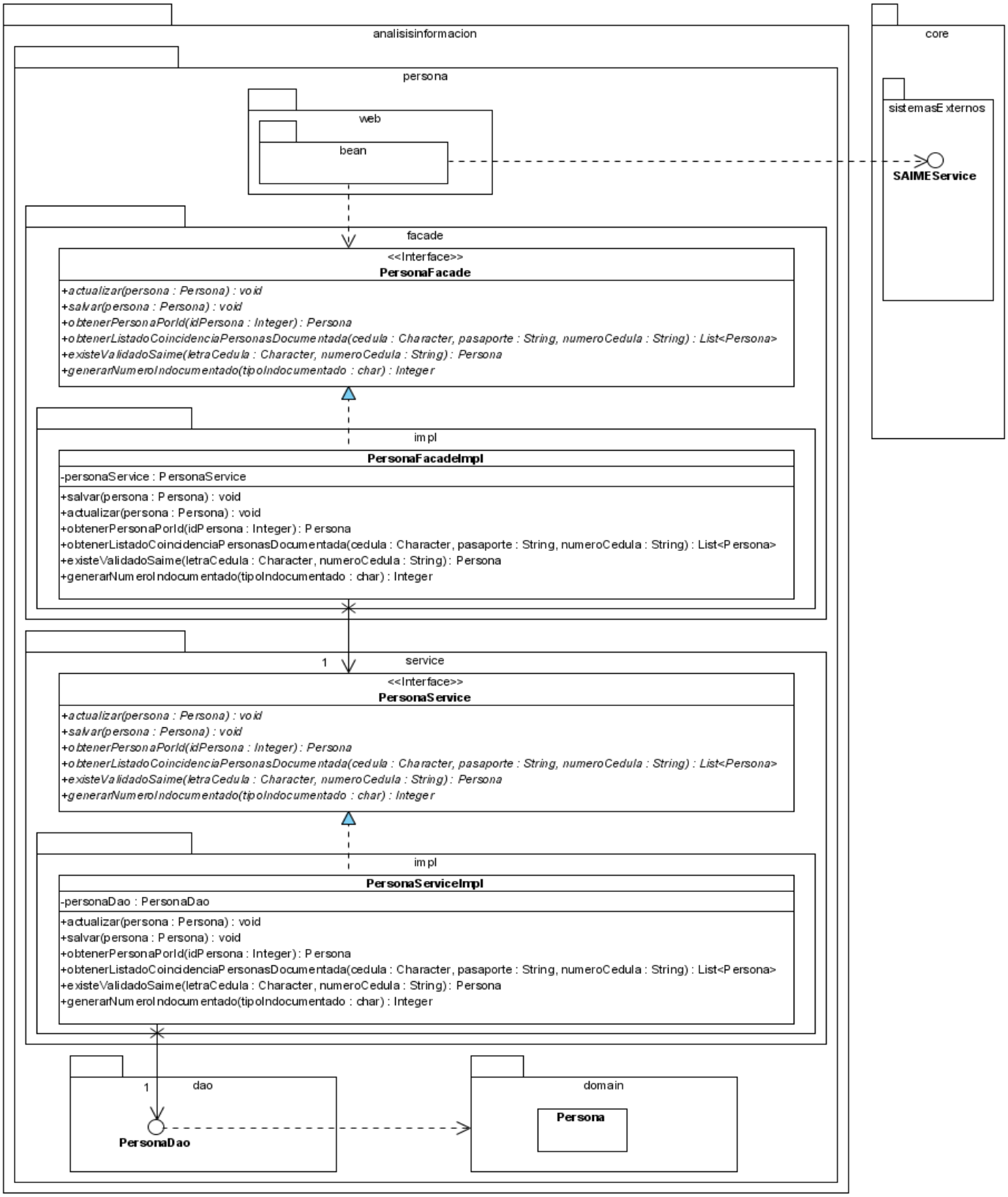

Fig 7. CUS Gestionar Persona.

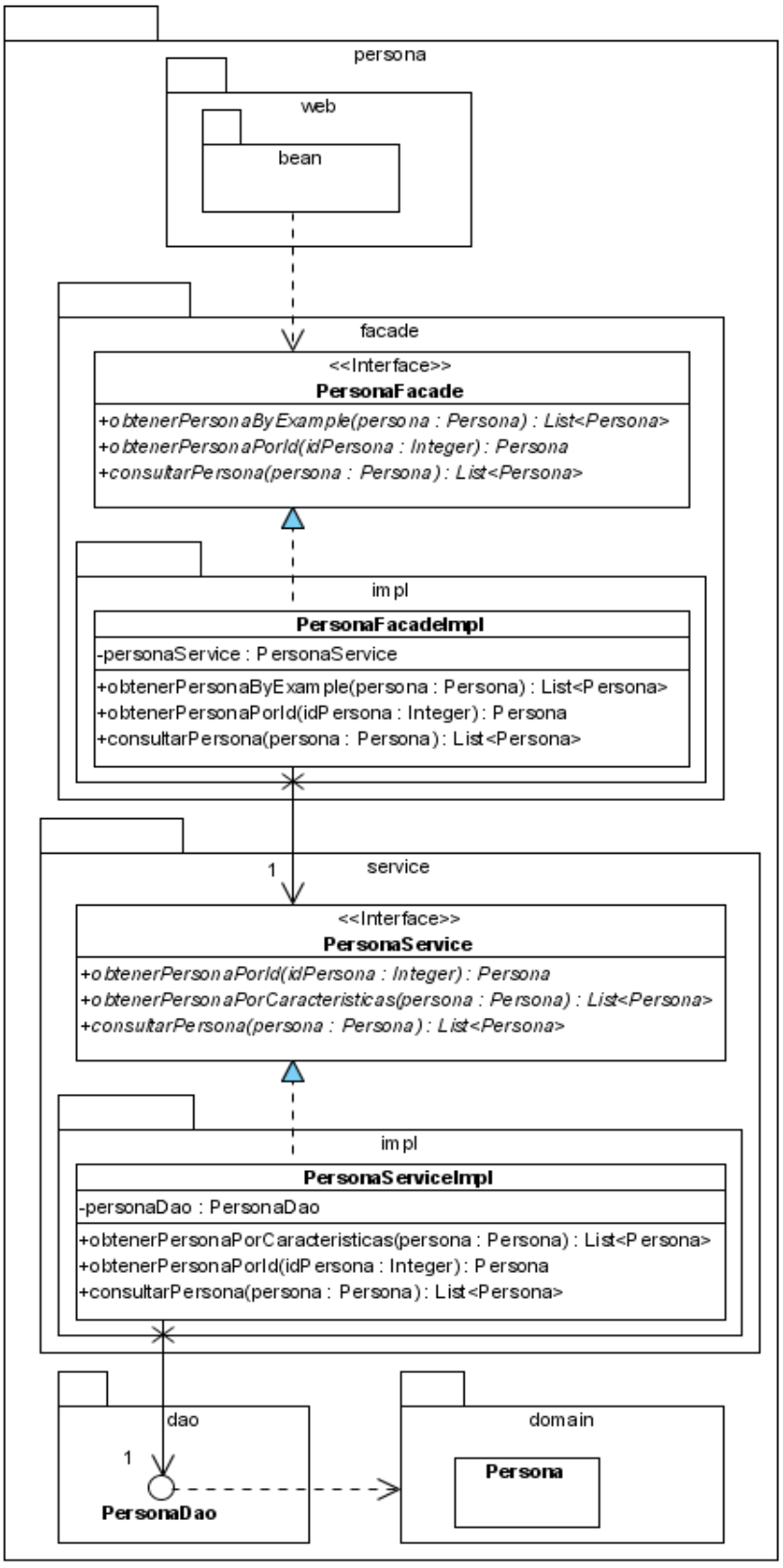

Fig 8. CUS Consultar Persona.

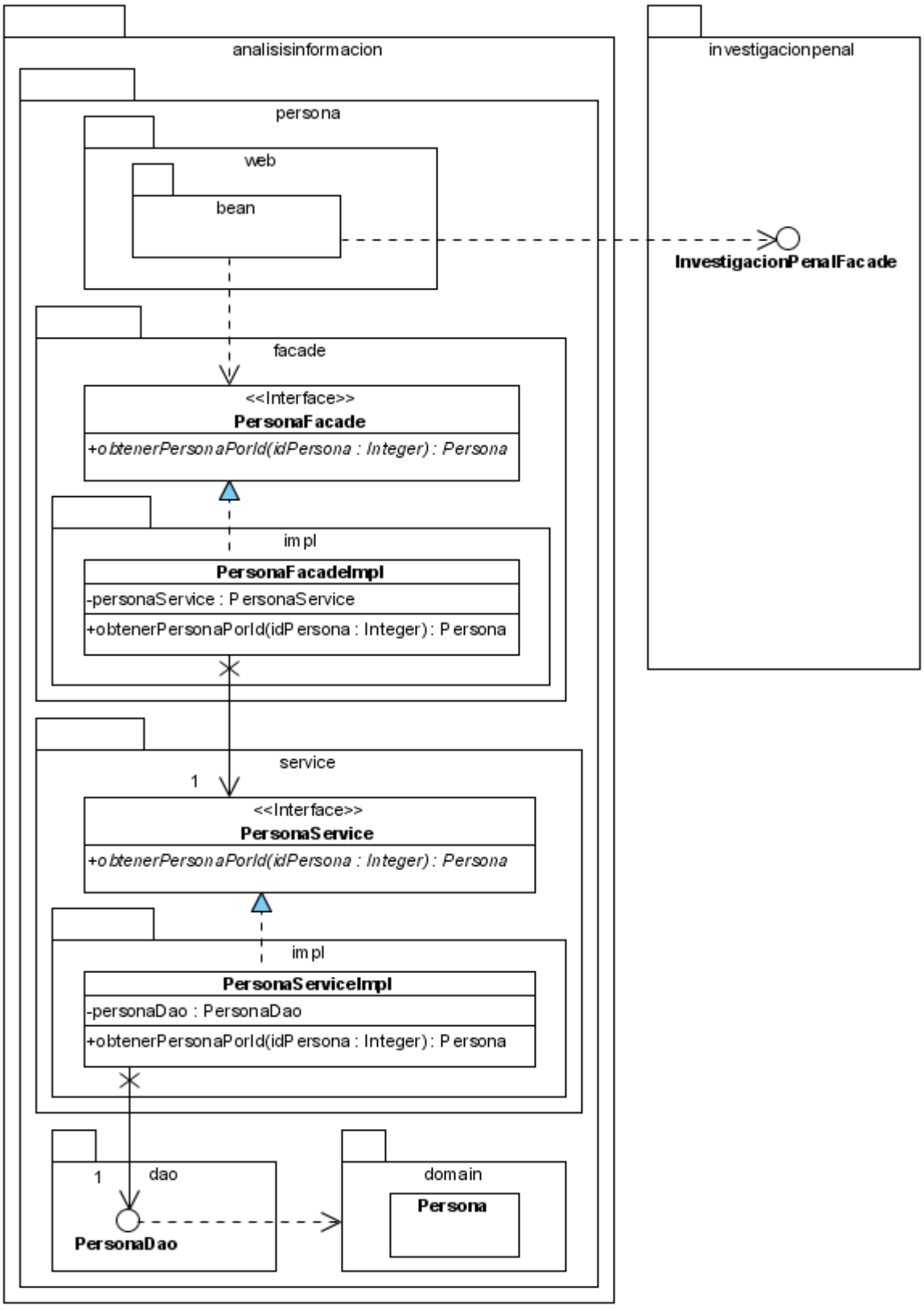

Fig 9. CUS Consultar Hoja de Vida de Persona.

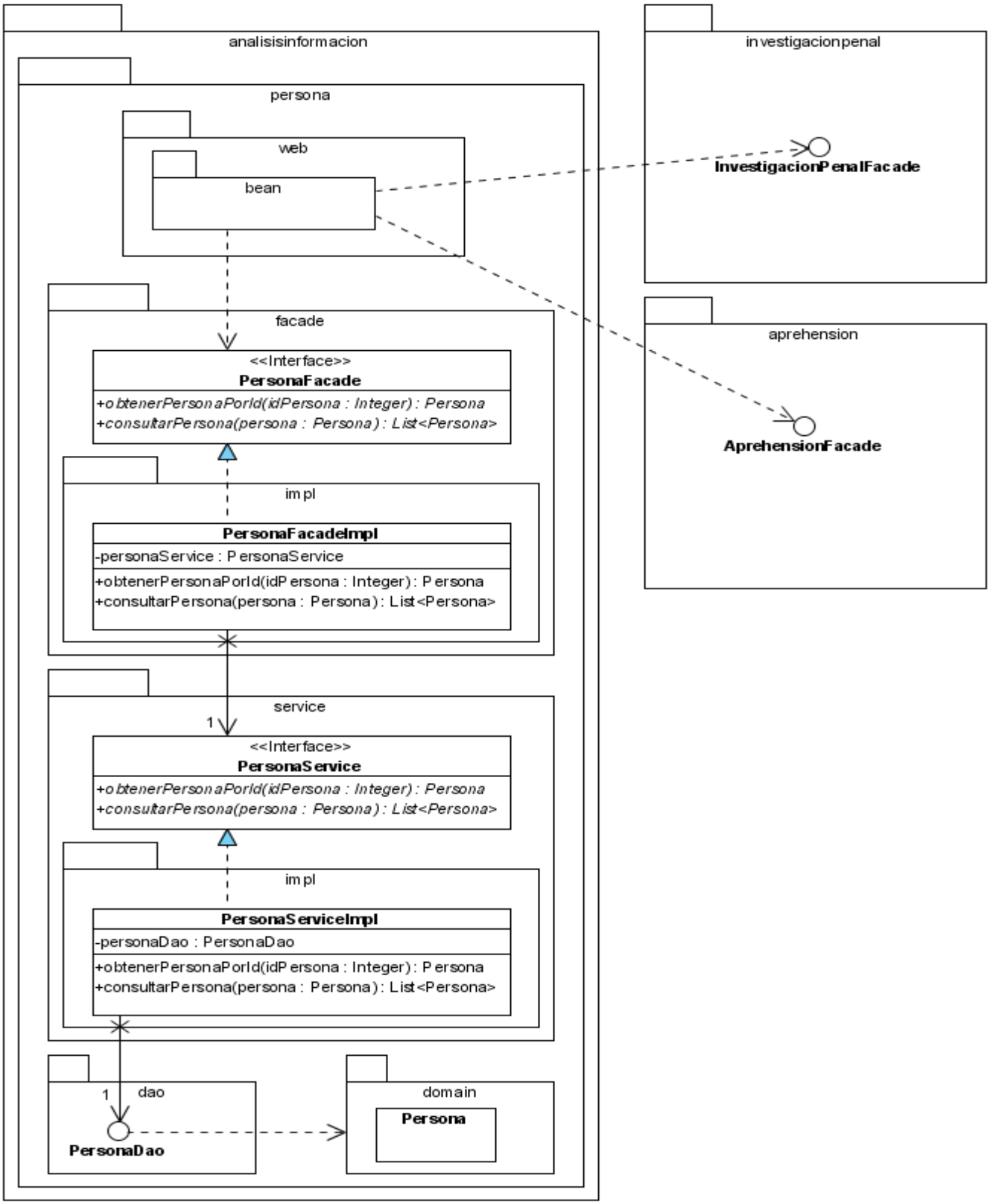

Fig 10. CUS Ver Relaciones de Persona.

2.4.5 Diagramas de Contrato entre Paquetes.

Un diagrama de contrato entre paquetes consiste en un diagrama de interacción de alto nivel que muestra el paso de mensajes entre las diferentes carpetas, sin mostrar todo el flujo de eventos que ocurre dentro de las mismas cuando llega una determinada petición. Su objetivo es brindar a los programadores, sin incurrir en demasiado nivel de detalle, lo necesario para la realización de los casos de uso. Este artefacto ahorra tiempo de diseño, ya que su realización no requiere demasiadas especificaciones, y simplifica de manera considerable los diagramas necesarios para las realizaciones. Se generó por cada CUS al menos un diagrama de contrato, aunque en algunos CUS fue necesario utilizar más de un diagrama debido a una mayor complejidad, citados posteriormente se pueden apreciar los del sub-módulo Persona, los demás forman parte de los anexos. (Ver Anexo 3)

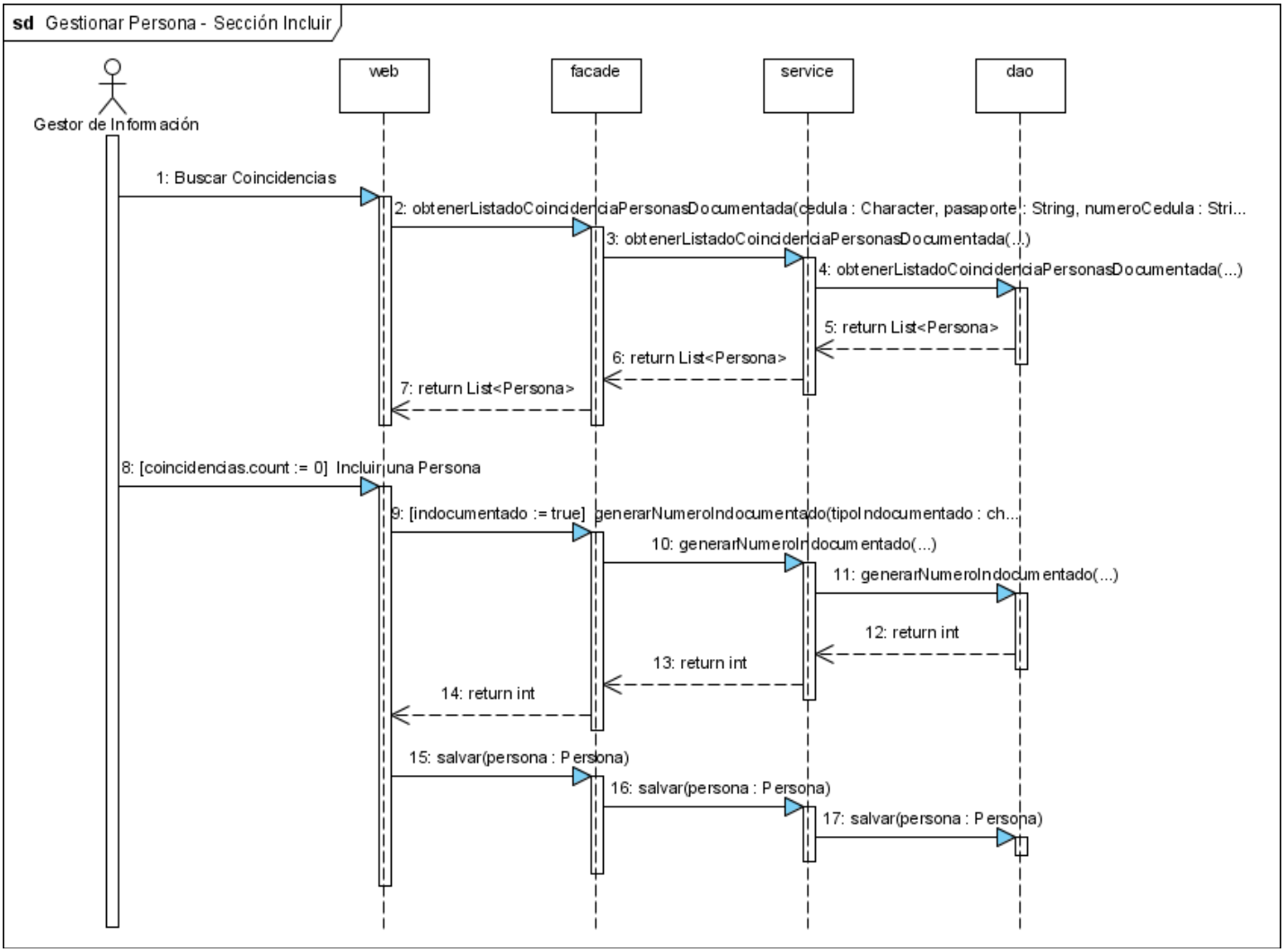

Fig 11. CUS Gestionar Persona – Sección Incluir.

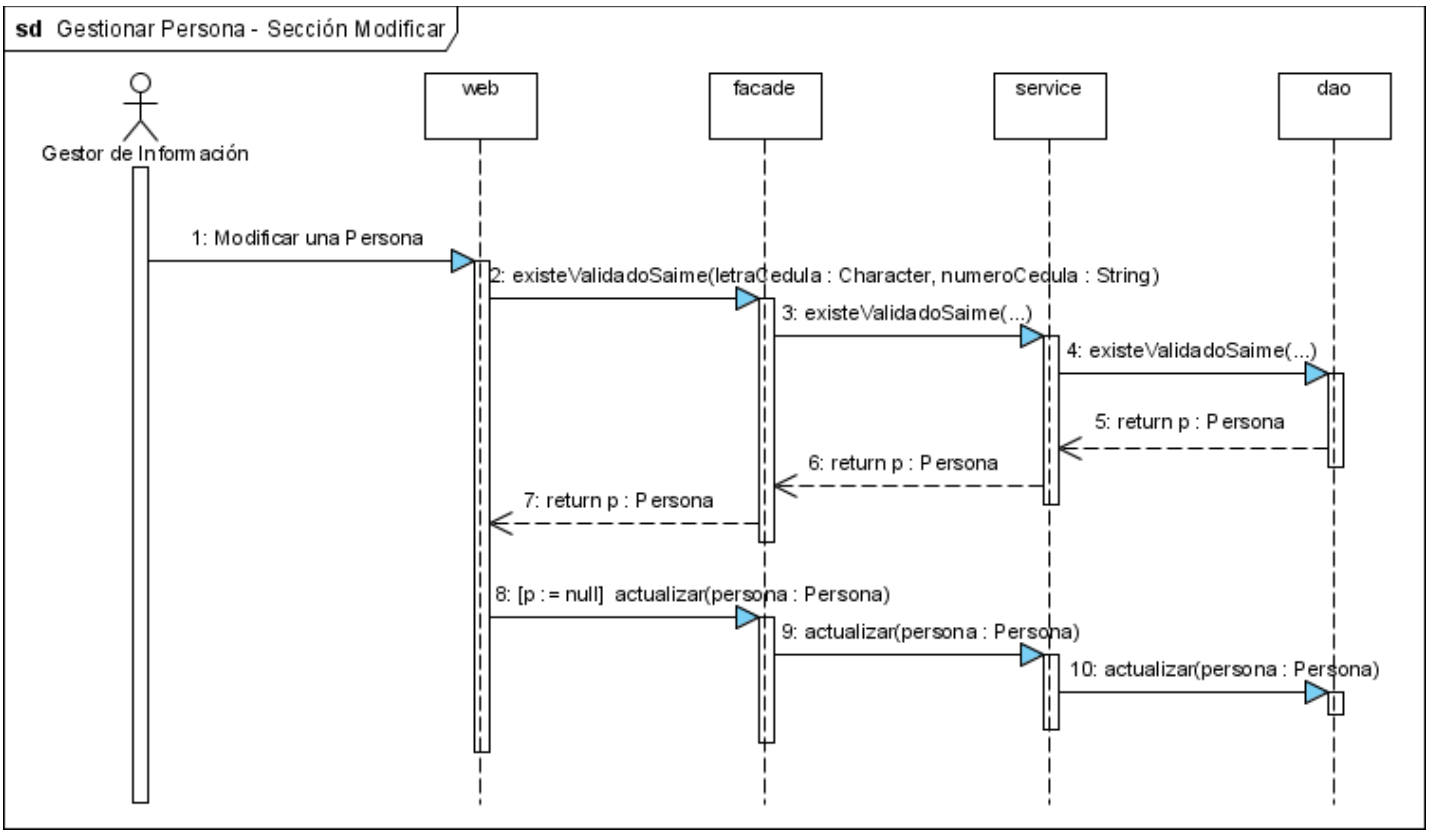

Fig 12. CUS Gestionar Persona – Sección Modificar.

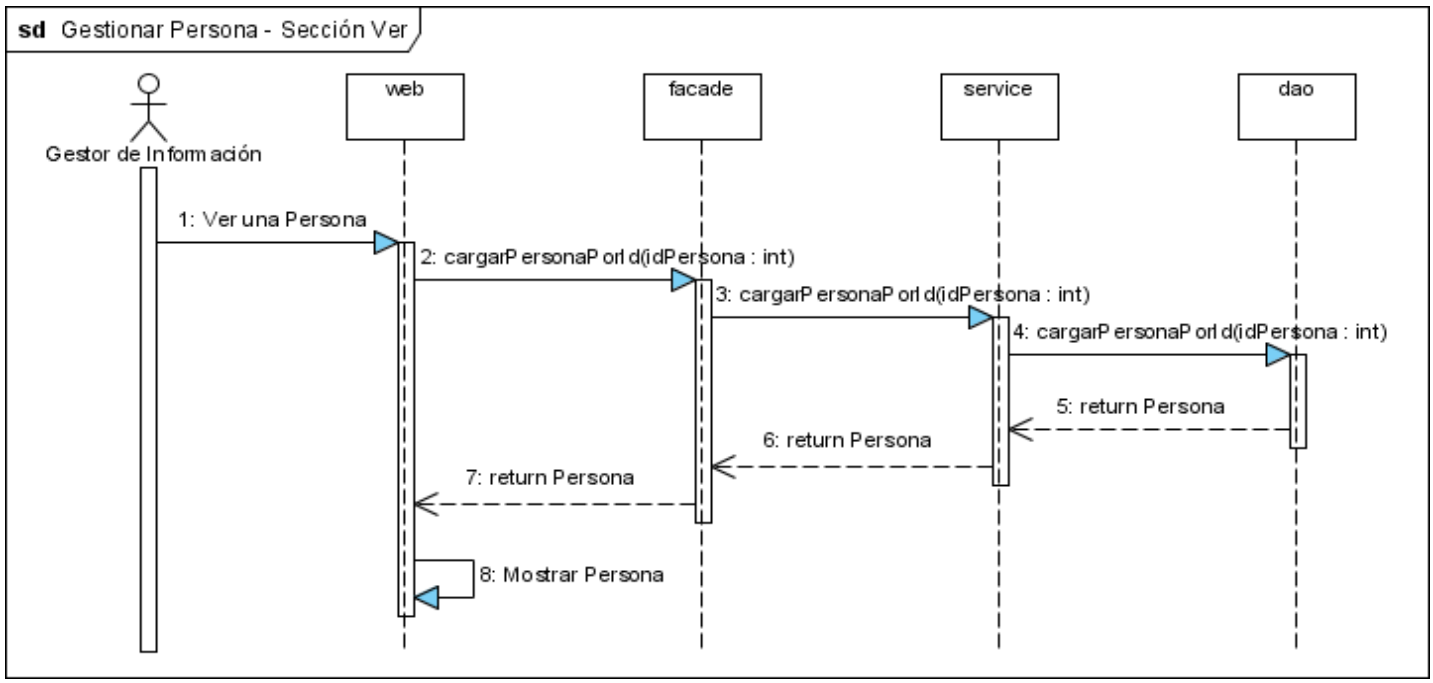

Fig 13. CUS Gestionar Persona – Sección Ver.

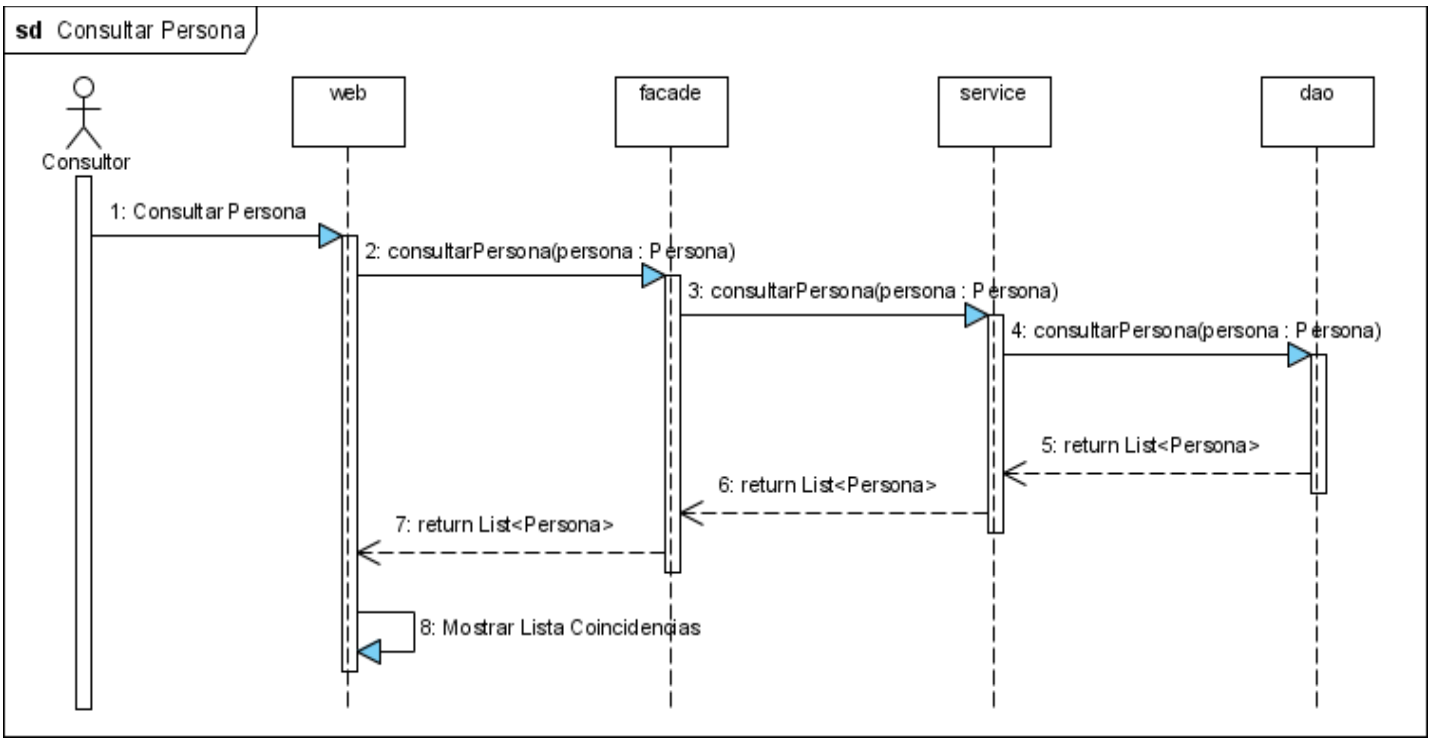

Fig 14. CUS Consultar Persona.

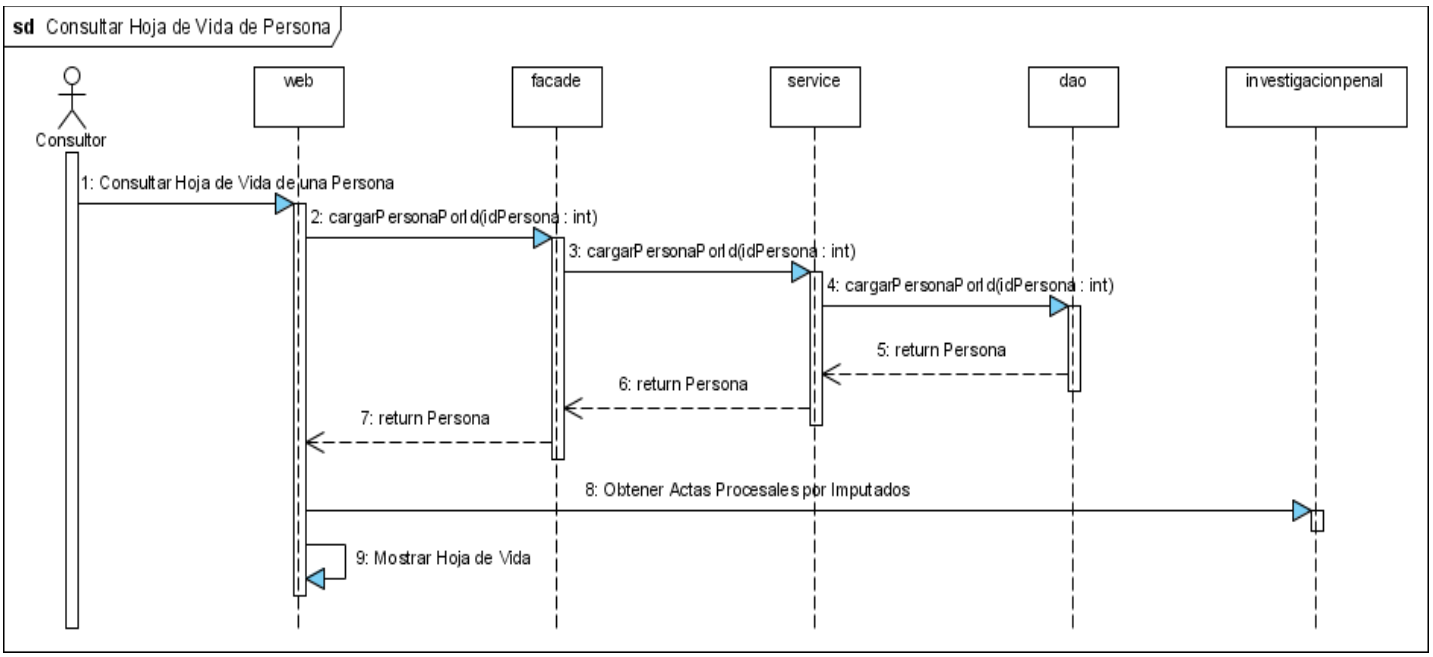

Fig 15. CUS Consultar Hoja de Vida de Persona.

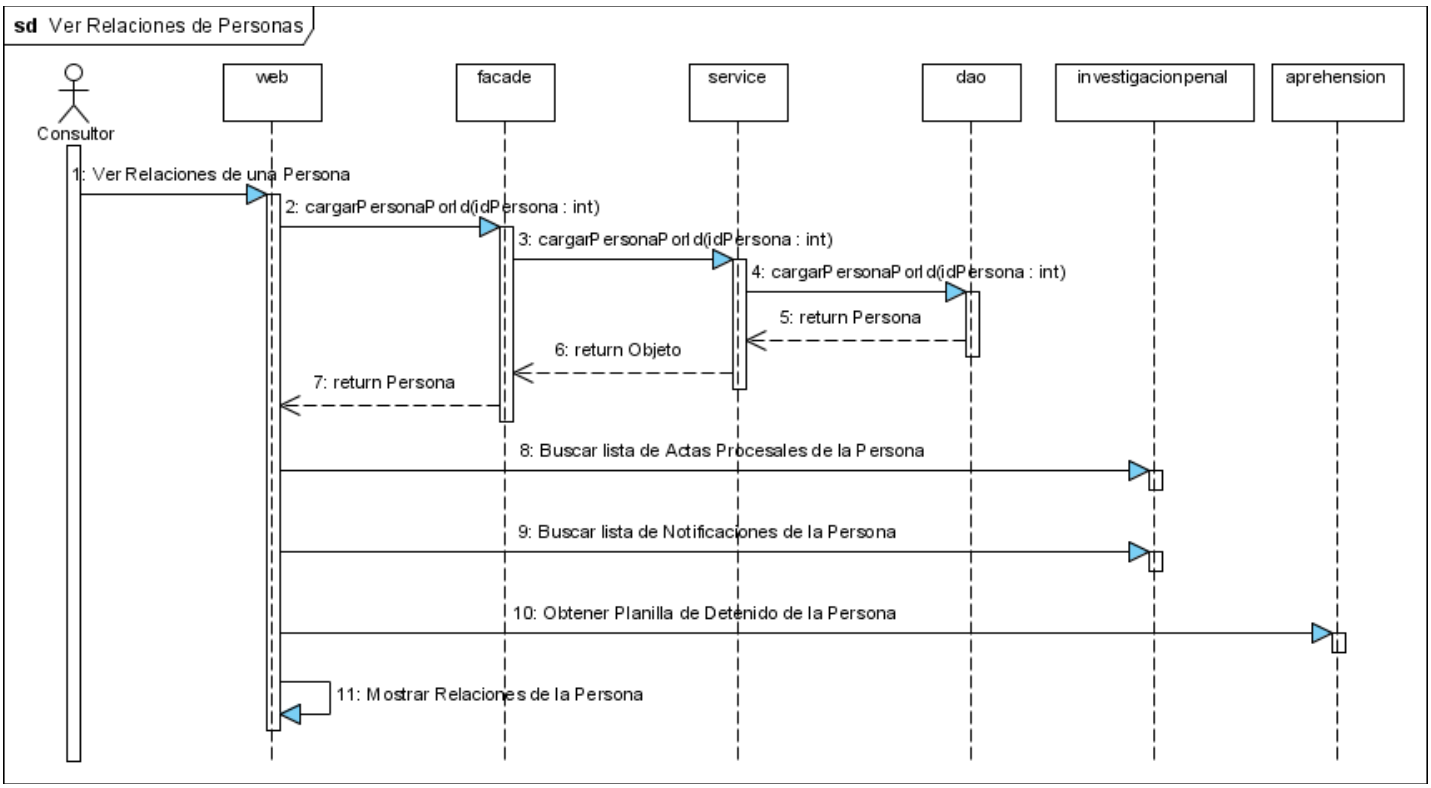

Fig 16. CUS Ver Relaciones de Persona.

2.4.6 Descripción de las Clases del Diseño.

Las descripciones de las clases del diseño ayudan a comprender mejor las responsabilidades de las mismas y al mismo tiempo tener una visión más exacta del diseño del sistema. Cada clase descrita posteriormente, implementa una interfaz como resultado de aplicar el patrón Fachada. Para estas interfaces no se generaron descripciones pues coinciden con las clases que las implementan en cuanto a las responsabilidades. A continuación se muestra la descripción de las clases del diseño del sub-módulo Persona, las demás descripciones pueden consultarse en los anexos. (Ver Anexo 4)

Nombre de la Clase:	PersonaFacadeImpl
Tipo de Clase:	Controladora
Atributo:	**Tipo:**
personaService	PersonaService
Para Cada Responsabilidad:	
Nombre:	void salvar(Persona persona)
Descripción:	*Esta función se utiliza para guardar una Persona*
Nombre:	void actualizar(Persona persona)
Descripción:	*Esta función se utiliza para actualizar un Persona*
Nombre:	Persona obtenerPersonaPorId(Integer idPersona)

Descripción:	*Esta función se utiliza para cargar una persona por id*
Nombre:	List<Persona> obtenerListadoCoincidenciaPersonasDocumentada(Character cedula, String pasaporte, String numeroCedula)
Descripción:	*Esta función se utiliza para obtener el listado de coincidencias de las personas documentadas que cumplen con los parámetros que se le pasan*
Nombre:	List<Persona> consultarPersona(Persona persona)
Descripción:	*Esta función se utiliza para obtener todas las personas que cumplan con las características contenidas en el objeto pasado como parámetro.*
Nombre:	Integer generarNumeroIndocumentado(char tipoIndocumentado)
Descripción:	*Esta función se utiliza para generar los números de indocumentados según los tipos*
Nombre:	Persona existeValidadoSaime(Character letraCedula, String numeroCedula)
Descripción:	*Esta función se utiliza para obtener la persona si existe en la base datos validada por SAIME*

Tabla 20. Descripción de la Clase PersonaFacadeImpl.

Nombre de la Clase:	PersonaServiceImpl
Tipo de Clase:	Controladora
Atributo:	**Tipo:**
personaDao	PersonaDao
carga	Carga
Para Cada Responsabilidad:	
Nombre:	void salvar(Persona persona)
Descripción:	*Esta función se utiliza para guardar una Persona*
Nombre:	void actualizar(Persona persona)
Descripción:	*Esta función se utiliza para actualizar un Persona*
Nombre:	Persona obtenerPersonaPorId(Integer idPersona)
Descripción:	*Esta función se utiliza para cargar una persona por id*
Nombre:	List<Persona> obtenerListadoCoincidenciaPersonasDocumentada(Character cedula, String pasaporte, String numeroCedula)
Descripción:	*Esta función se utiliza para obtener el listado de coincidencias de las personas documentadas que cumplen con los parámetros que se le pasan*
Nombre:	List<Persona> consultarPersona(Persona persona)
Descripción:	*Esta función se utiliza para obtener todas las personas que cumplan con las características contenidas en el objeto pasado como parámetro.*
Nombre:	Integer generarNumeroIndocumentado(char tipoIndocumentado)
Descripción:	*Esta función se utiliza para generar los números de indocumentados según los tipos*
Nombre:	Persona existeValidadoSaime(Character letraCedula, String numeroCedula)
Descripción:	*Esta función se utiliza para obtener la persona si existe en la base datos validada por SAIME*

Tabla 21. Descripción de la Clase PersonaServiceImpl.

2.5 Conclusión.

En este capítulo se han expuesto los resultados más relevantes del análisis y el diseño para la capa de lógica de negocio del módulo Análisis de Información del SIIPOL a partir de los CUS asociados. Se hizo la descripción de las clases necesarias para llegar a la implementación, partiendo de artefactos obtenidos como los diagramas de clases del diseño y los diagramas de contrato entre paquetes. Se estudió el manejo de los nomencladores, así como la función del paquete común dentro del módulo. Además se expuso la arquitectura de la aplicación, permitiendo de esta forma que cualquier desarrollador tenga una idea de cómo debe ser implementada la lógica de negocio del módulo.

CAPÍTULO 3: IMPLEMENTACIÓN

3.1 Introducción.

El flujo de trabajo implementación comienza con el resultado del diseño y se realiza el sistema en términos de componentes y cómo estos se organizan de acuerdo a los nodos específicos en el modelo de despliegue. Los diagramas de despliegue y componentes conforman lo que se conoce como un modelo de implementación ya que describen los componentes a construir, su organización y dependencia entre nodos físicos en los que funcionará la aplicación.

3.2 Modelo de Implementación.

3.2.1 Modelo de Despliegue.

El Modelo de Despliegue describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo y dónde el sistema será puesto en funcionamiento. Las estaciones de trabajo, dispositivos y procesadores son reflejados como nodos y su estructura interna puede ser representada adicionando otros nodos o artefactos. El modelo de despliegue se utiliza como entrada fundamental en las actividades de diseño e implementación debido a que la distribución del sistema tiene una influencia principal en su diseño.

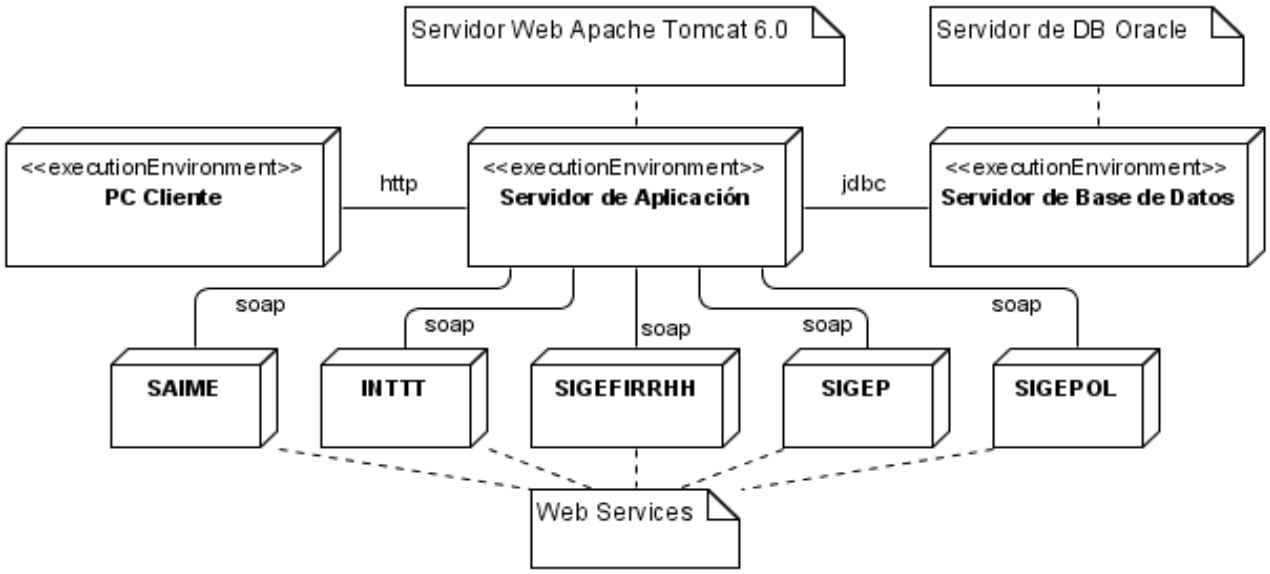

Fig 17. Diagrama de Despliegue.

3.2.2 Diagramas de Componentes.

El Diagrama de Componentes define cómo las clases, artefactos y otros elementos de bajo nivel, se unen para formar componentes de alto nivel y las conexiones entre ellos. Los componentes son artefactos de software compilados que trabajan acoplados para brindar el comportamiento requerido dentro de las restricciones definidas en el proceso de captura de requisitos. (Ver Anexo 5)

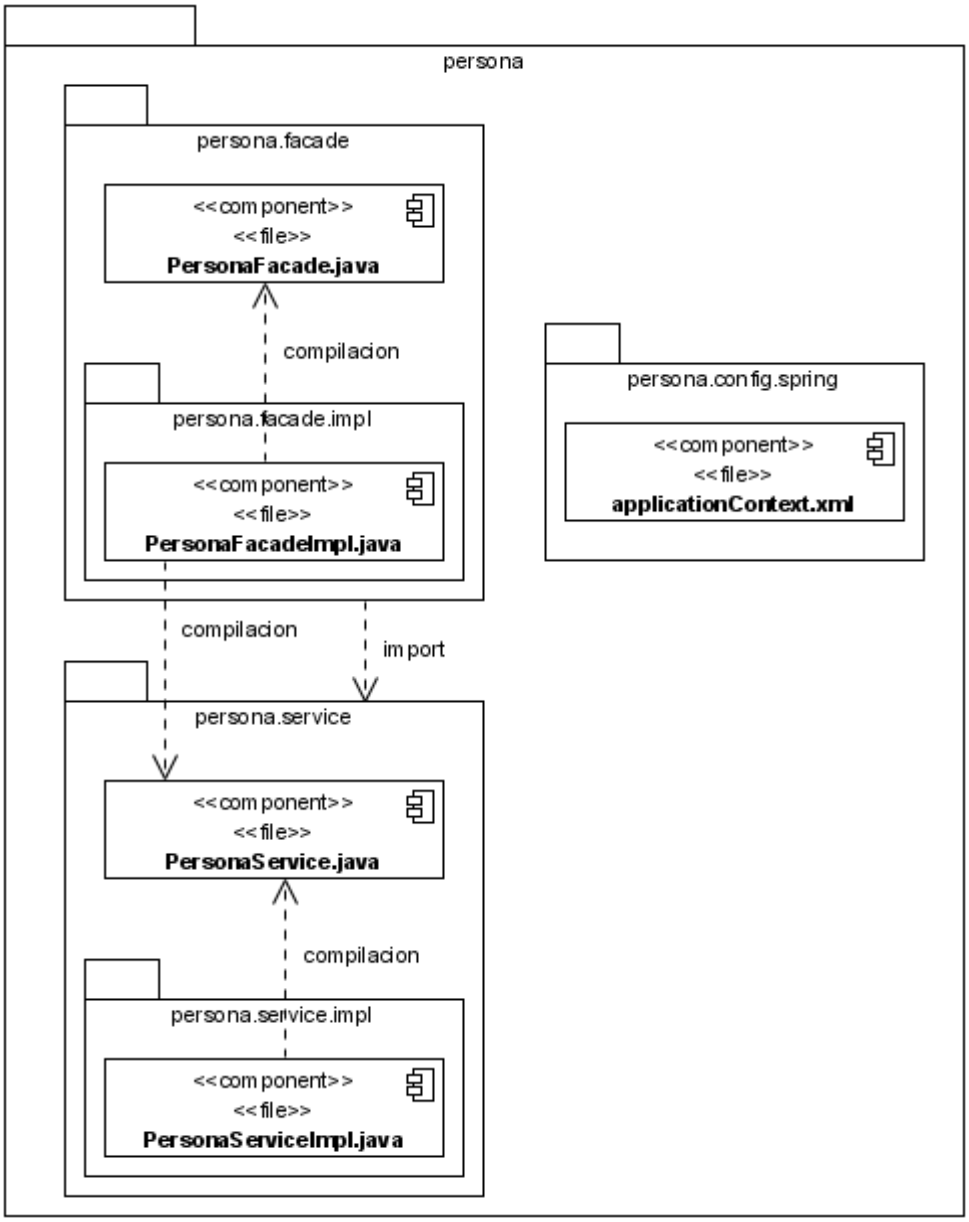

Fig 18. Sub-módulo Persona.

3.3 Conclusión.

En el capítulo se generó el diagrama de despliegue dando una visión de cómo será desplegado el sistema en los nodos de hardware, así como también se analizó en término de componentes cada uno de los sub-módulos pertenecientes al módulo Análisis de Información del SIIPOL.

CONCLUSIONES GENERALES

Para lograr un producto robusto se necesita de varias herramientas, y el mérito estará en lograr una medida justa entre el poder de análisis y el resultado final, por lo que se ha desarrollado una capa de lógica de negocio para el módulo Análisis de Información del SIIPOL que ha logrado integrar correctamente la aplicación y que ha dado como resultado la algoritmización de las funcionalidades que soportan los requisitos tanto funcionales como no funcionales relacionados al módulo, esperando mejorar la rapidez, eficiencia y agilidad con que se procesa la información que se trabaja en el CICPC. Durante la realización de este trabajo se obtienen las siguientes conclusiones:

- Se logró una clara representación de las especificidades que debe cumplir la aplicación.
- Se realizaron los pasos propuestos por la metodología RUP para el análisis, diseño e implementación.
- El estudio previo del Modelo de Casos de Uso del Sistema sirvió como punto de partida para la realización del presente trabajo.
- Se logró definir cómo funcionaría el sistema teniendo en cuenta que cumpliera los requisitos funcionales y no funcionales del módulo.
- Se ha demostrado la eficacia de los lenguajes y tecnologías utilizadas para el desarrollo del sistema.
- La solución propuesta ha sido acertada, los requerimientos soportan al sistema y los casos de uso satisfacen las necesidades funcionales.

Con el estudio realizado y la propuesta de análisis, diseño e implementación de lógica de negocio para el módulo Análisis de Información, se cumple con el objetivo propuesto: Analizar, diseñar e implementar una capa de lógica de negocio para el módulo Análisis de Información del SIIPOL que se adapte a los requisitos funcionales y no funcionales de la aplicación.

RECOMENDACIONES

Con el objetivo de mejorar y concluir una versión más completa del prototipo propuesto en este documento, recomendamos lo siguiente:

- Continuar este trabajo desarrollando el segundo ciclo de desarrollo.
- Refactorizar el código de la capa de lógica de negocio para mejorar los tiempos de acceso y manejo de la información.
- Implementar la ayuda del prototipo para esta parte de la aplicación.
- Analizar en todo momento qué otras facilidades se le pueden ofrecer a los usuarios finales para mejorar el procesamiento de toda la información del SIIPOL.

REFERENCIAS BIBLIOGRÁFICAS

1. **Manuel Peralta.** *Sistemas de Información.* 2003.
2. DMapas - Mapas Digitales S.A. *Seguridad Ciudadana y Gestión Policial.* [En línea] [Citado el: 29 de 11 de 2007.] http://www.dmapas.com/productos_stegpol.htm.
3. HOY.es. *La Policía Local se moderniza con un programa informático de gestión.* [En línea] [Citado el: 29 de 11 de 2007.] http://www.hoy.es/prensa/20070616/merida/policia-local-moderniza-programa_20070616.html.
4. **R.S. Pressman.** *Ingeniería del Software. Un Enfoque Práctico.* 1998.
5. Wikipedia. *Metodología (Ingeniería de Software).* [En línea] 2007. [Citado el: 2 de 12 de 2007.] http://es.wikipedia.org/wiki/Metodolog%C3%ADa_(ingenier%C3%ADa_de_software).
6. **James Rumbaugh, Ivar Jacobson, Grady Booch.** *El Lenguaje Unificado de Modelado.* s.l. : Addison Wesley Publishing Company, 1999.
7. **C. Walls, R. Breidenbach.** *Spring in Action.* [PDF] s.l. : Manning Publications, 2005.
8. **R. Johnson, J. Hoeller, A. Arendsen.** Spring. Java/J2EE Application Framework. [En línea] 2004-2005. [Citado el: 4 de 12 de 2007.] http://www.springframework.org/documentation.
9. Programación en castellano. *La Capa de Lógica-de-Negocio y el Marco de Trabajo Spring.* [En línea] [Citado el: 5 de 12 de 2007.] http://www.programacion.net/tutorial/jap_jsfwork/3/.
10. **Ivar Jacobson, Grady Booch, James Rumbaugh.** El Proceso Unificado de Desarrollo de Software. s.l. : Addison Wesley Publishing Company, 1999.

BIBLIOGRAFÍA

1. Jacabson, Ivar, Booch, Grady y Rumbaugh, James. El proceso Unificado de Desarrollo de Software. Madrid: Addison Wesley, 2000. 84-7829-036-2.
2. Rumbaugh, James, Jacobson, Ivar y Booch, Grady. Lenguaje Unificado de Modelado. Manual de Referncia. Madrid: Pearson Educación, 2000.
3. Conferencia 1. Introducción a la Ingeniería de Software. teleformacion.uci.cu. [En línea] 2007. [Citado el: 12 de mayo de 2008.] http://teleformacion.uci.cu/file.php/42/Clases_Curso_2007-2008/conferencias/Conferencia_1/Profesores/Conferencia_1.pdf.
4. Conferencia 3 Flujo de trabajo Análisis y Diseño. teleformacion.uci.cu. [En línea] 2007. [Citado el: 15 de mayo 2008.] http://teleformacion.uci.cu/file.php/43/CONFERENCIAS/Conferencia_3/Culminacion_FT_AyD._Modelo_de_datos.pdf.
5. Conferencia 6 Flujo de Análisis y Diseño. teleformacion.cu. [En línea] 2007. [Citado el: 12 de mayo de 2008.] http://teleformacion.uci.cu/file.php/42/Clases_Curso_2007-2008/conferencias/Conferencia_6/Conferencia_6.pdf.
6. Parihar, Mridula. ASP.NET. Madrid: ANAYA, 2002. ISBN: 84-4 15-1385-6.
7. Molpeceres, Alberto. Procesos de Desarrollo: RUP, XP y FDD. T-Systems ITS GmbH, s.l.: 2003.
8. Java 2 Platform, Enterprise Edition (J2EE) Overview. sun.com. [En línea] Sun Microsystem. [Citado el: 23 de 5 de 2008.] http://java.sun.com/j2ee/appmodel.html.
9. Becerril, Francisco. Java a su alcance. Atlanta : McGraw-Hill, 1998. ISBN 970-10-1774-9.
10. Java Comunity Process. jcp.org. [En línea] [Citado el: 30 de 4 de 2008.] http://jcp.org/en/jsr/overview.
11. Quintana, Abraham Otero. Estándares libre y Java: ¿Es el JCP un organismo que crea estándares libres? javahispano.net. [En línea] [Citado el: 10 de 5 de 2008.] javahispano.net/frs/download.php/127/JavaYEstandaresLibres.pdf.
12. Hapner, Mark y Shannon, Bill. JSR 151: JavaTM 2 Platform, Enterprise Edition 1.4 (J2EE 1.4) Specification. Java Comunity Process. [En línea] Sun Microsystem. [Citado el: 17 de marzo de 2008.] http://jcp.org/en/jsr/detail?id=151.
13. Allamaraju, Subrahmanyam, y otros. Programación Java Server con J2EE 1.3. s.l. : ANAYA.
14. Bogss, Wendy y Bogss, Michael. Mastering UML with Rational Rose. 2002. 0−7821−4017−3.
15. Gallardo, David, Burnette, Ed y McGovern, Robert. Eclipse in Action. s.l.: Manning, 2003.
16. NetBeans - The Only IDE You Need. www.netbeans.org. [En línea] [Citado el: 1 de junio de 2008.] http://www.netbeans.org/features/.
17. Comparativas. Seam City. [En línea] [Citado el: 10 de febrero de 2008.] http://seamcity.madeinxpain.com/archives/date/2007/12/16.
18. JBoss Seam. JBoss. [En línea] [Citado el: 10 de febrero de 2008.] http://www.jboss.com/products/seam.
19. Walls, Craig. Spring in Action. Greenwick : Manning, 2008. ISBN 1-933988-13-4.
20. Young, Mike y Young, Curtis W. Deploying Solutions with .NET Enterprise Servers. Canadá: Wiley, 2003.
21. Yang Shen, Derek. Integración de JSF, Spring e Hibernate para crear una Aplicación Web del Mundo Real. Programación en Castellano. [En línea] [Citado el: 9 de febrero de 2008.] http://www.programacion.net/tutorial/jap_jsfwork/3/#34_logica.

22. Vilas, Ana Fernández. Grupo de Redes e Ingeniería de Software. [En línea] 20 de 03 de 2001. [Citado el: 6 de 05 de 2008.] www-gris.det.uvigo.es/~avilas/UML/node50.html.
23. Savit, Jeff, Wilcox, Sean y Jayaraman, Bhuvana. Java para la empresa. México : McGraw-Hill, 2000. 970-10-2567-9.
24. Sanchez, María A. Mendoza. Metodologías De Desarrollo De Software. Informatízate. [En línea] 7 de junio de 2004. [Citado el: 10 de febrero de 2008.] http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html.
25. Quiroga, Lourdes Aja. Gestión de información, gestión del conocimiento y gestión de la calidad en las organizaciones. Gestión de información, gestión del conocimiento y gestión de la calidad en las organizaciones. [En línea] 10 de mayo de 2002. [Citado el: 9 de enero de 2008.] http://bvs.sld.cu/revistas/aci/vol10_5_02/aci04502.htm.
26. Pressman, Roger S. Ingeniería del Software. Un enfoque práctico. España : McGraw-Hill, 2002.
27. Herrera, Cristhian. Comparativa entre EJB 3 y Spring Framework. Adictos al Trabajo. [En línea] 17 de octubre de 2007. [Citado el: 9 de febrero de 2008.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=EJB3vsSpring#1.Comparativa%20entre%20EJB%203%20y%20Spring%20Framework|outline.
28. García Molina, Jesus J., Moreira, Ana y Rossi, Gustavo. Asociación de Técnicos de Informática. [En línea] [Citado el: 28 de 02 de 2008.] http://www.ati.es/novatica/2004/168/168-4.pdf.
29. D.D.I.Software. Conferencia 5.Fase de Inicio. Flujo de Análisis y Diseño. Modelo de. 2006.
30. Canales Mora, Roberto. Adictos al Trabajo. [En línea] 02 de 02 de 2004. [Citado el: 17 de 03 de 2008.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=vparadigm.
31. Bartle, Phil. Información para la Gestión y Gestión de la Información. Potenciación Comunitaria. [En línea] [Citado el: 9 de febrero de 2008.] http://www.scn.org/mpfc/modules/mon-miss.htm.
32. Visual Paradigm for UML. User's Guide (Part 1).
33. Visual Paradigm. [En línea] [Citado el: 29 de 03 de 2008.] http://www.visual-paradigm.com/product/vpuml/.
34. Indudata. Soluciones en Informática. [En línea] [Citado el: 10 de 03 de 2008.] http://www.indudata.com/1rational_rose.htm.
35. Programas a Medida. [En línea] 2008. [Citado el: 27 de 02 de 2008.] http://www.haztuprograma.com/tiposAplicaciones.html.
36. JavaHispano. [En línea] 2002. [Citado el: 27 de 02 de 2008.] http://www.javahispano.org/contenidos/es/aplicaciones_de_escritorio_eficientes.
37. About Spring. Spring Framework. [En línea] [Citado el: 10 de febrero de 2008.] http://www.springframework.org/about.

GLOSARIO

A continuación, en orden alfabético, se muestra el significado de algunos términos usados en este documento cuyo uso no es común y que pueden dificultar la compresión del mismo:

AJAX: Siglas en inglés de Asinchronous Javascript And Xml (Javascript asincrónico y XML).
AOP: Siglas en inglés de Aspect Oriented Programming (programación orientada a aspectos). Permite al desarrollador separar las tareas que no deben ser mezcladas entre módulos.
API: Siglas en inglés de Application Program Interface (programa de aplicación de interfaz). Conjunto de especificaciones de comunicación entre componentes de software. Representa un método para conseguir abstracción en la programación.
Código abierto: Open Source en inglés. Conlleva como idea más importante la posibilidad de acceder al código fuente, modificarlo y redistribuirlo como se considere conveniente, estando sujeto al marco legal que brinda el open source.
DAO: Siglas en inglés de Data Access Object (objeto de acceso a datos). Se trata de un objeto que realiza una labor de interfaz entre la aplicación y los datos de la misma.
Framework: Marco de trabajo, solución reutilizable y extensible.
Herramienta CASE: (Computer Aided Software Engineering) Ingeniería de Software Asistida por Ordenador. Diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.
Hibernate: Framework libre y de código abierto objeto relacional para el acceso a datos.
IDE: Siglas en inglés de Integrated Development Environment (ambiente integrado de desarrollo).
IoC: Siglas en inglés de Inversion of Control (inversión del control). Mueve la responsabilidad de realizar las tareas al interior del framework y fuera del código de la aplicación.
J2EE: Siglas en inglés de Java 2 Platform Enterprise Edition (edición empresarial de la plataforma Java 2). Comprende un conjunto de especificaciones y funcionalidades orientadas al desarrollo de aplicaciones empresariales.
Java: Tecnología, lenguaje de programación creado por Sun Microsystems.
JDBC: Siglas en inglés de Java DataBase Connectivity (conectividad de Java a bases de datos). Es un API que permite operaciones sobre bases de datos desde el lenguaje Java.

JSF: Siglas en inglés de JavaServer Faces. Framework desarrollado por Sun Microsystems como estándar para la vista de las aplicaciones web en Java.

JSP: Siglas en inglés de JavaServer Pages (páginas web Java del lado del servidor).

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

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

RUP: Siglas en inglés de Rational Unified Process (proceso unificado de desarrollo de software).

Software libre: Propone que los programas deben estar al alcance de todos de manera gratuita.

Spring: Framework de desarrollo para la plataforma Java que por su diseño ofrece muchas facilidades y libertades a los desarrolladores y se considera una alterna a la tecnología de Enterprise JavaBeans.

UML: Siglas en inglés de Unified Modeling Languaje (lenguaje unificado de modelado).

Web Service: Servicio web. Se utilizan para intercambiar información entre aplicaciones de diferente arquitectura.

XML: (Extensible Markup Language) Lenguaje de Marcas Extensible. Metalenguaje extensible de etiquetas desarrollado por el W3C. Es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

ANEXOS

ANEXO 1: Diagramas de Clases del Análisis.

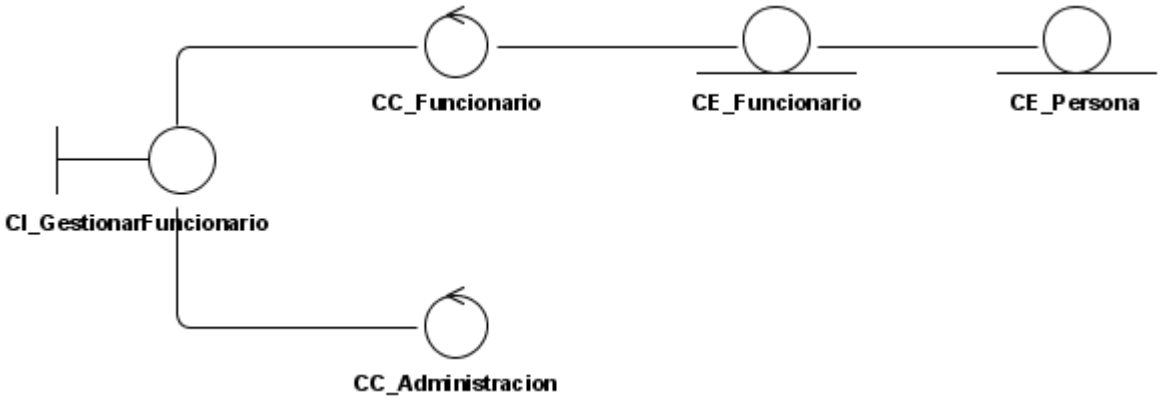

Fig 19. CUS Gestionar Funcionario.

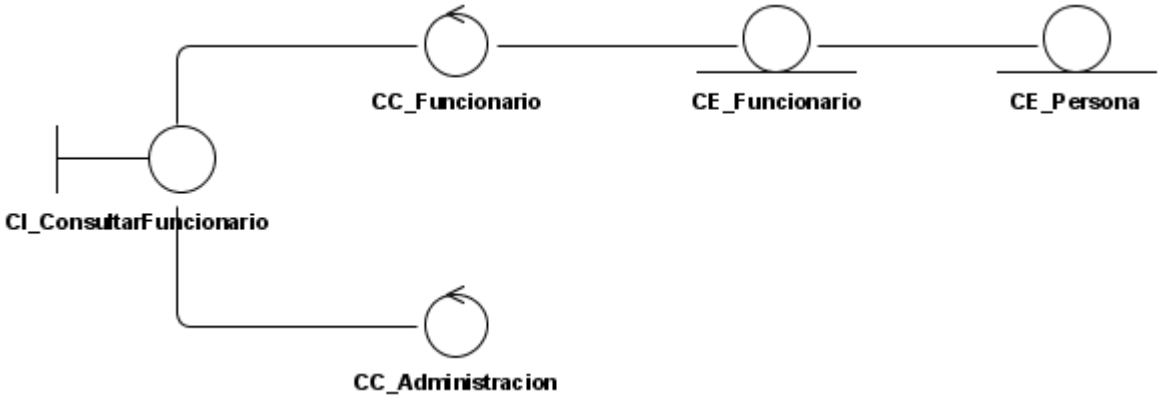

Fig 20. CUS Consultar Funcionario.

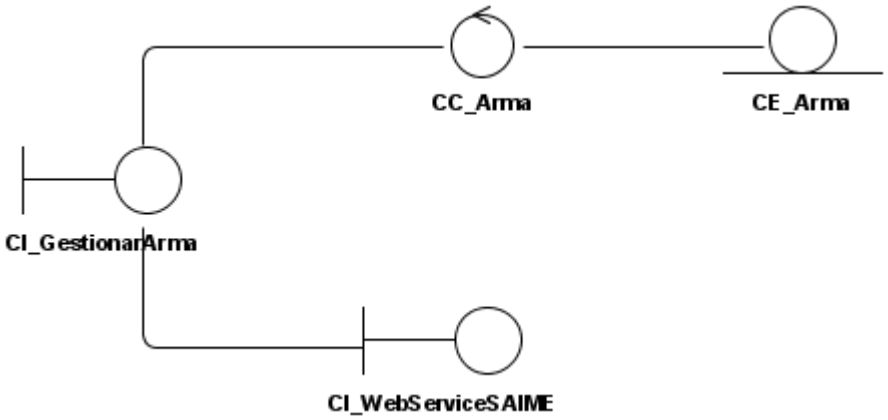

Fig 21. CUS Gestionar Arma.

Fig 22. CUS Consultar Arma.

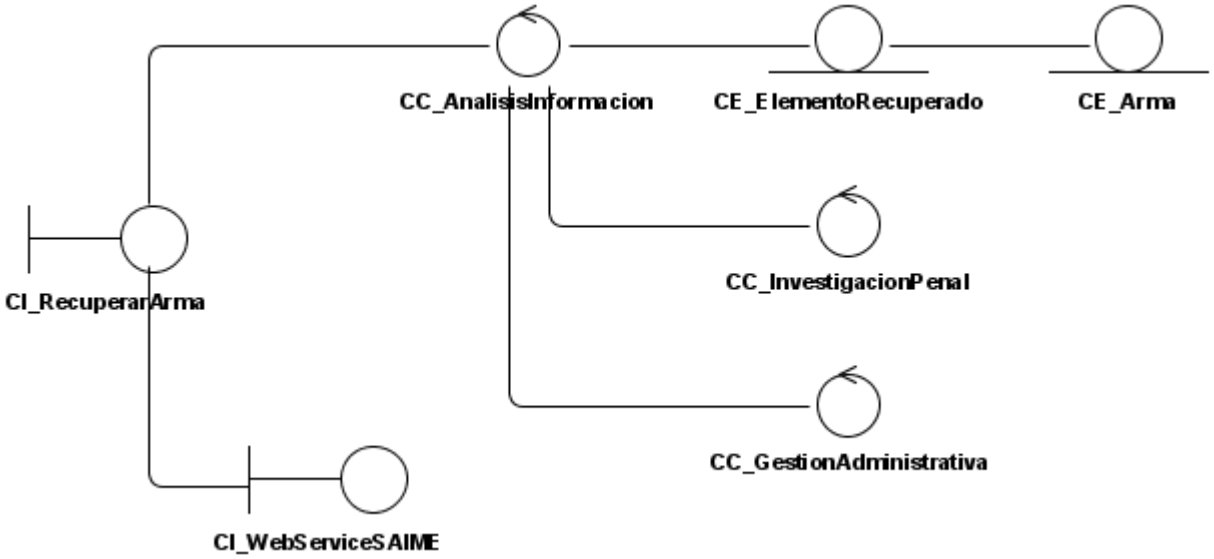

Fig 23. CUS Registrar Recuperación de Arma.

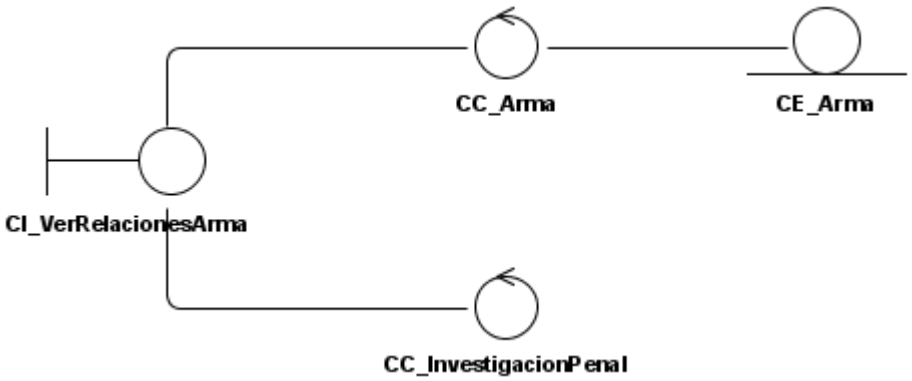

Fig 24. CUS Ver Relaciones de Arma.

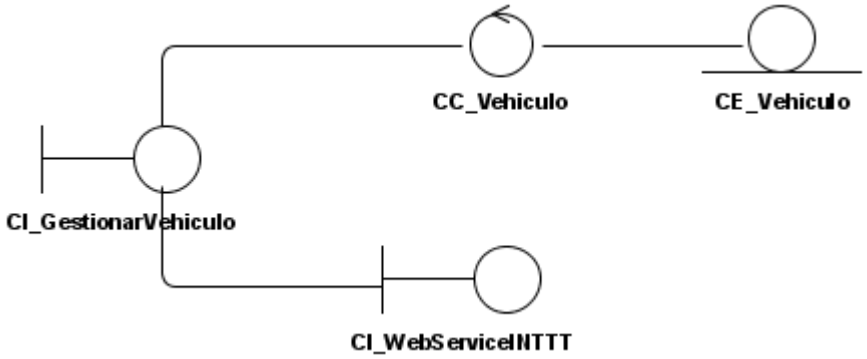

Fig 25. CUS Gestionar Vehículo.

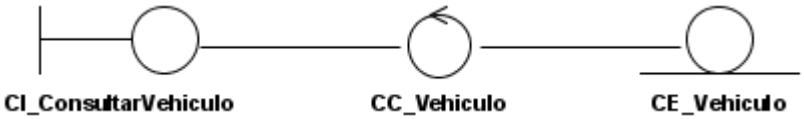

Fig 26. CUS Consultar Vehículo.

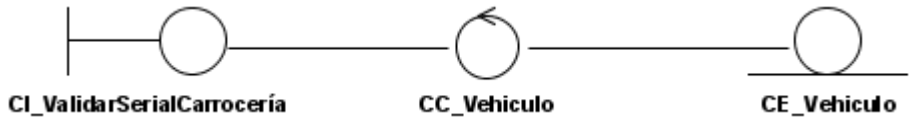

Fig 27. CUS Validar Serial de Carrocería.

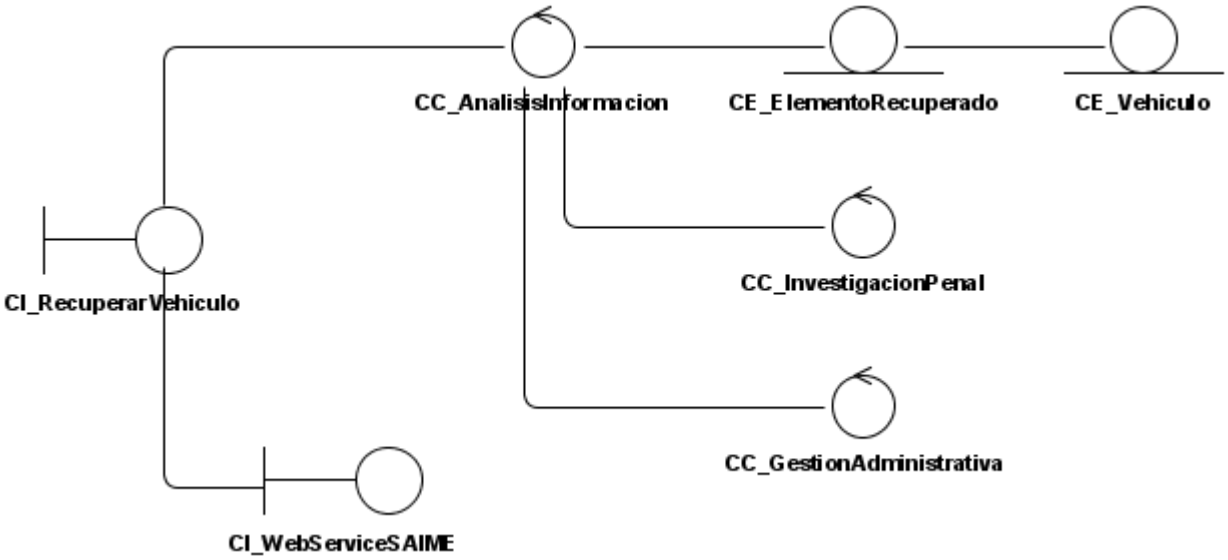

Fig 28. CUS Registrar Recuperación de Vehículo.

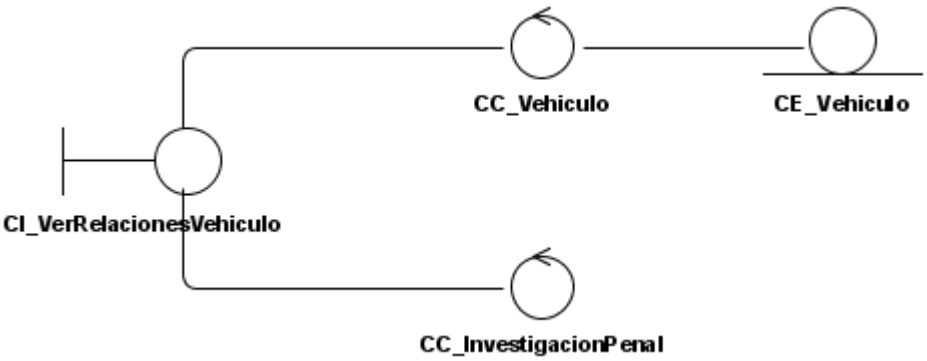

Fig 29. CUS Ver Relaciones de Vehículo.

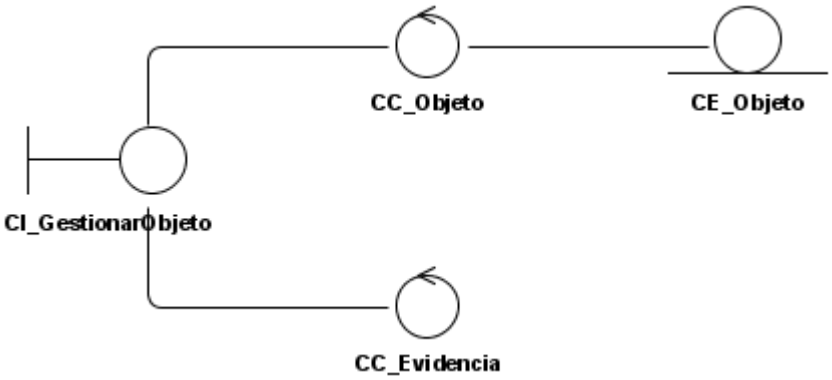

Fig 30. CUS Gestionar Objeto.

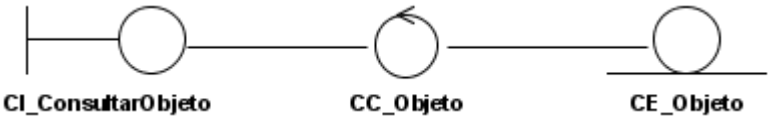

Fig 31. CUS Consultar Objeto.

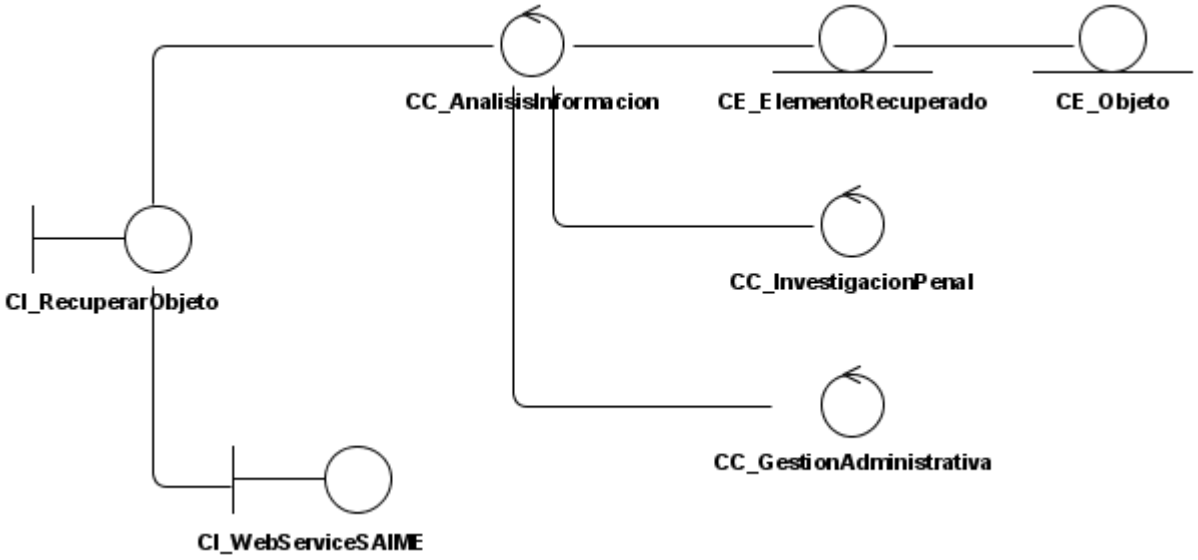

Fig 32. CUS Registrar Recuperación de Objeto.

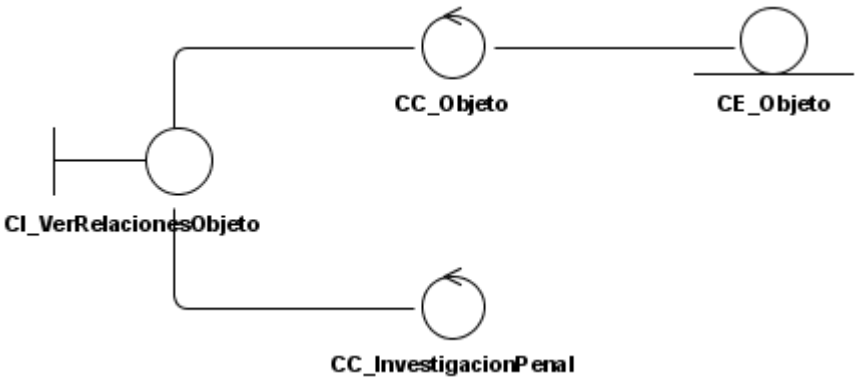

Fig 33. CUS Ver Relaciones de Objeto.

ANEXO 2: Diagramas de Clases del Diseño.

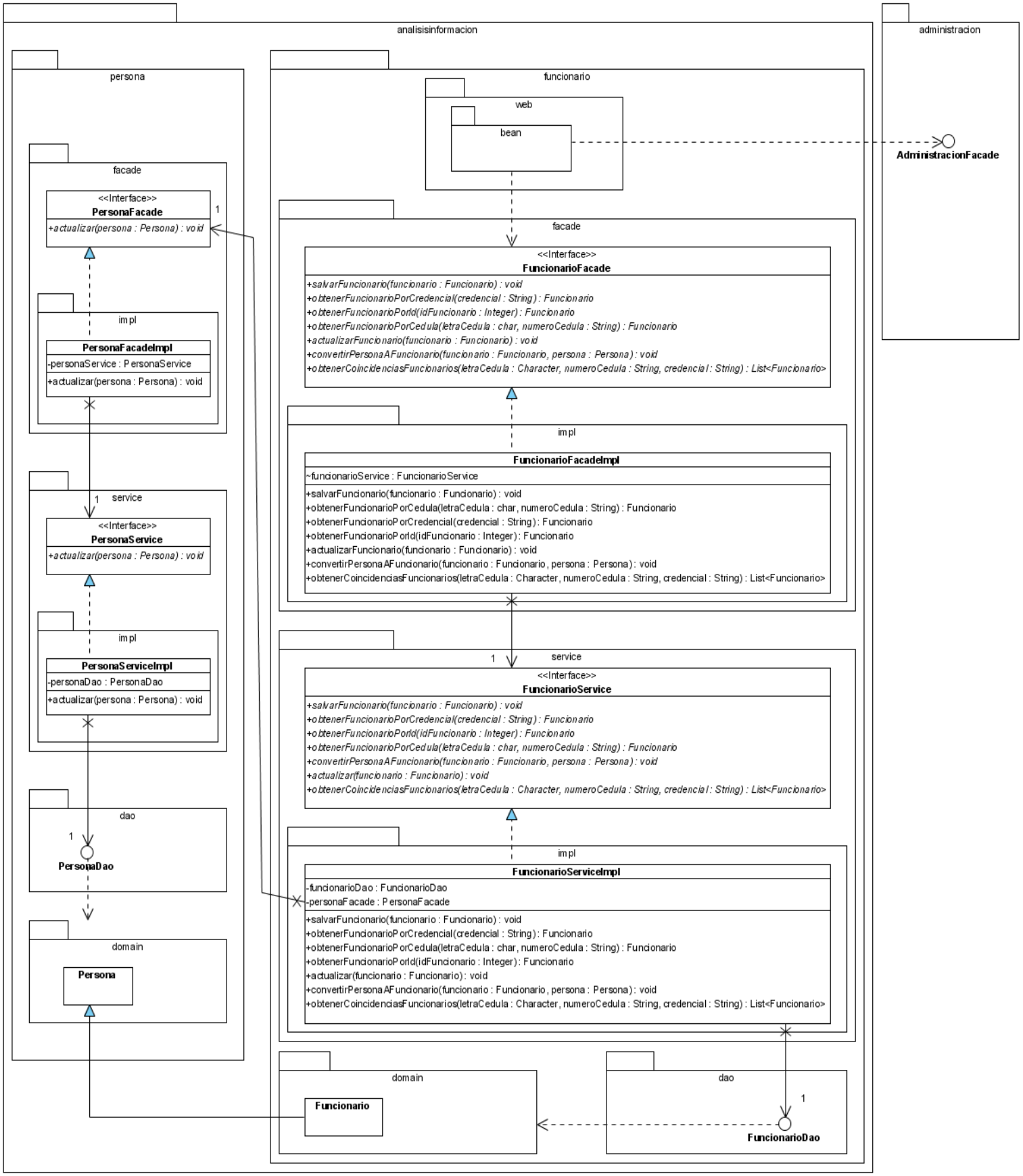

Fig 34. CUS Gestionar Funcionario.

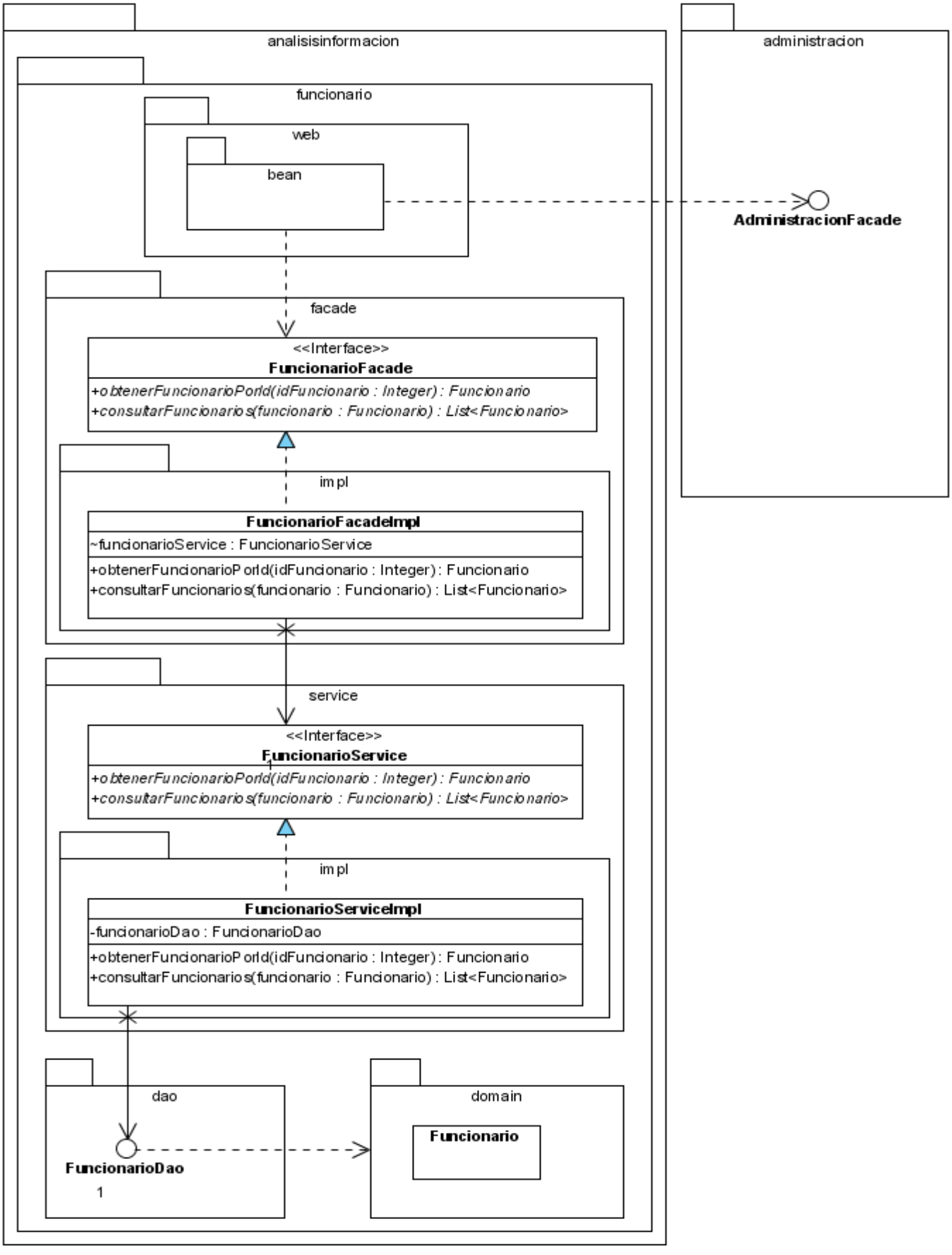

Fig 35. CUS Consultar Funcionario.

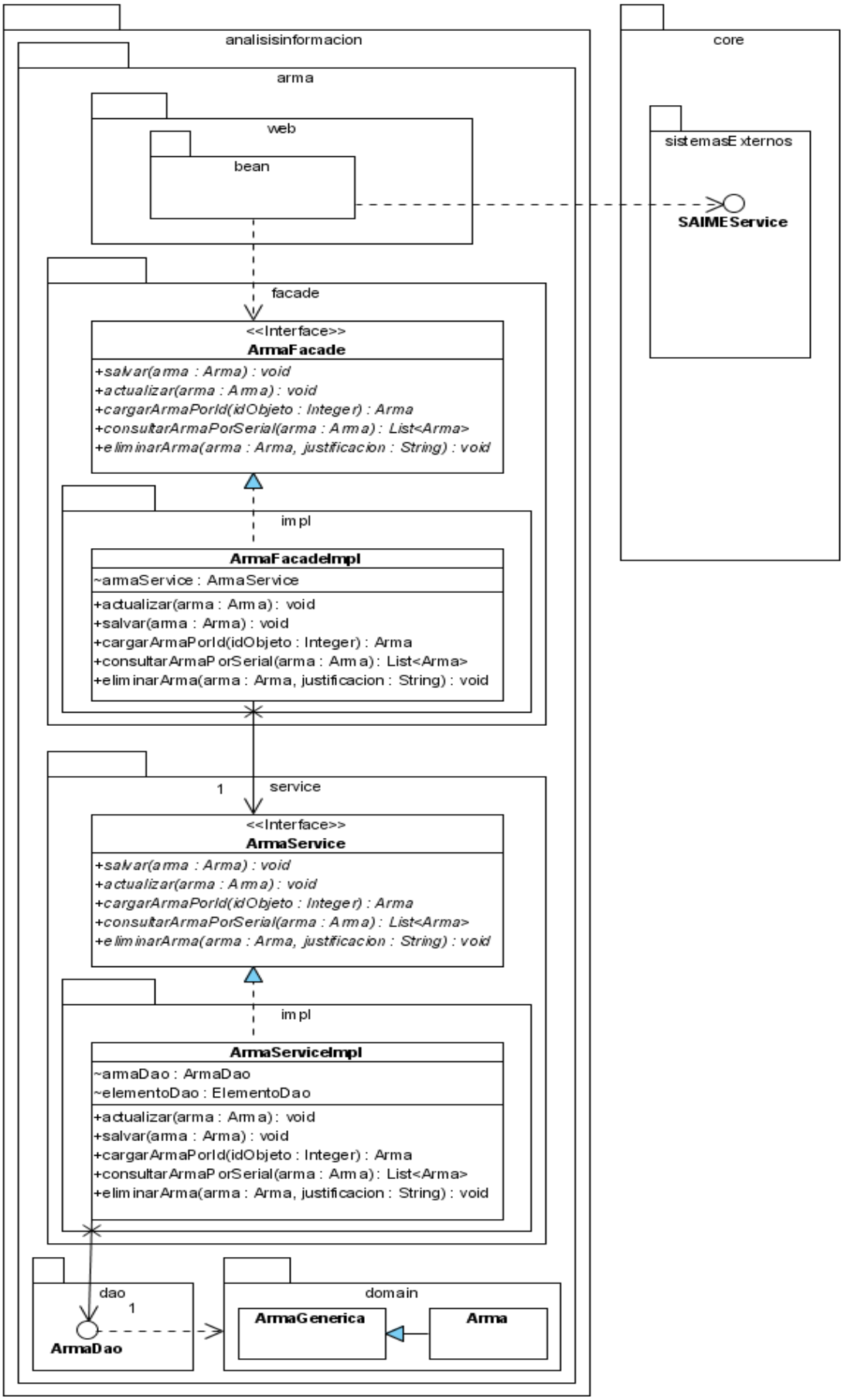

Fig 36. CUS Gestionar Arma.

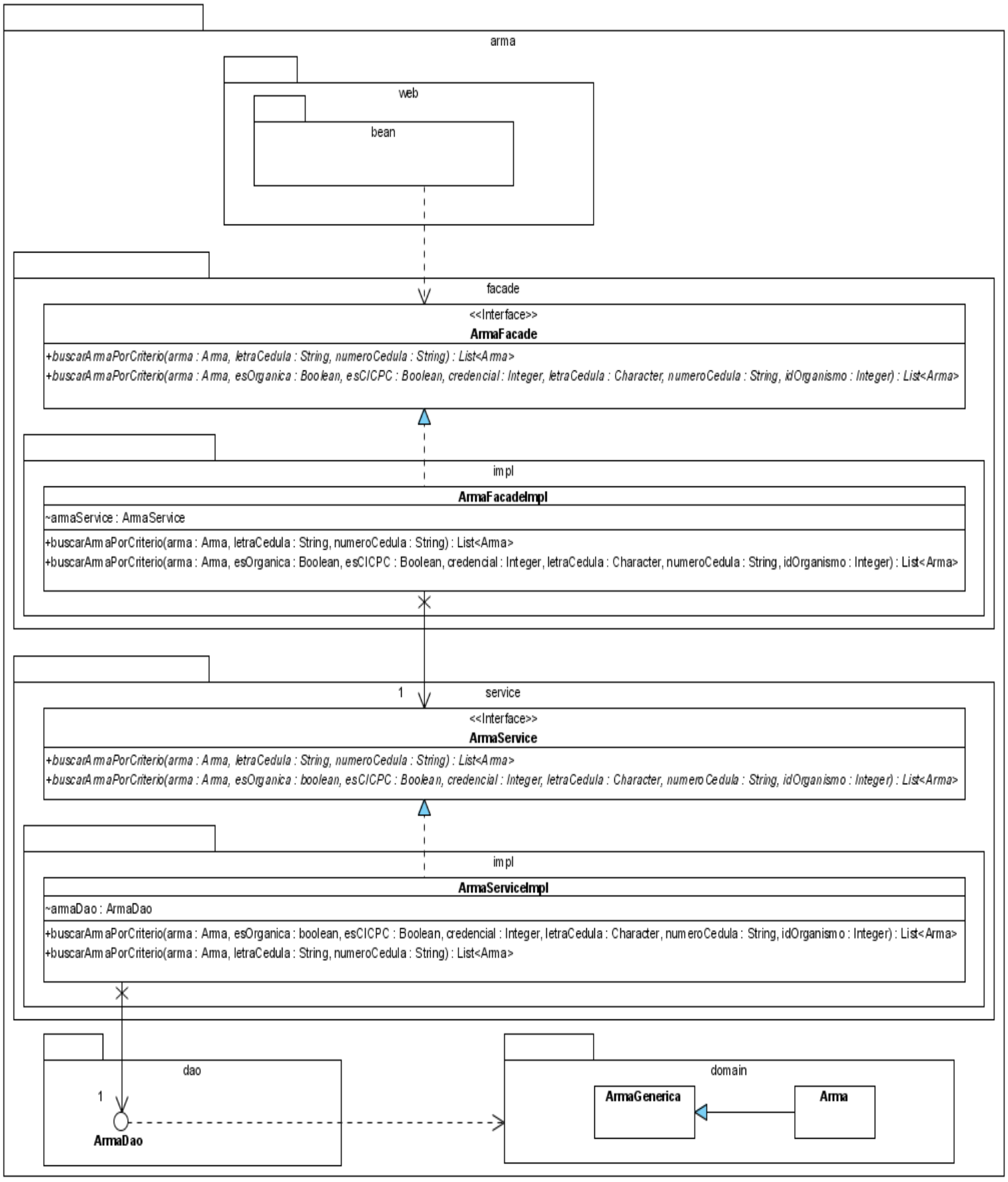

Fig 37. CUS Consultar Arma.

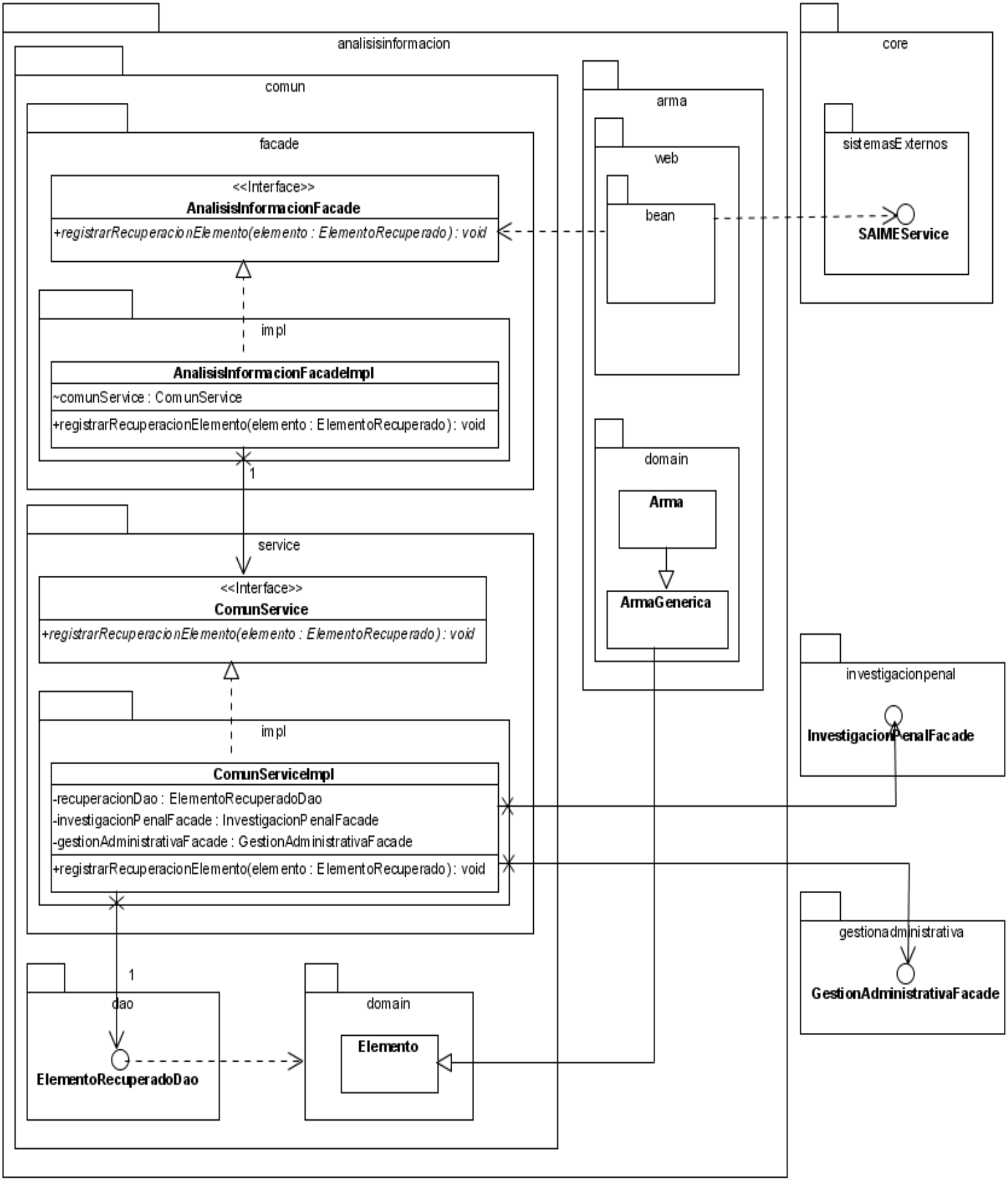

Fig 38. CUS Registrar Recuperación de Arma.

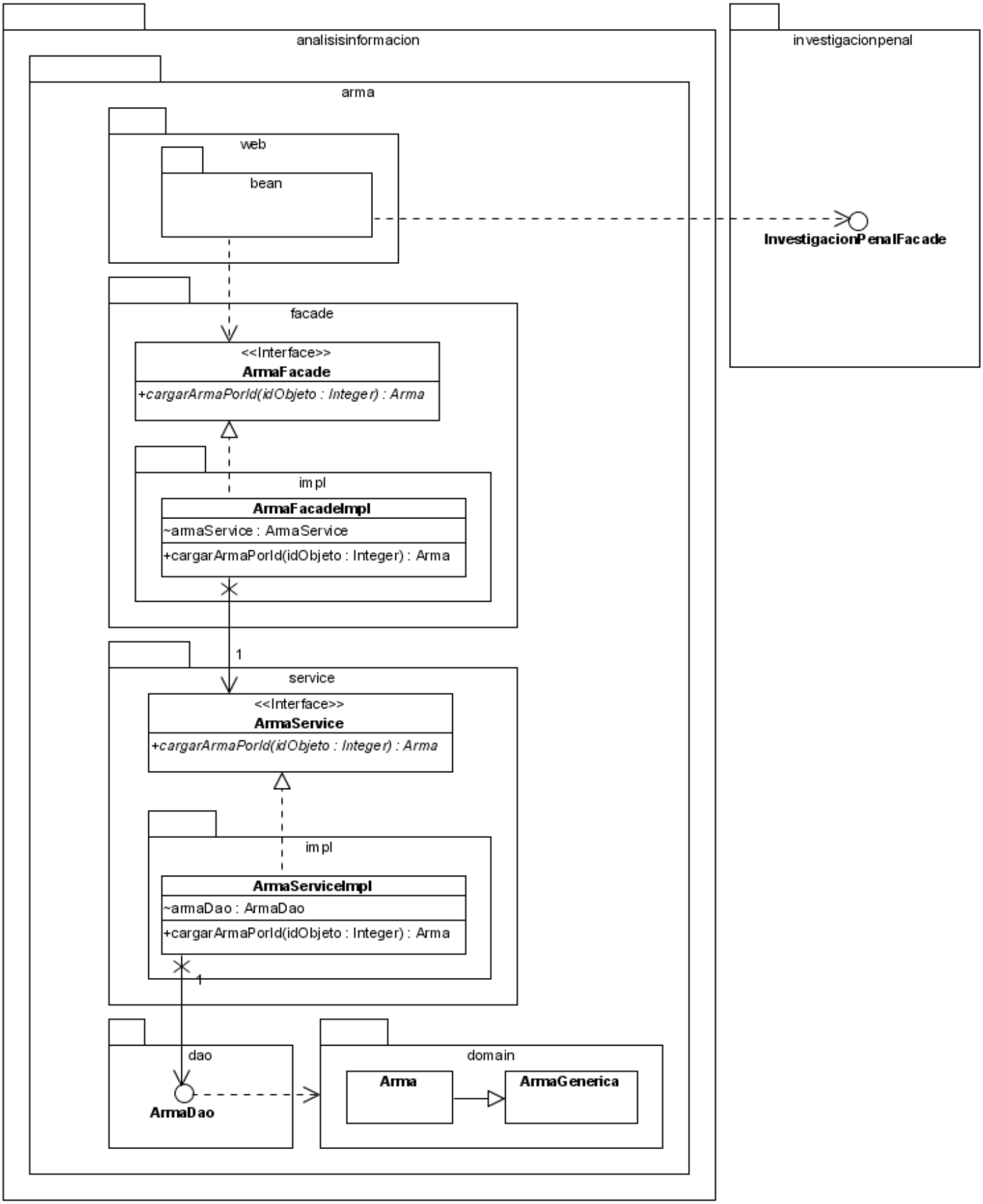

Fig 39. CUS Ver Relaciones de Arma.

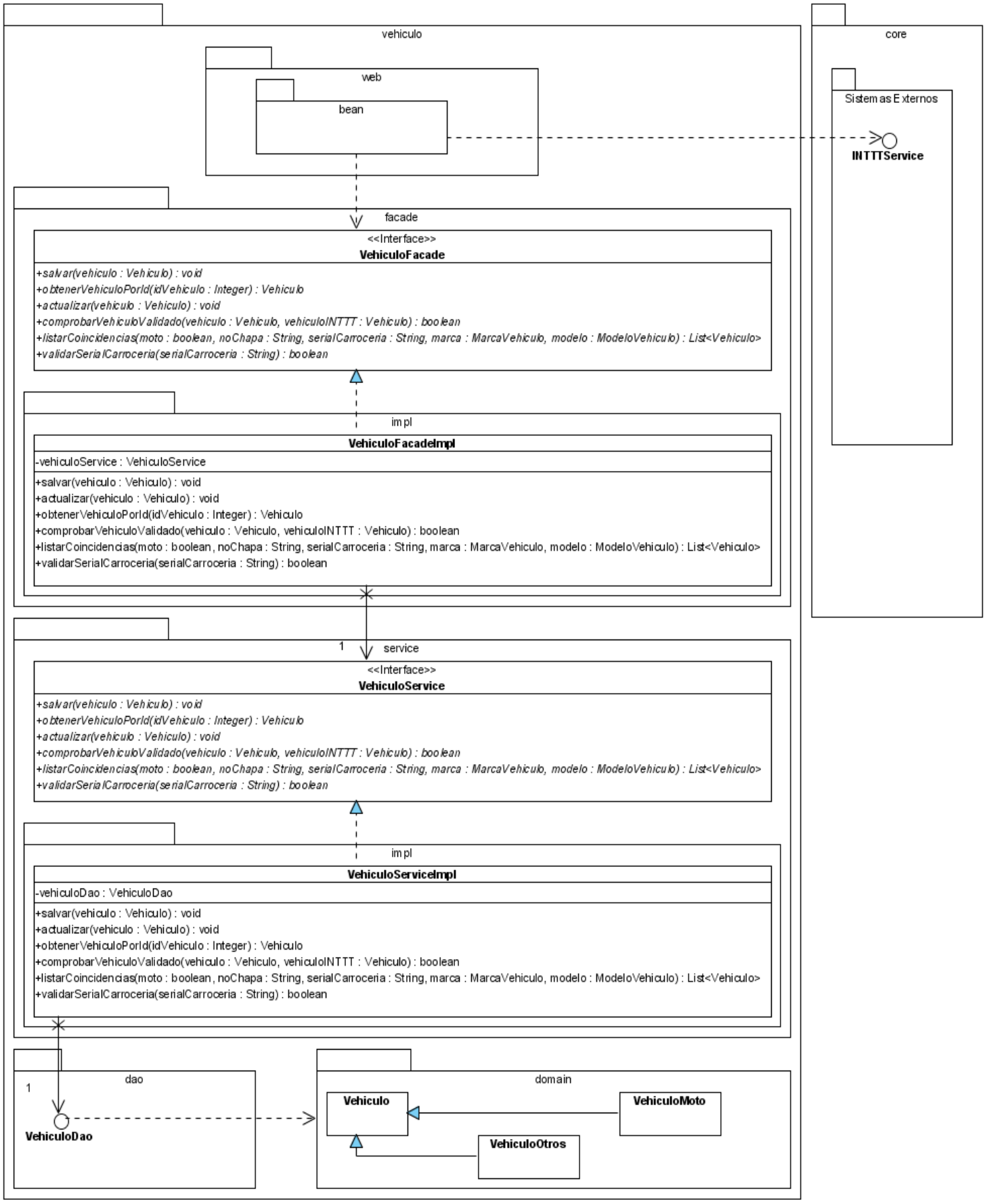

Fig 40. CUS Gestionar Vehículo.

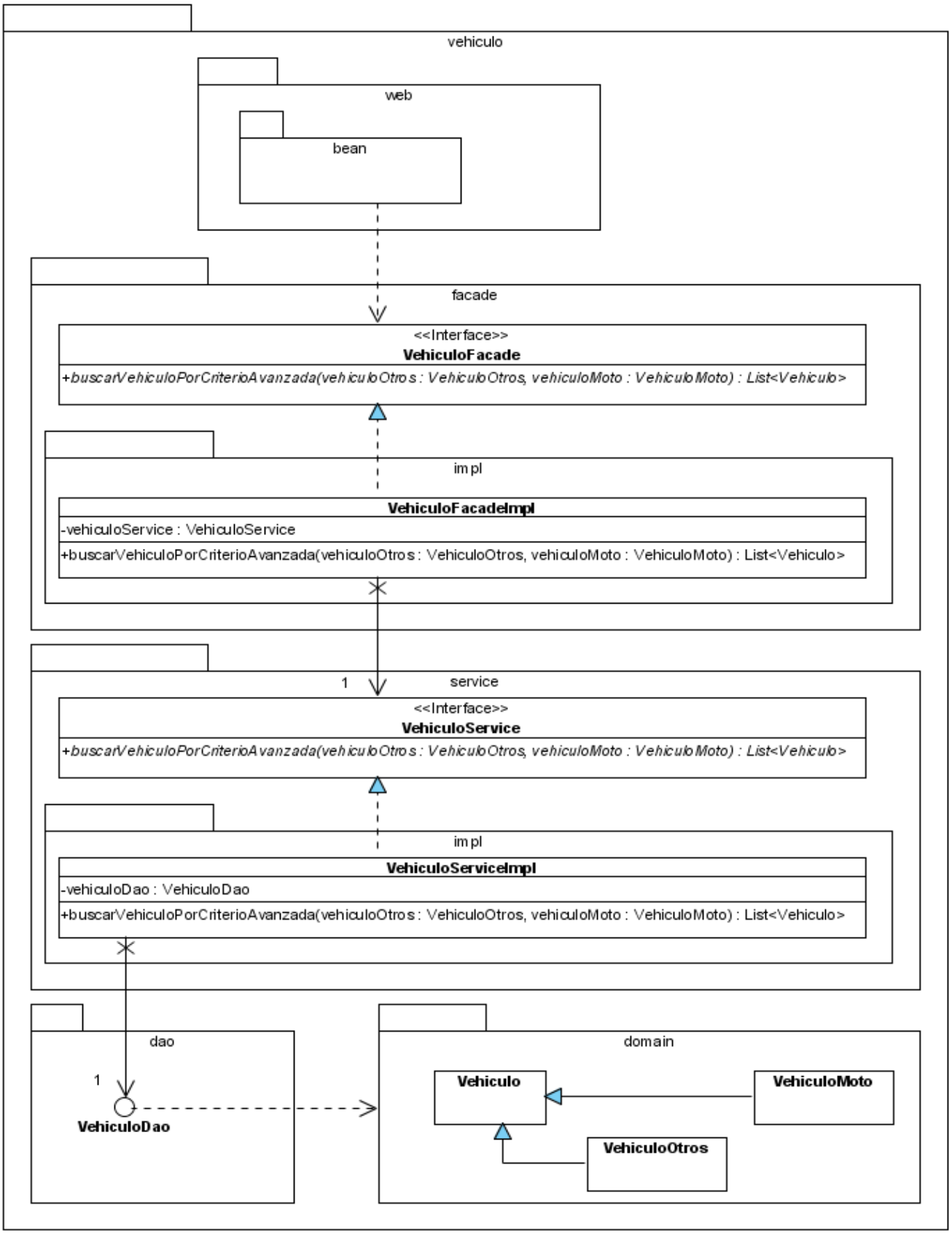

Fig 41. CUS Consultar Vehículo.

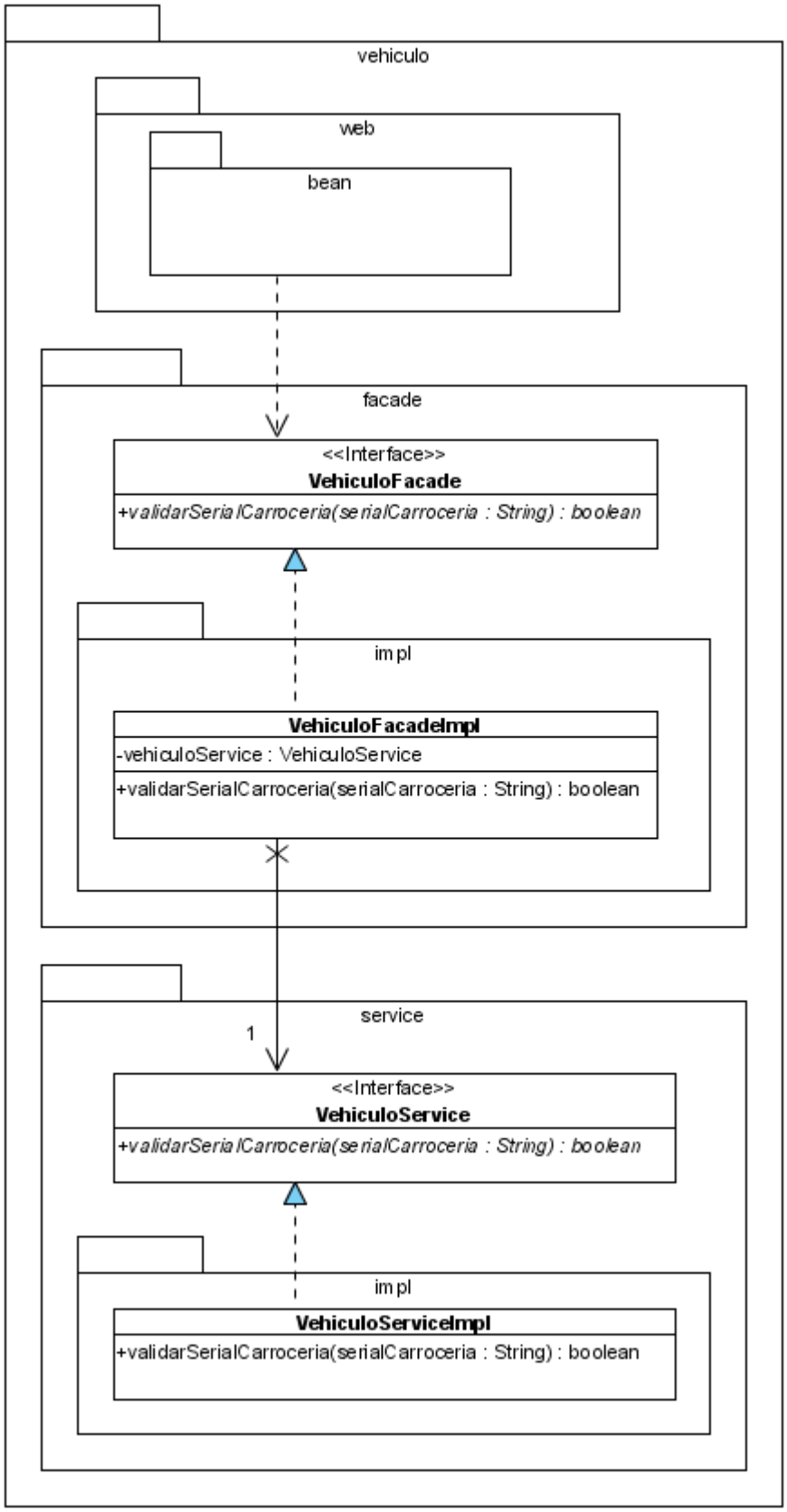

Fig 42. CUS Validar Serial de Carrocería.

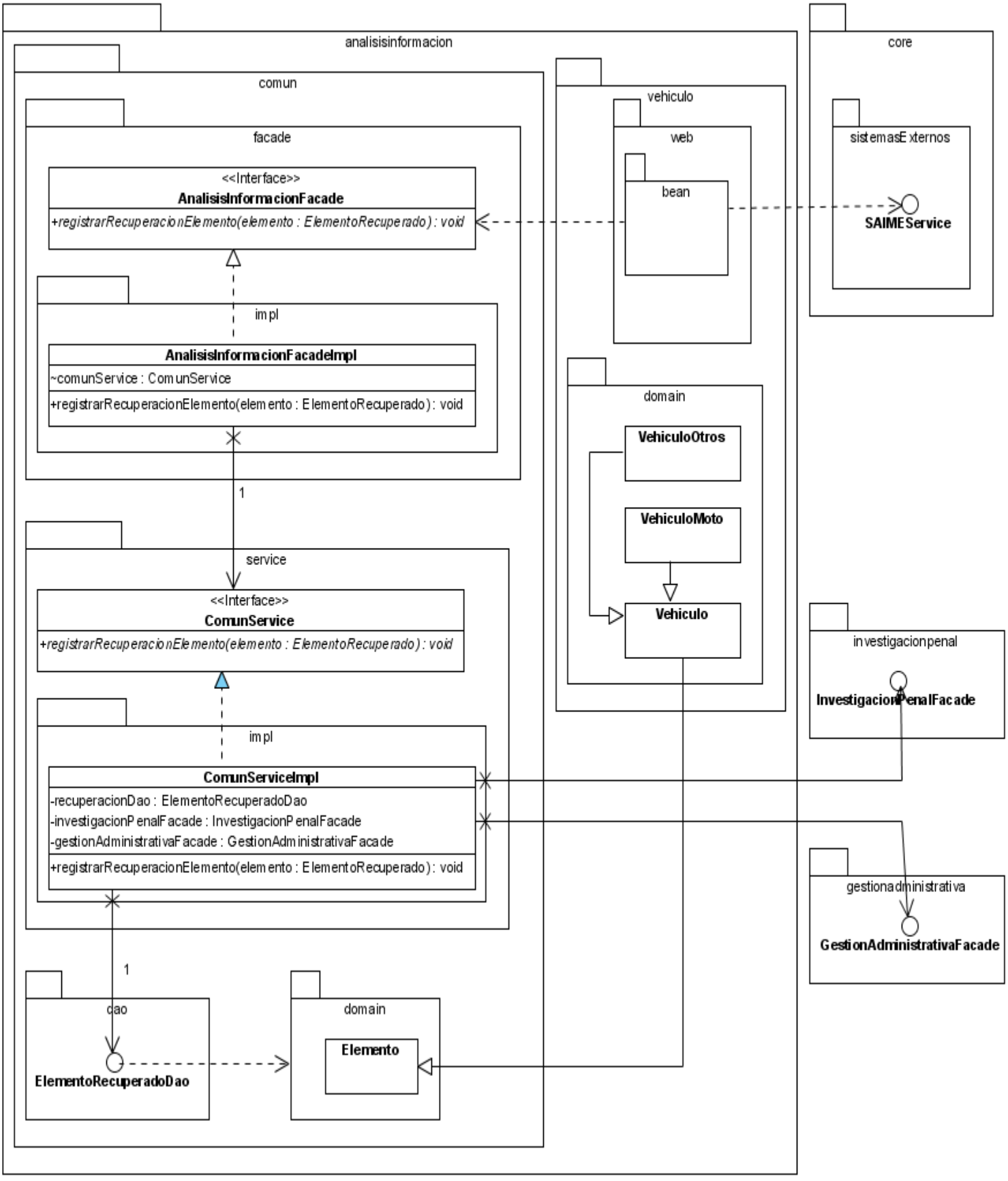

Fig 43. CUS Registrar Recuperación de Vehículo.

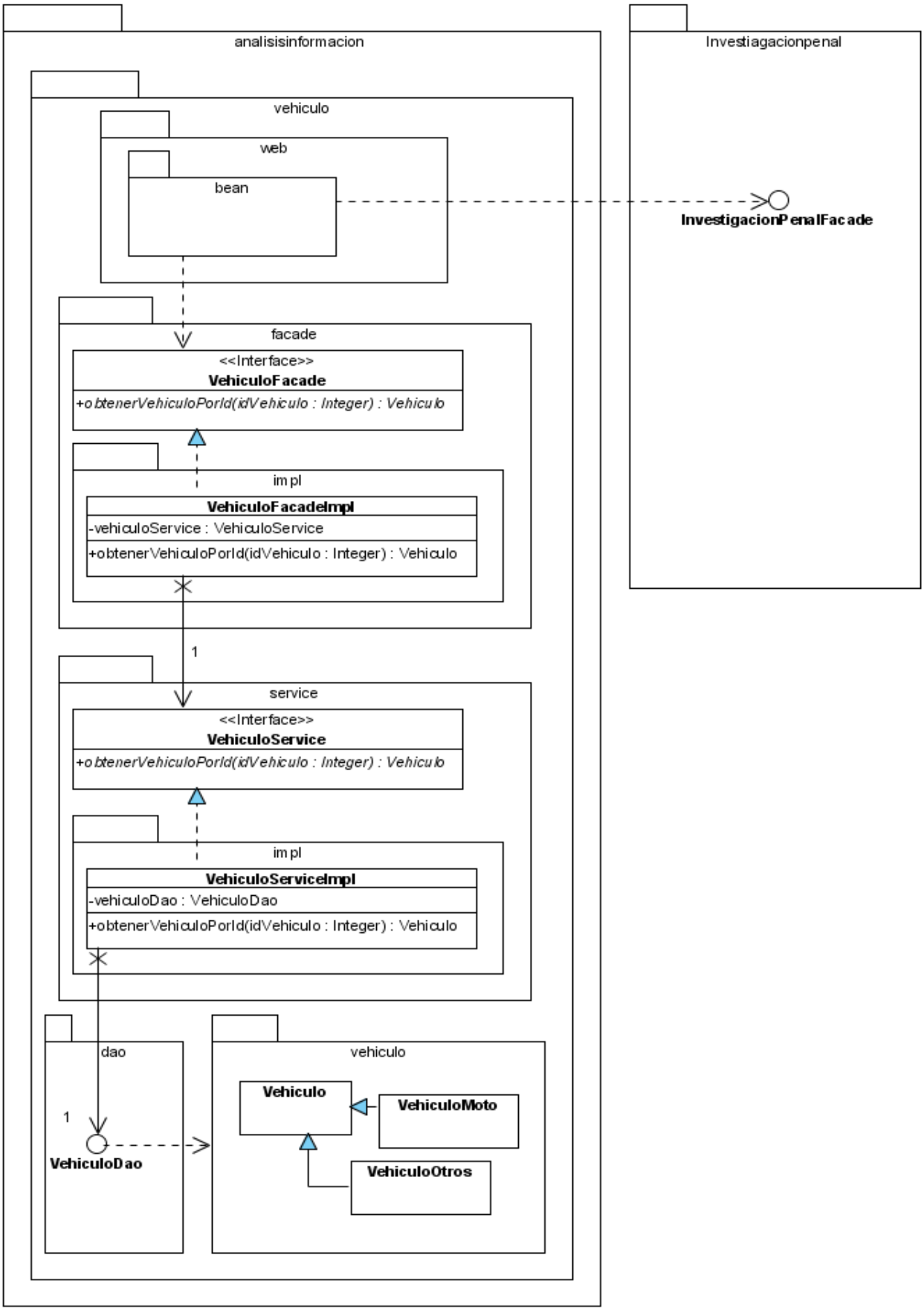

Fig 44. CUS Ver Relaciones de Vehículo.

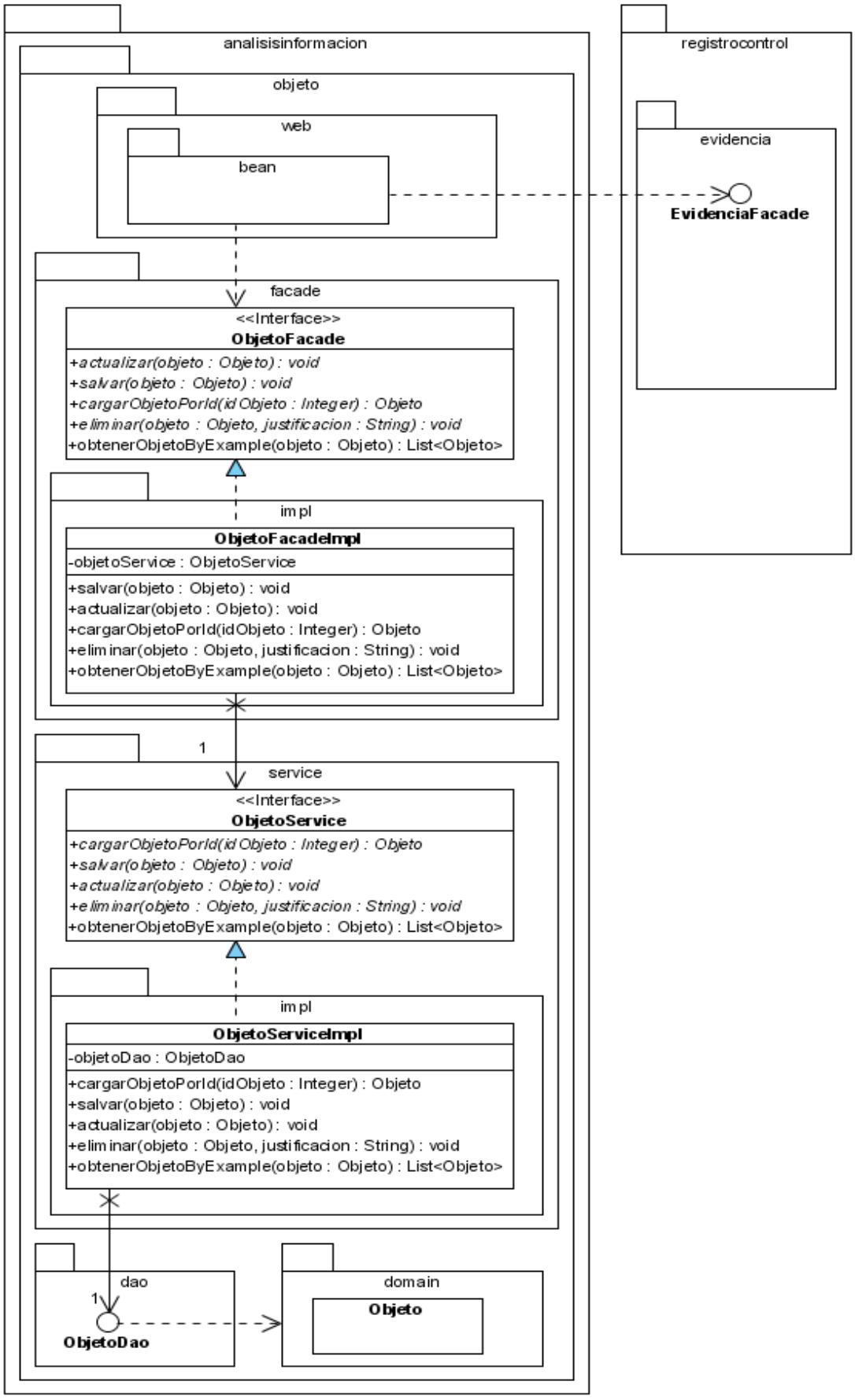

Fig 45. CUS Gestionar Objeto.

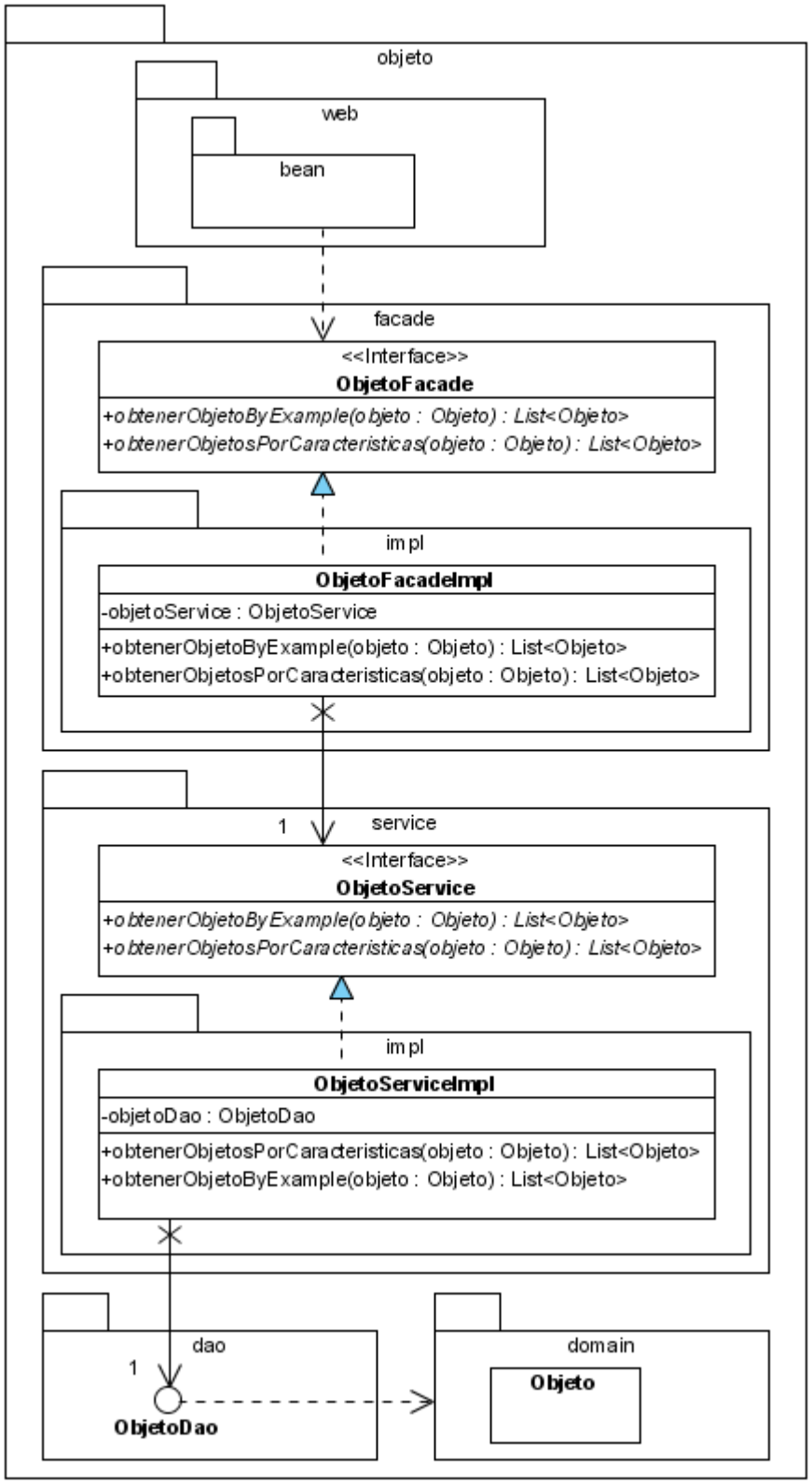

Fig 46. CUS Consultar Objeto.

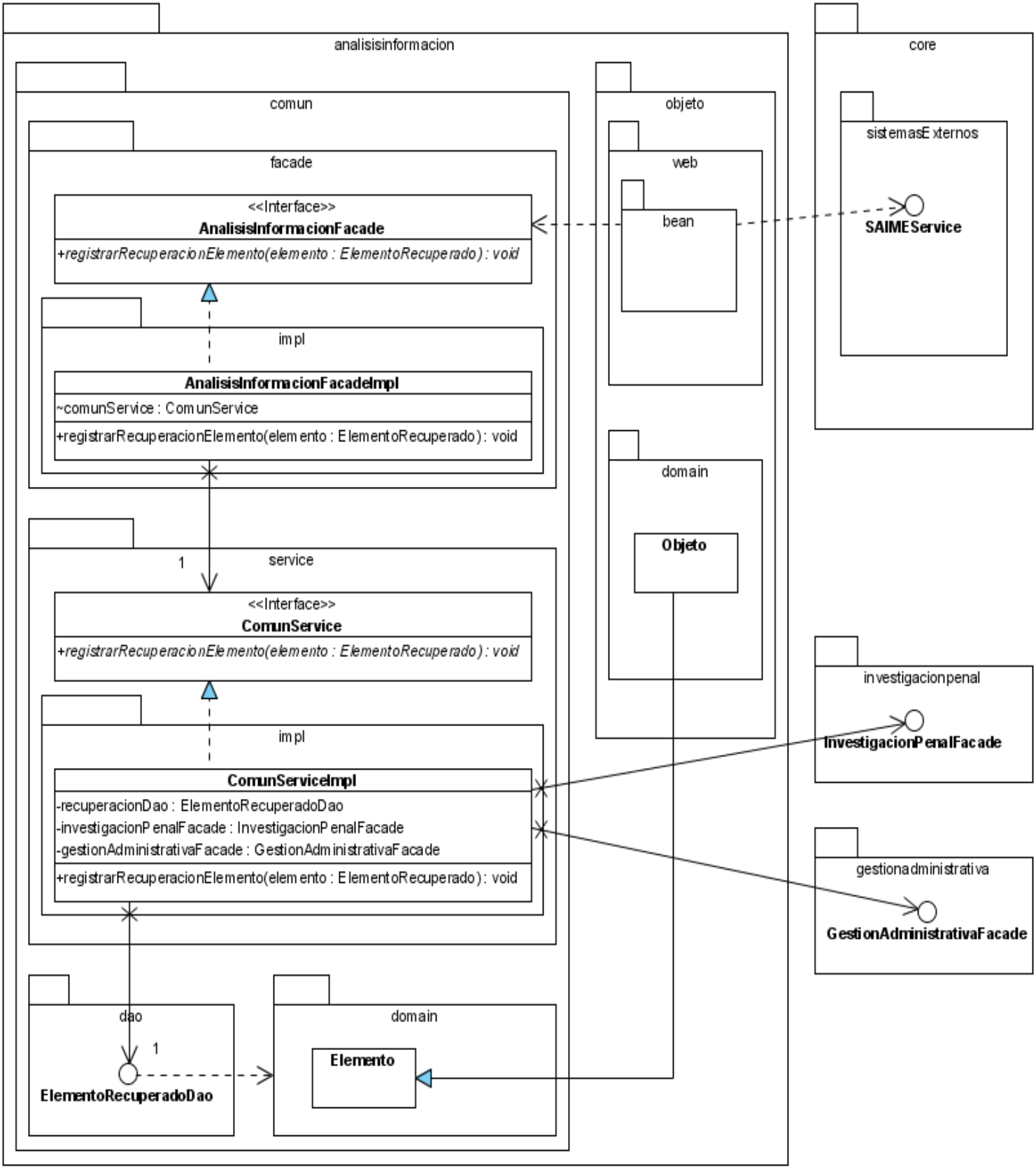

Fig 47. CUS Registrar Recuperación de Objeto.

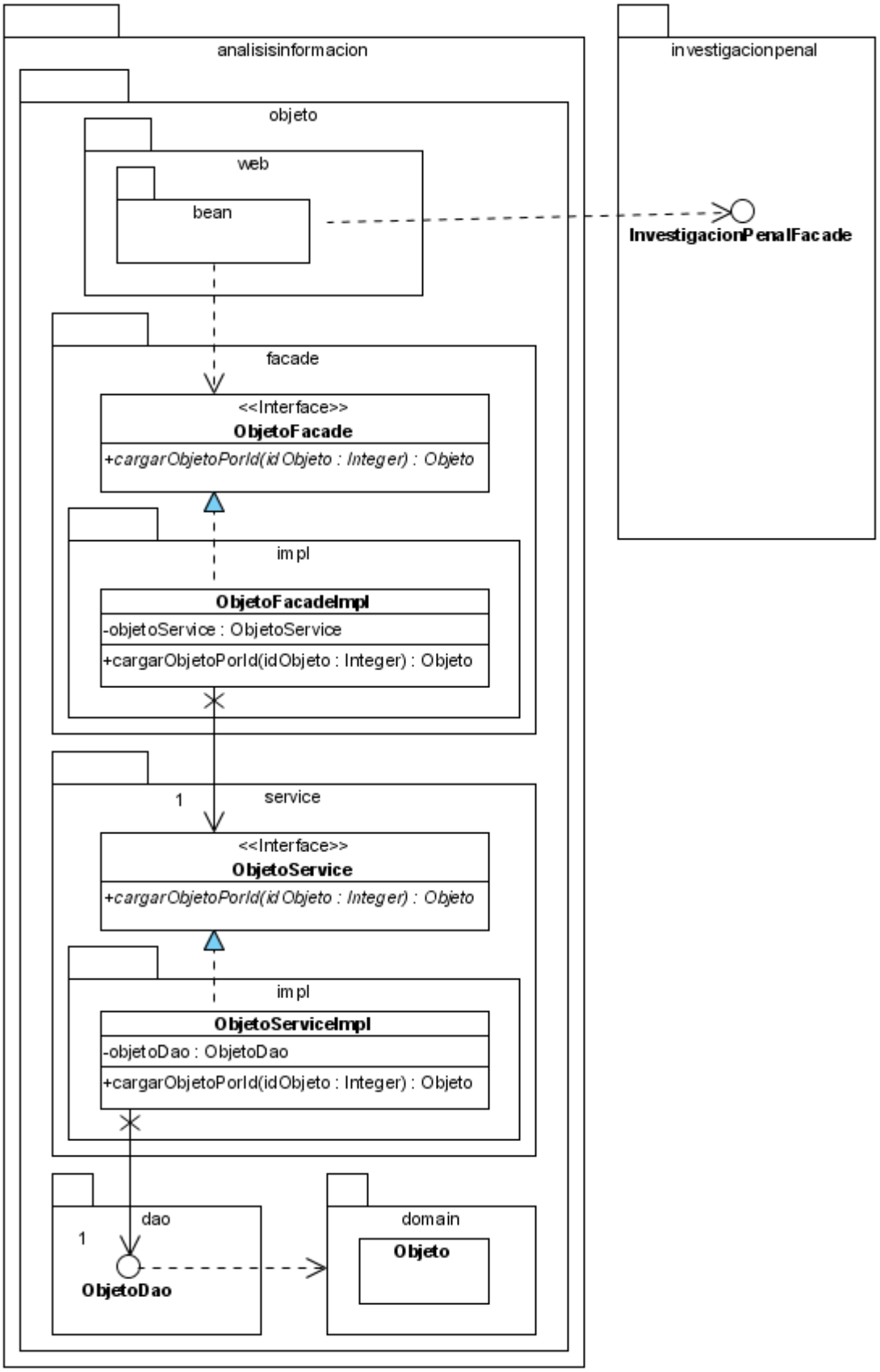

Fig 48. CUS Ver Relaciones de Objeto.

ANEXO 3: Diagramas de Contrato entre Paquetes.

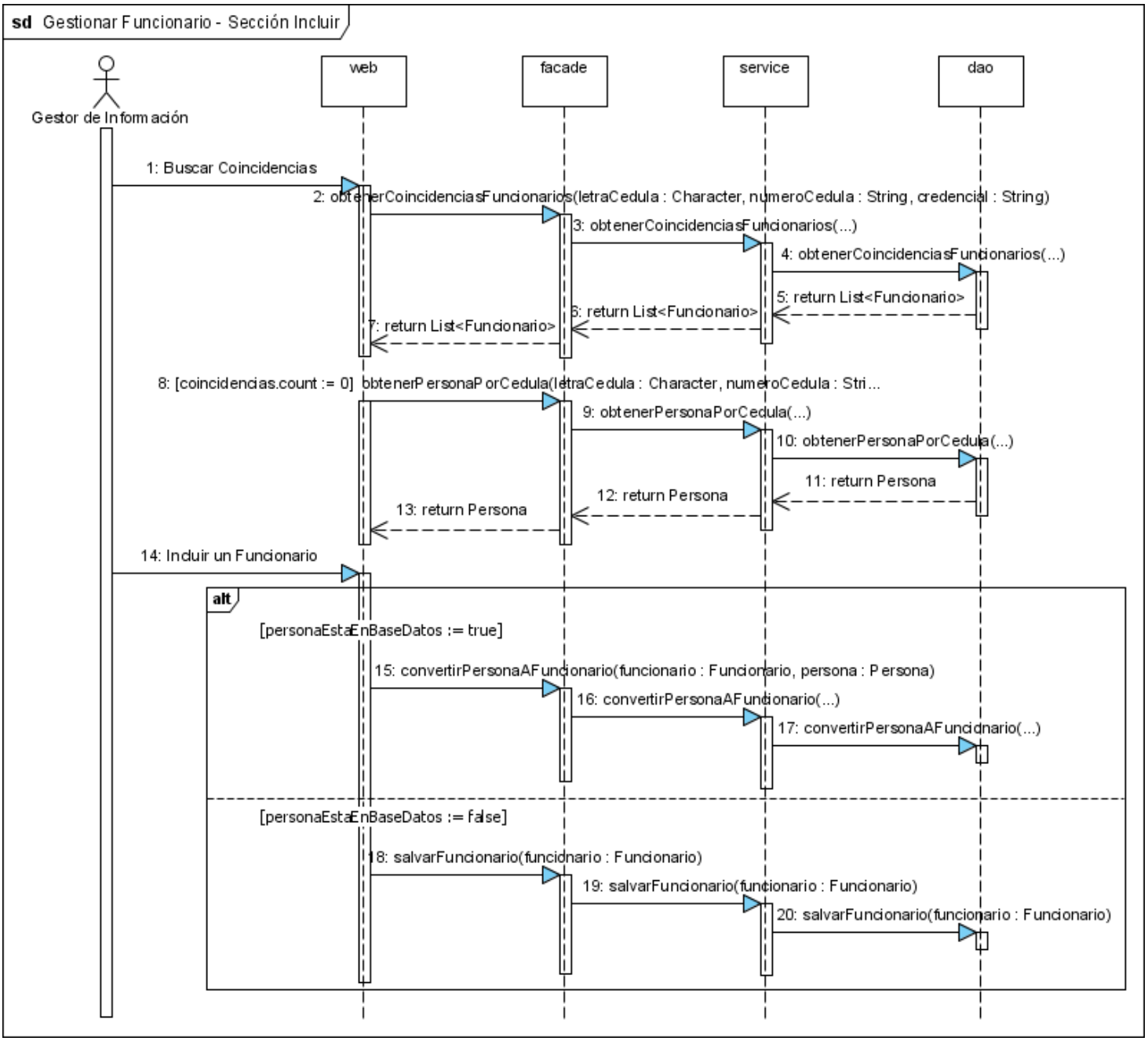

Fig 49. CUS Gestionar Funcionario – Sección Incluir.

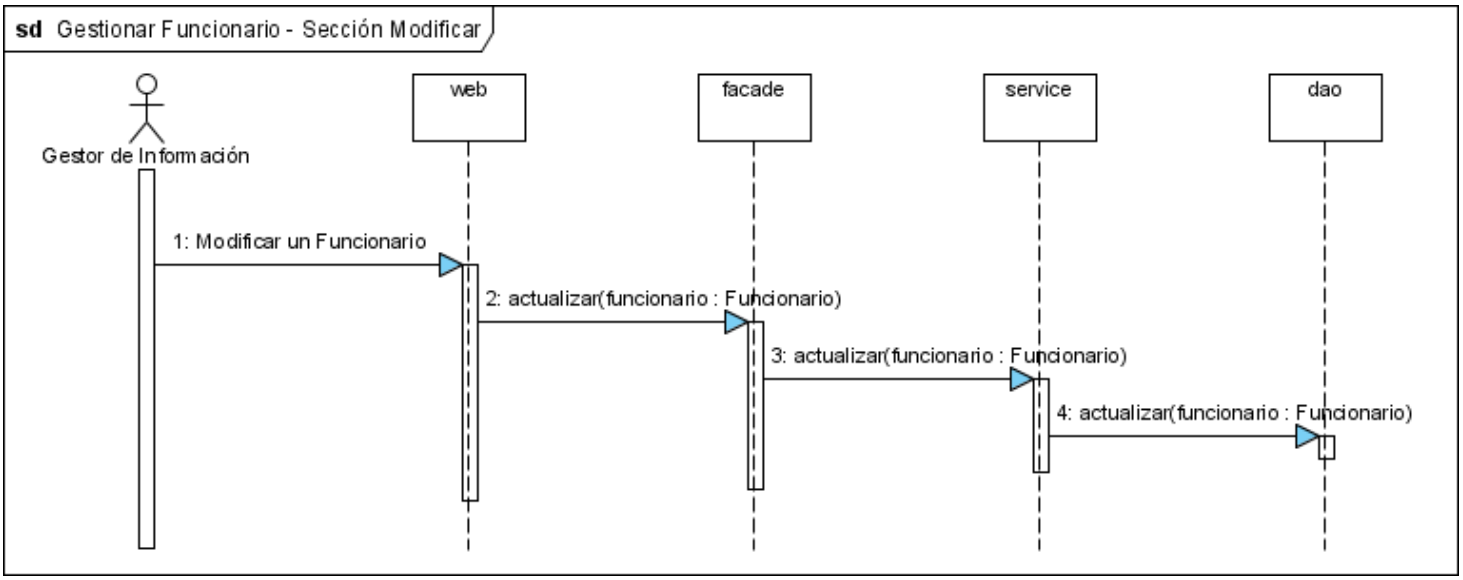

Fig 50. CUS Gestionar Funcionario – Sección Modificar.

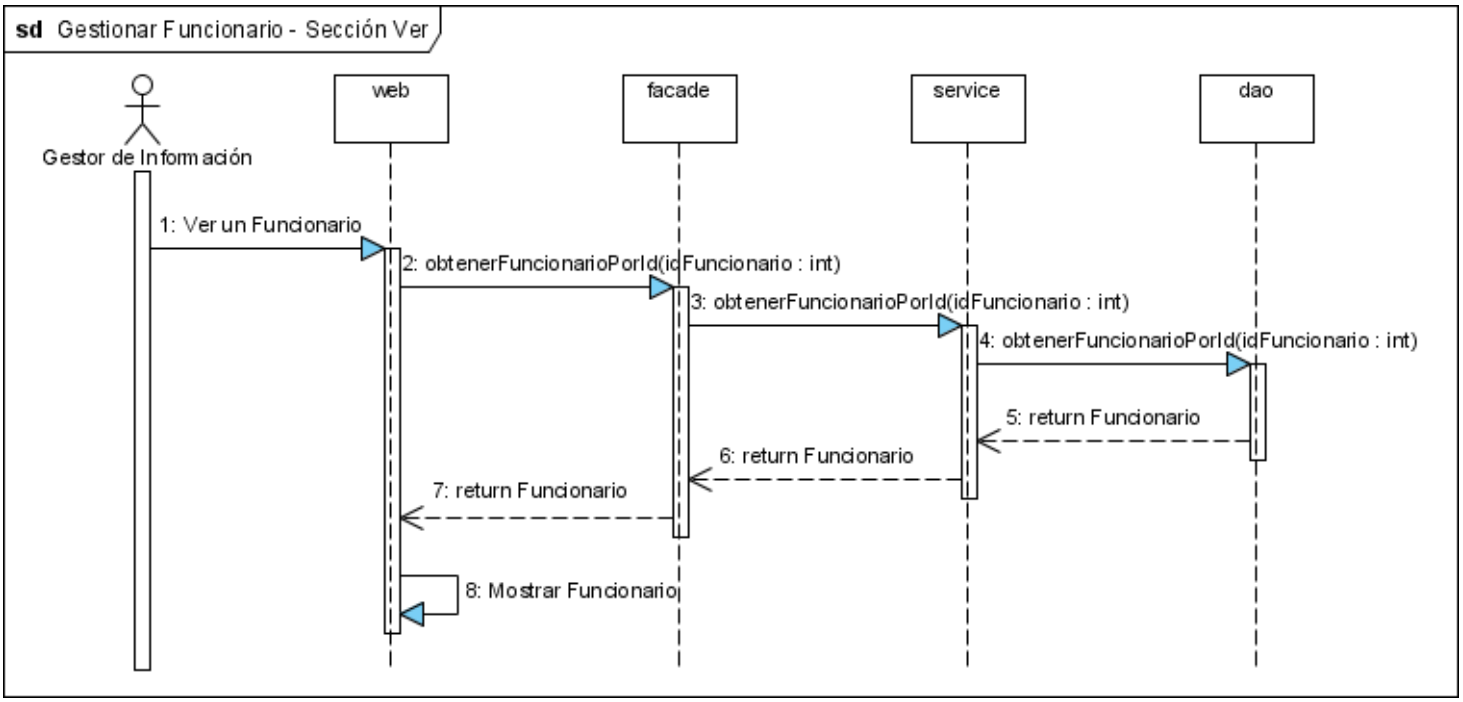

Fig 51. CUS Gestionar Funcionario – Sección Ver.

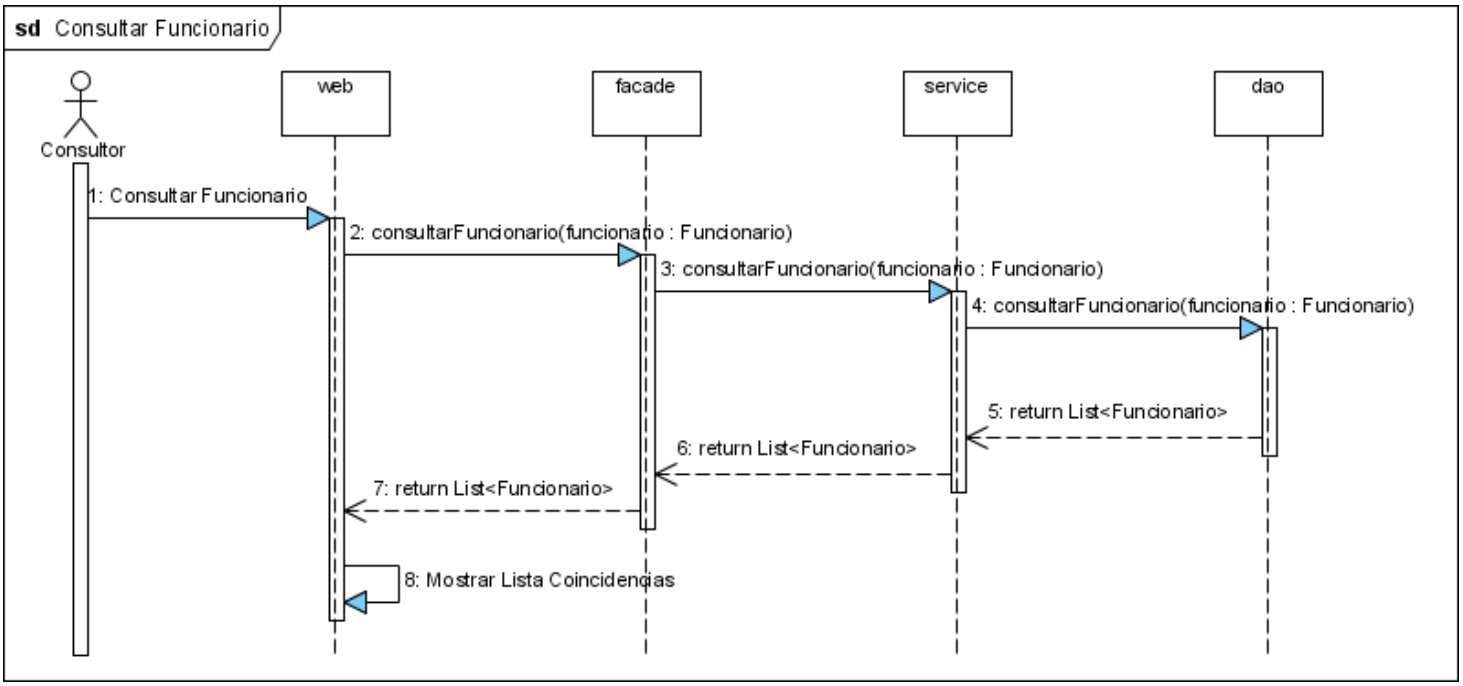

Fig 52. CUS Consultar Funcionario.

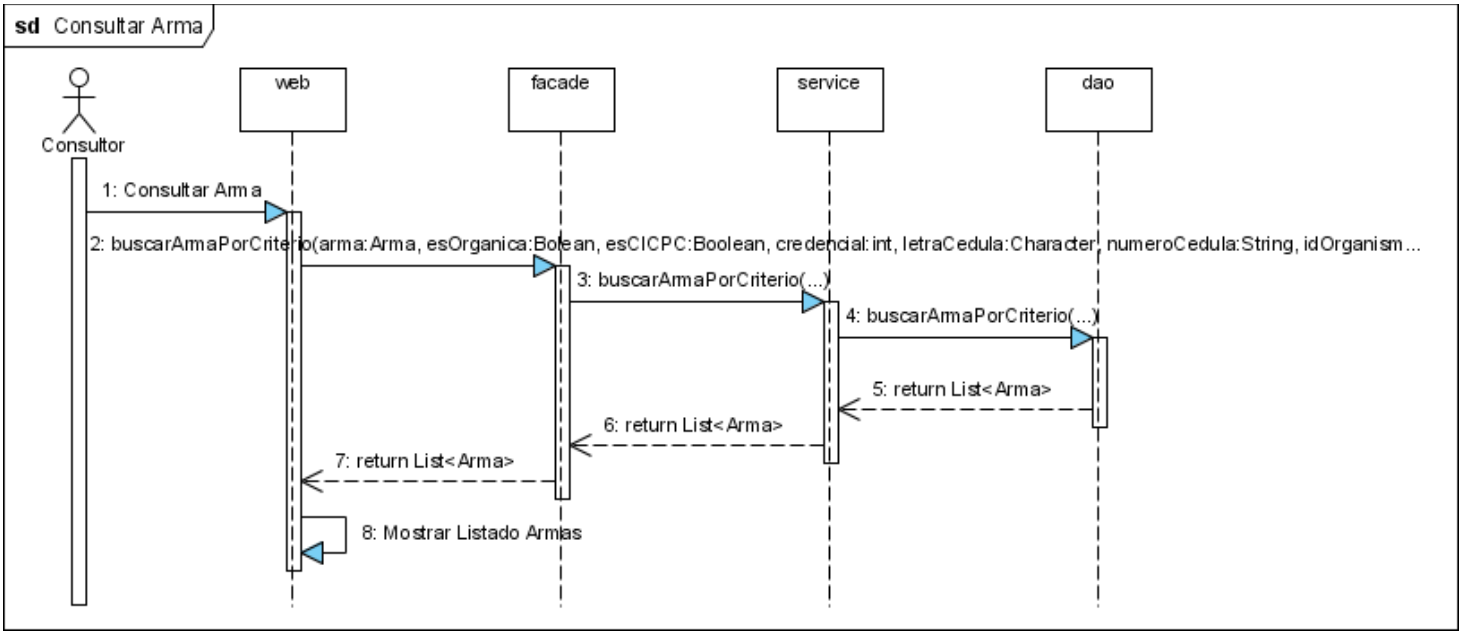

Fig 53. CUS Consultar Arma.

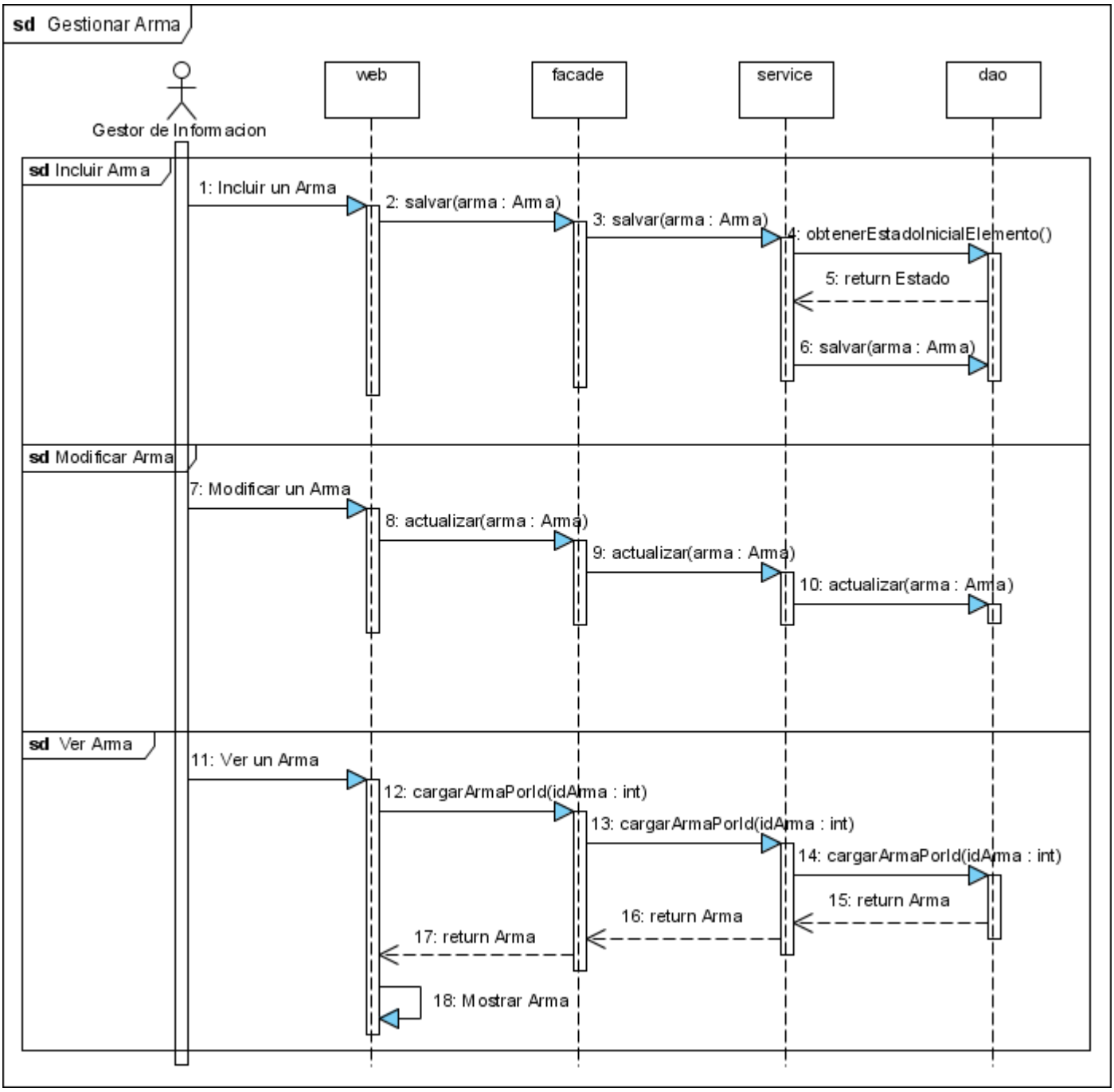

Fig 54. CUS Gestionar Arma.

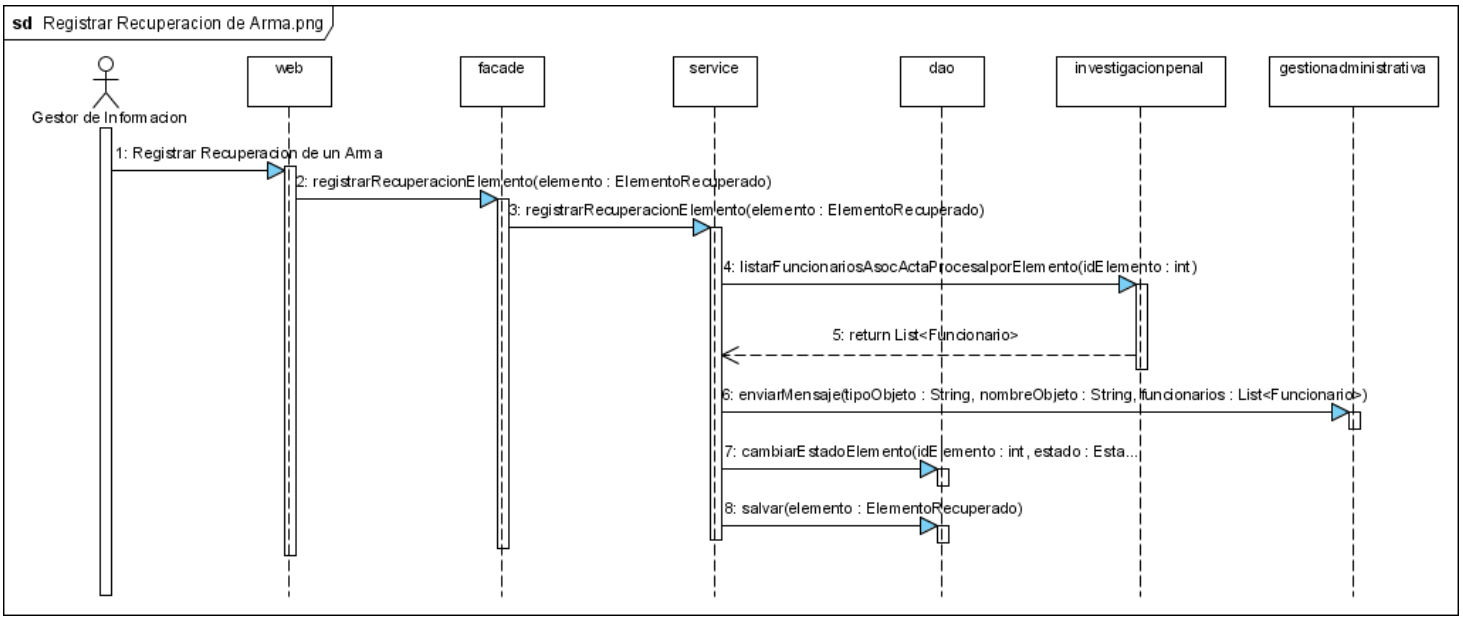

Fig 55. CUS Registrar Recuperación de Arma.

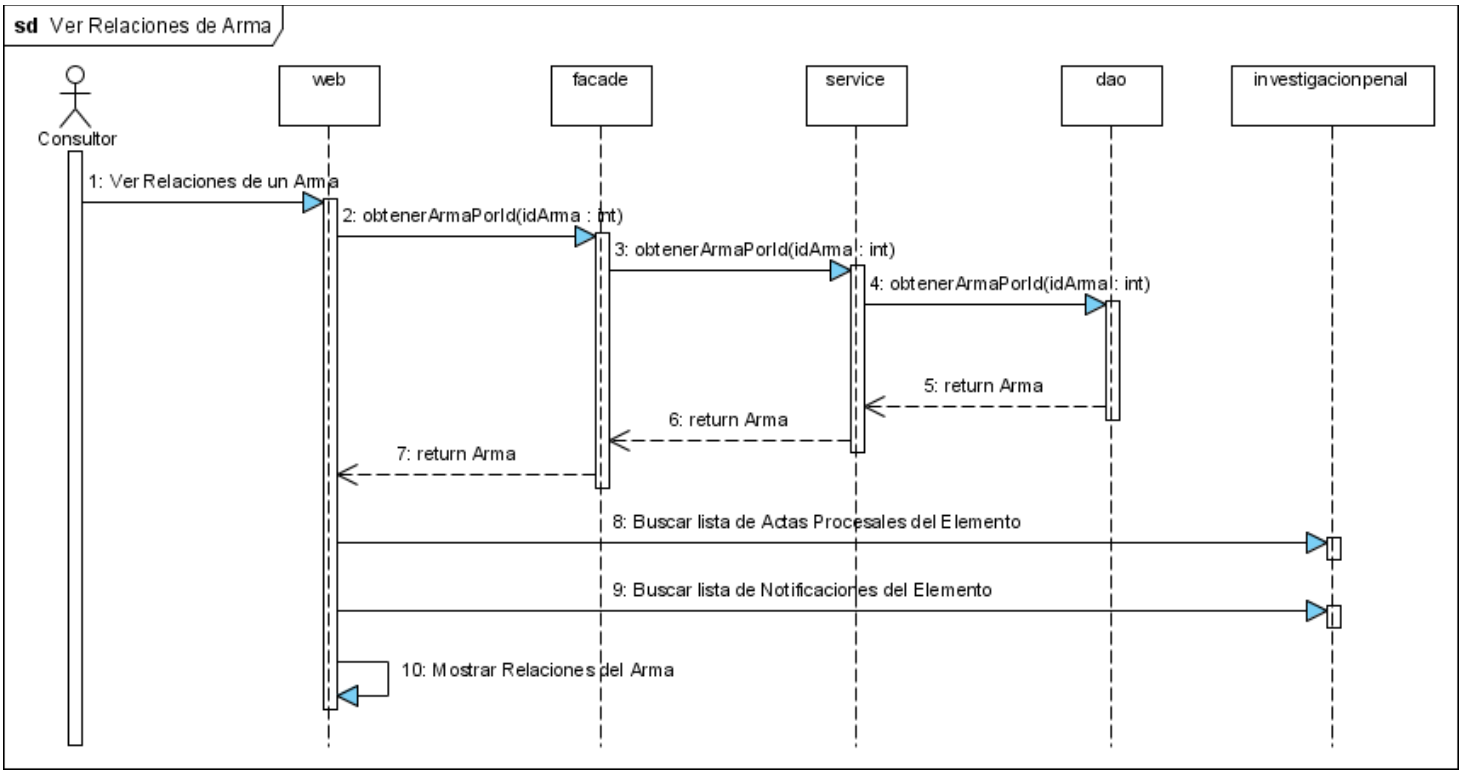

Fig 56. CUS Ver Relaciones de Arma.

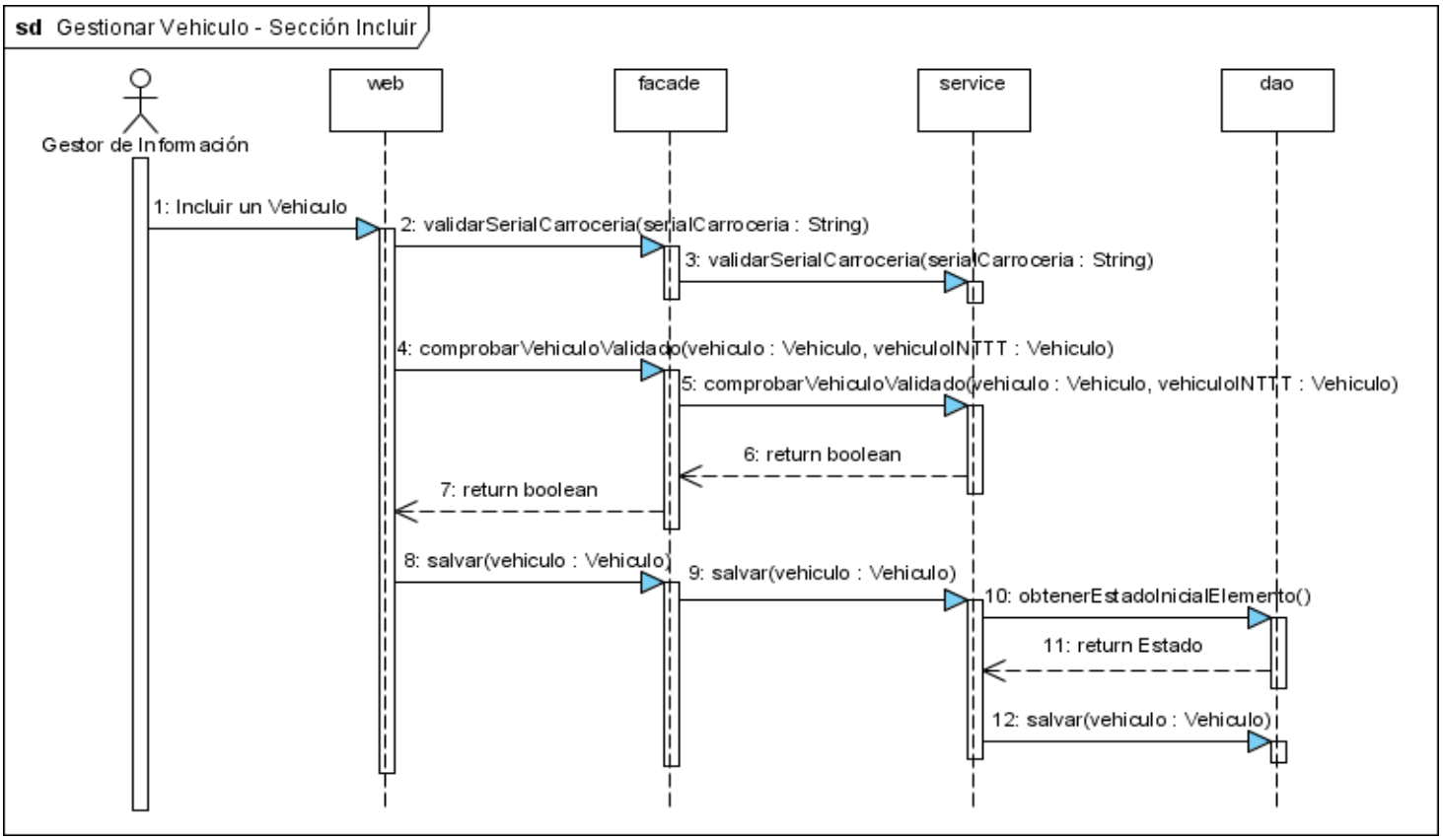

Fig 57. CUS Gestionar Vehículo – Sección Incluir.

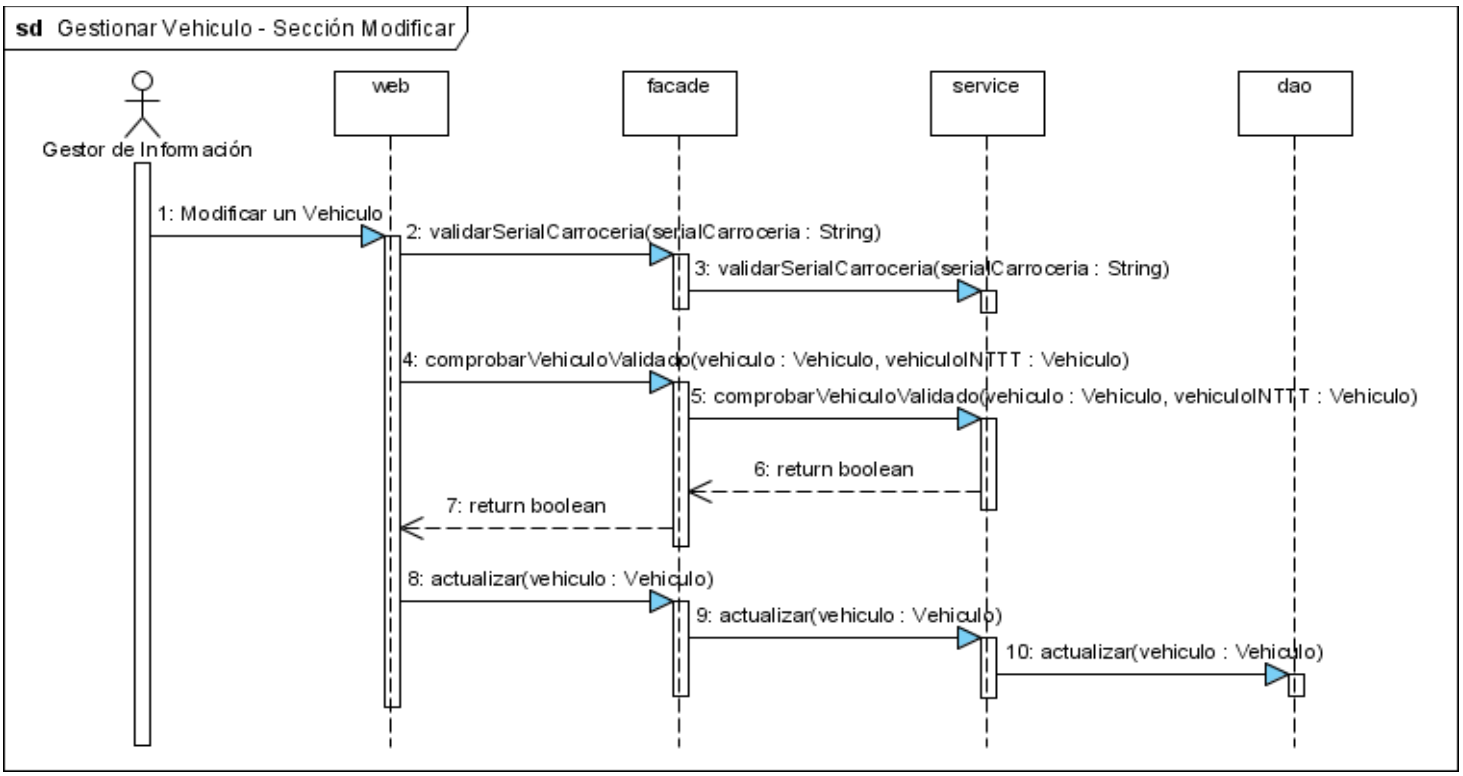

Fig 58. CUS Gestionar Vehículo – Sección Modificar.

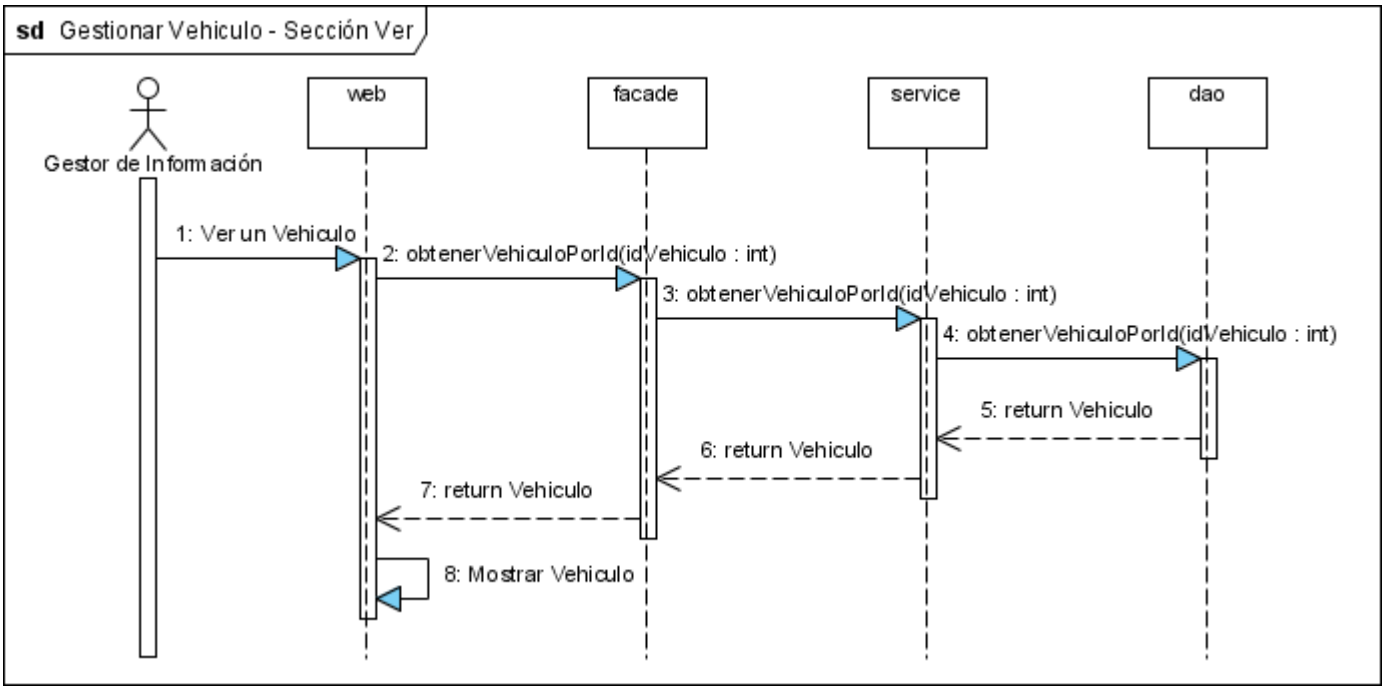

Fig 59. CUS Gestionar Vehículo – Sección Ver.

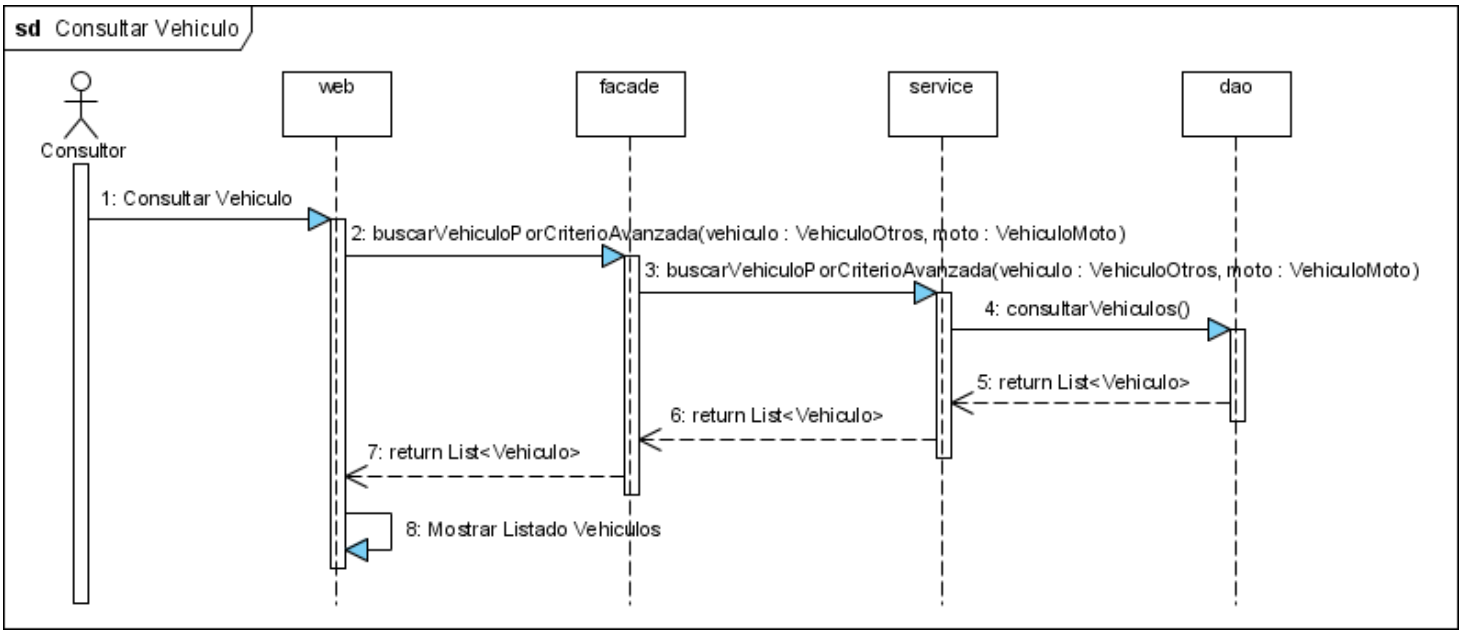

Fig 60. CUS Consultar Vehículo.

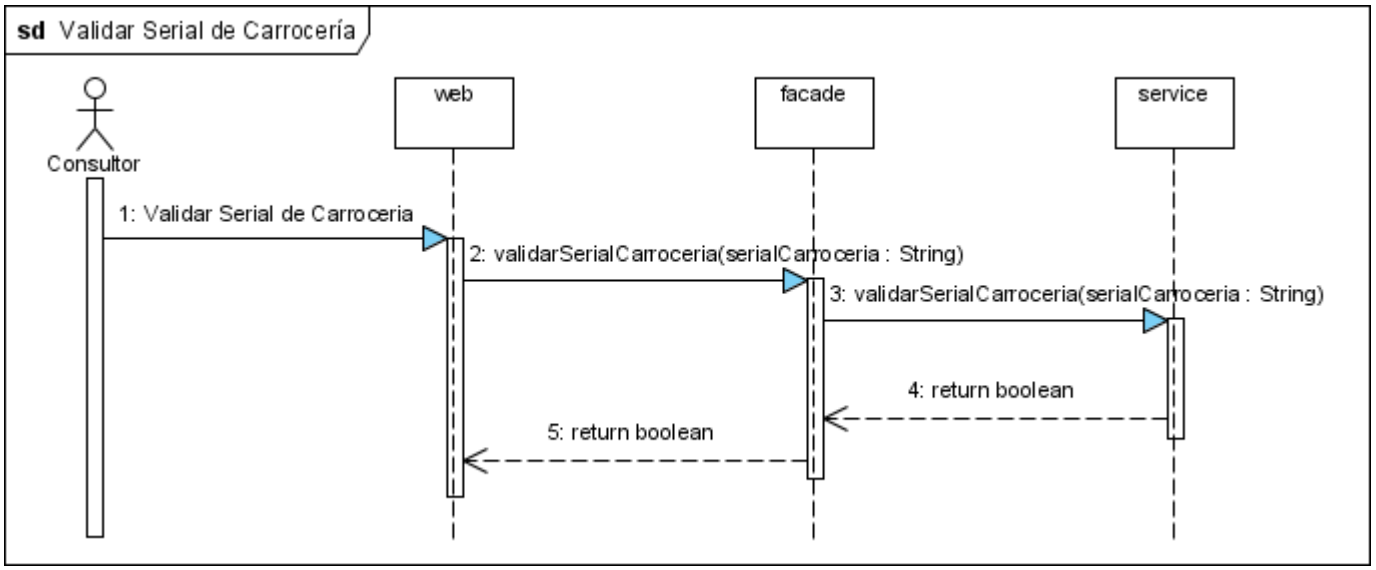

Fig 61. CUS Validar Serial de Carrocería.

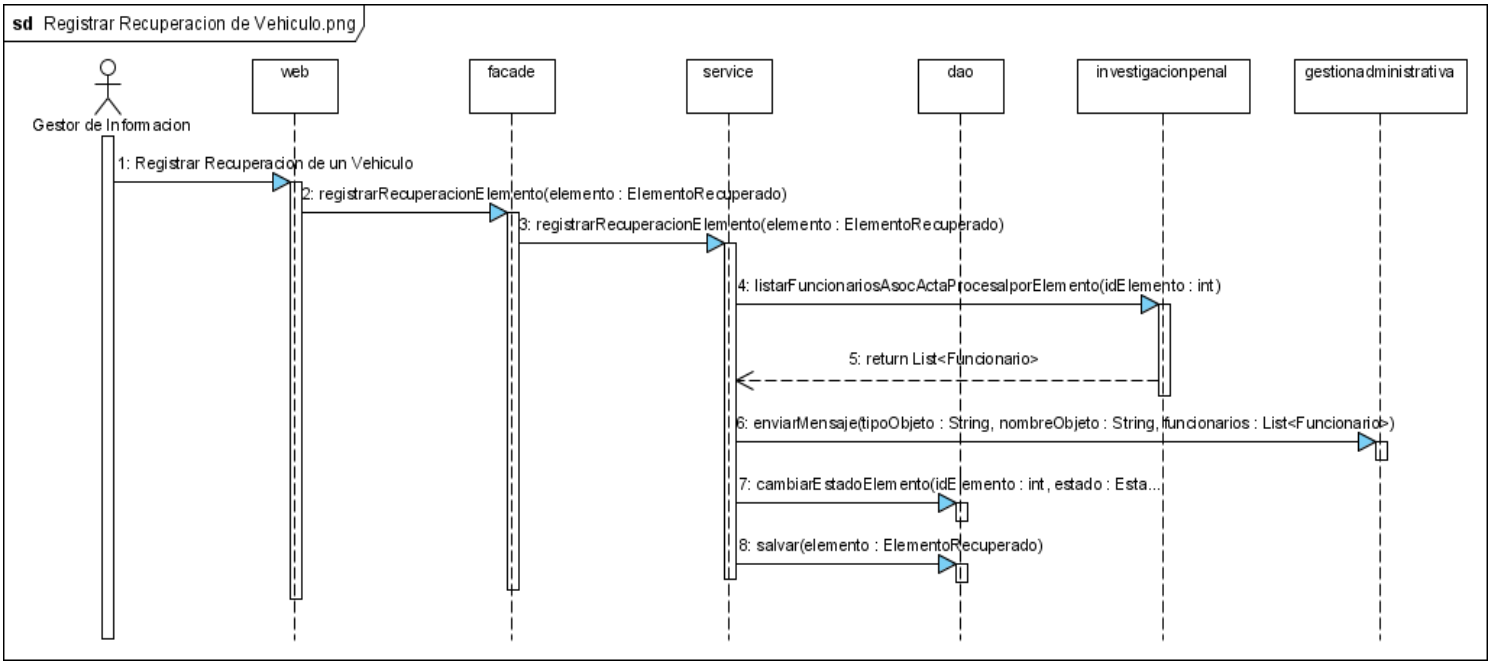

Fig 62. CUS Registrar Recuperación de Vehículo.

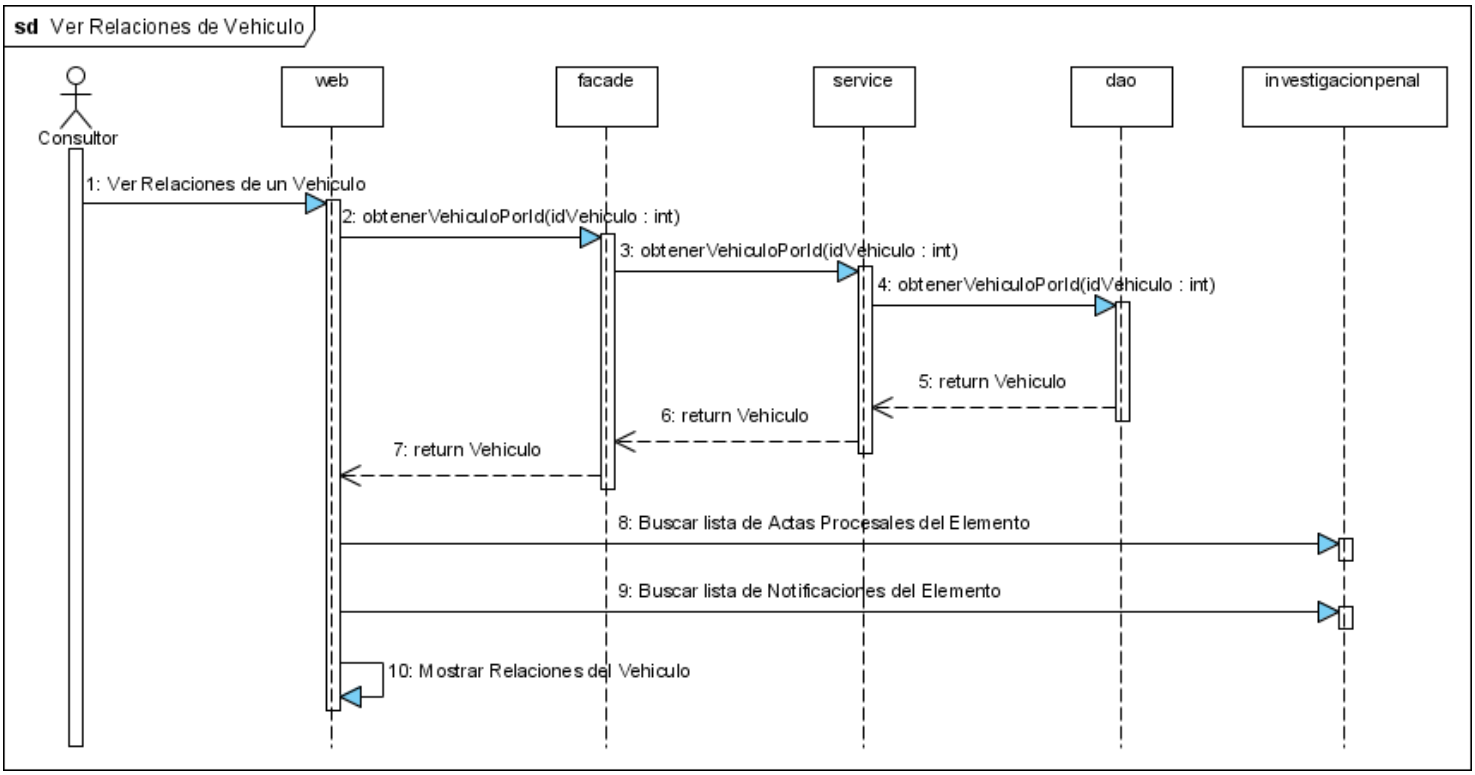

Fig 63. CUS Ver Relaciones de Vehículo.

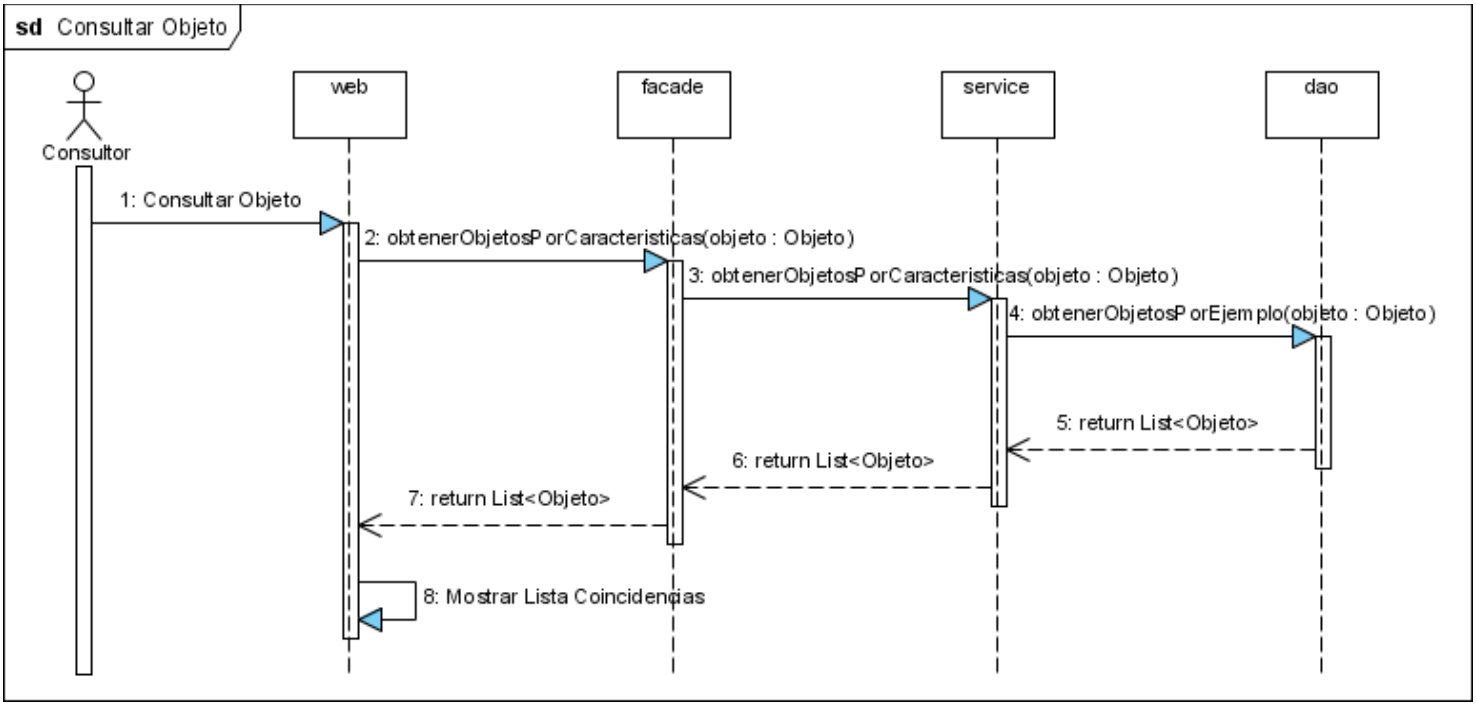

Fig 64. CUS Consultar Objeto.

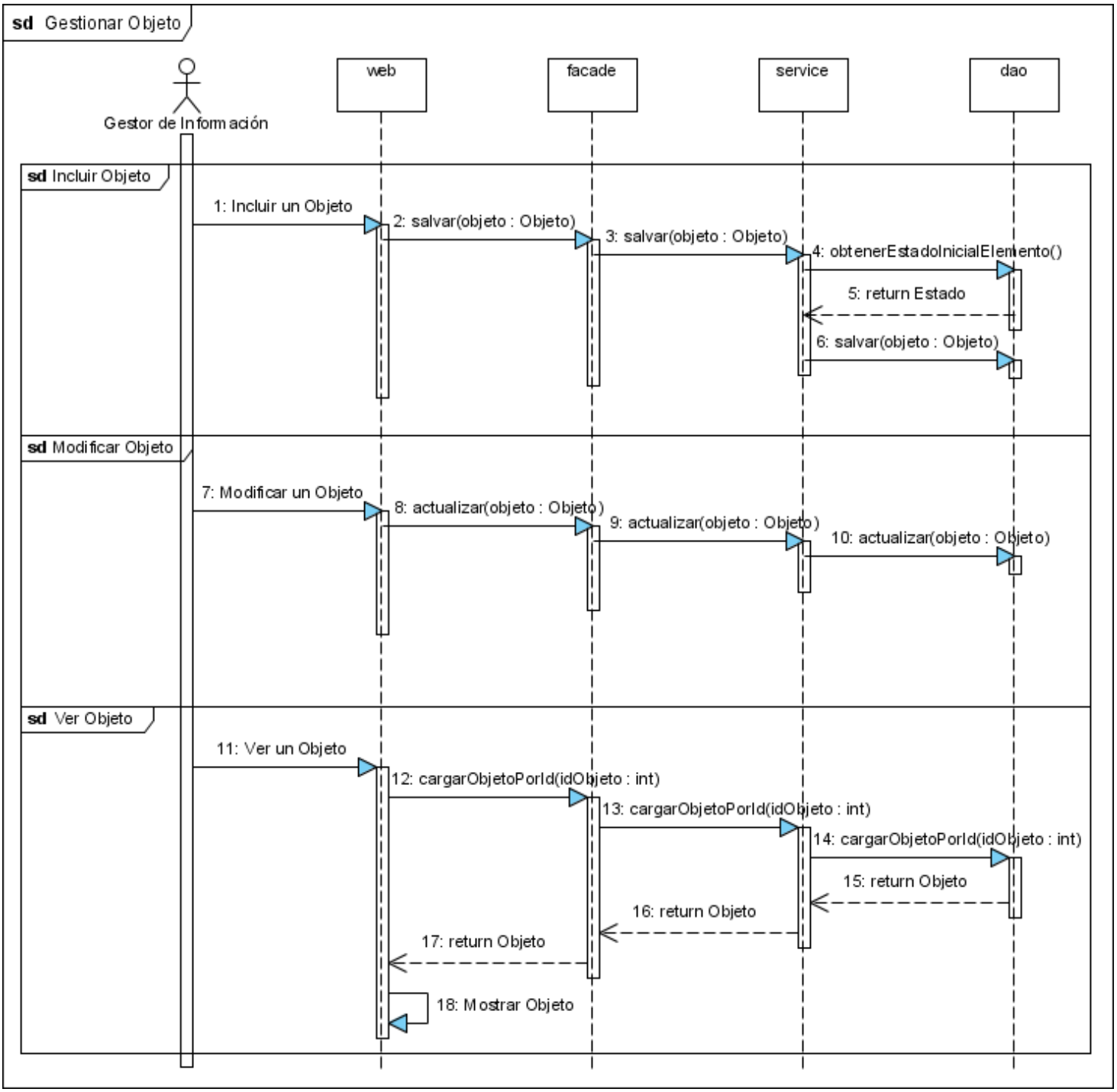

Fig 65. CUS Gestionar Objeto.

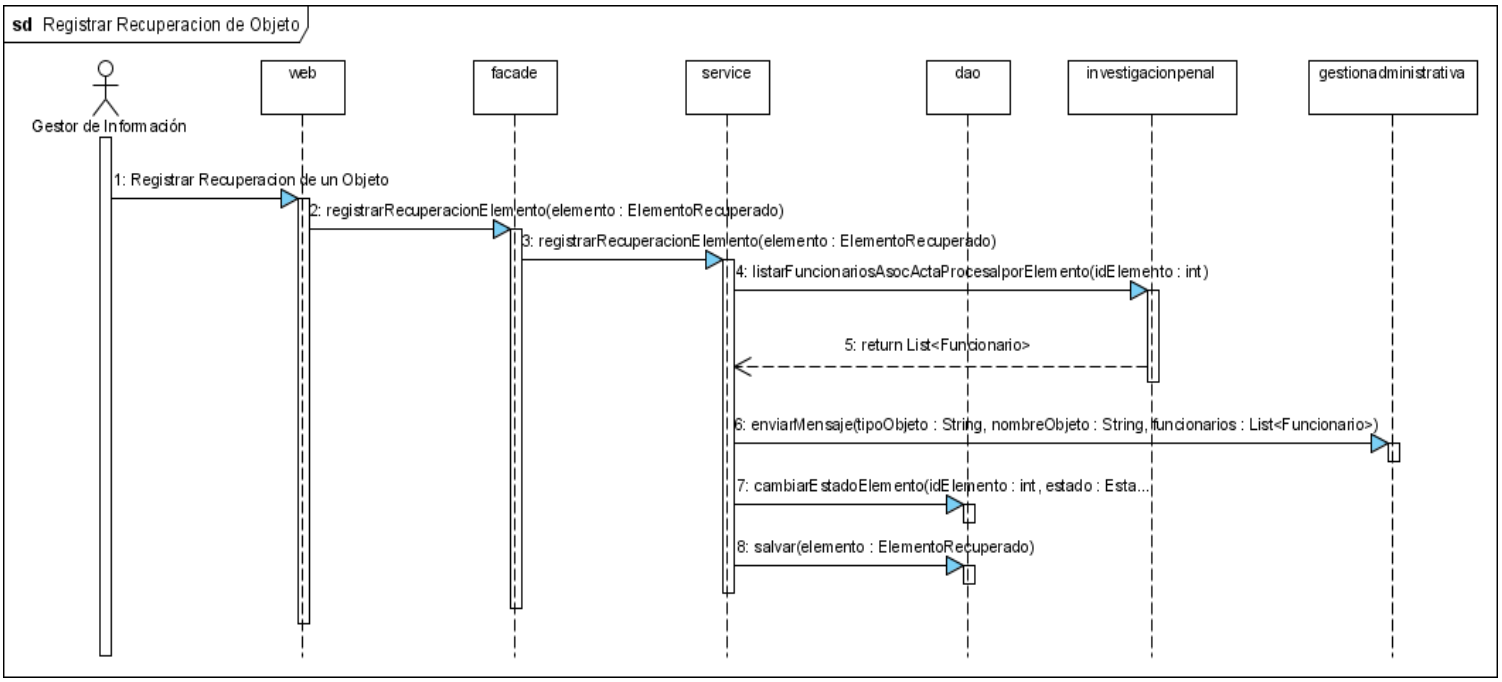

Fig 66. CUS Registrar Recuperación de Objeto.

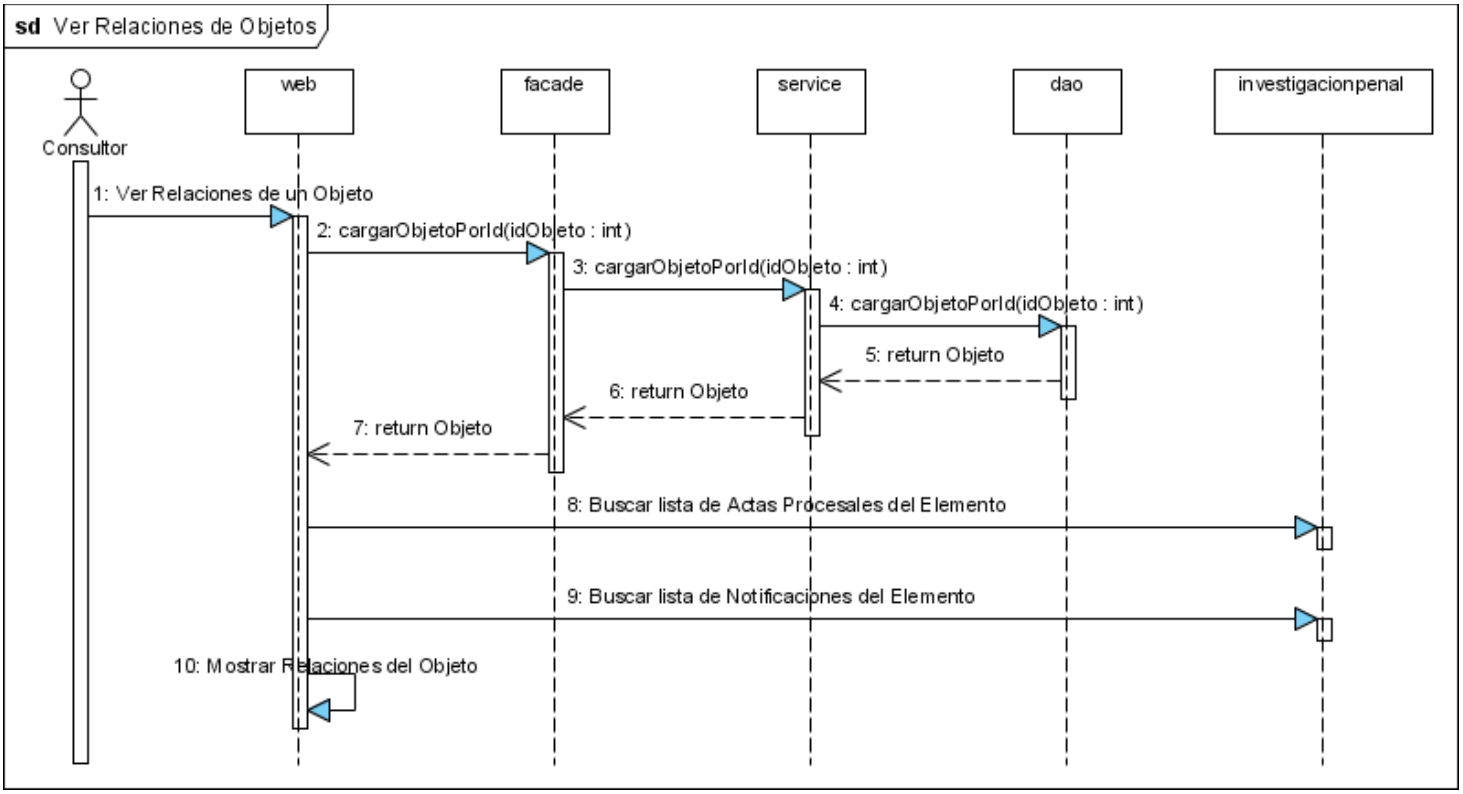

Fig 67. CUS Ver Relaciones de Objeto.

ANEXO 4: Descripción de las Clases del Diseño.

Nombre de la Clase:	FuncionarioFacadeImpl
Tipo de Clase:	Controladora
Atributo:	**Tipo:**
funcionarioService	FuncionarioService
Para Cada Responsabilidad:	
Nombre:	void salvarFuncionario(Funcionario funcionario)
Descripción:	*Esta función se utiliza para guardar un Funcionario*
Nombre:	void actualizarFuncionario(Funcionario funcionario)
Descripción:	*Esta función se utiliza para actualizar un Funcionario*
Nombre:	Arma obtenerFuncionarioPorId(Integer idFuncionario)
Descripción:	*Esta función se utiliza para cargar un funcionario por id*
Nombre:	Arma obtenerFuncionarioPorCredencial(String credencial)
Descripción:	*Esta función se utiliza para cargar un funcionario por su credencial*
Nombre:	Arma obtenerFuncionarioPorCedula (char letraCedula,String numeroCedula)
Descripción:	*Esta función se utiliza para cargar un funcionario por su letra y numero de cedula*
Nombre:	List<Funcionario> consultarFuncionarios(Funcionario funcionario)
Descripción:	*Esta función se utiliza para obtener todos los vehículos que cumplan con las características contenidas en el objeto pasado como parámetro.*
Nombre:	List<Funcionario> obtenerCoincidenciasFuncionarios(Character letraCedula, String numeroCedula, String credencial)
Descripción:	*Esta función se utiliza para obtener el listado de coincidencias sobre un funcionario*
Nombre:	void convertirPersonaAFuncionario(Funcionario funcionario, Persona persona)
Descripción:	*Esta función convierte una persona existente en el sistema en un funcionario*

Tabla 22. Descripción de la Clase FuncionarioFacadeImpl.

Nombre de la Clase:	FuncionarioServiceImpl
Tipo de Clase:	Controladora
Atributo:	**Tipo:**
funcionarioDao	FuncionarioDao
Carga	Carga
personaFacade	PersonaFacade
Para Cada Responsabilidad:	
Nombre:	void salvarFuncionario(Funcionario funcionario)
Descripción:	*Esta función se utiliza para guardar un Funcionario*
Nombre:	void actualizarFuncionario(Funcionario funcionario)
Descripción:	*Esta función se utiliza para actualizar un Funcionario*
Nombre:	Arma obtenerFuncionarioPorId(Integer idFuncionario)
Descripción:	*Esta función se utiliza para cargar un funcionario por id*
Nombre:	Arma obtenerFuncionarioPorCredencial(String credencial)
Descripción:	*Esta función se utiliza para cargar un funcionario por su credencial*

Nombre:	Arma obtenerFuncionarioPorCedula (char letraCedula,String numeroCedula)
Descripción:	*Esta función se utiliza para cargar un funcionario por su letra y numero de cedula*
Nombre:	List<Funcionario> consultarFuncionarios(Funcionario funcionario)
Descripción:	*Esta función se utiliza para obtener todos los vehículos que cumplan con las características contenidas en el objeto pasado como parámetro.*
Nombre:	List<Funcionario> obtenerCoincidenciasFuncionarios(Character letraCedula, String numeroCedula, String credencial)
Descripción:	*Esta función se utiliza para obtener el listado de coincidencias sobre un funcionario*
Nombre:	void convertirPersonaAFuncionario(Funcionario funcionario, Persona persona)
Descripción:	*Esta función convierte una persona existente en el sistema en un funcionario*

Tabla 23. Descripción de la Clase FuncionarioServiceImpl.

Nombre de la Clase:	ArmaFacadeImpl
Tipo de Clase:	Controladora
Atributo:	**Tipo:**
armaService	ArmaService
Para Cada Responsabilidad:	
Nombre:	void salvar(Arma arma)
Descripción:	*Esta función se utiliza para guardar un Arma*
Nombre:	void actualizar(Arma arma)
Descripción:	*Esta función se utiliza para actualizar un Arma*
Nombre:	Arma cargarArmaPorId(Integer idObjeto)
Descripción:	*Esta función se utiliza para cargar un arma por id*
Nombre:	List<Arma> consultarArmaPorSerial(Arma arma)
Descripción:	*Esta función se utiliza para listar todos las armas que tengan el tipo de arma y algún numero de serie según los datos de un arma*
Nombre:	List<Arma> buscarArmaPorCriterio(Arma arma, String letraCedula, String numeroCedula)
Descripción:	*Esta función se utiliza para realizar una búsqueda avanzada de una arma*
Nombre:	void eliminarArma(Arma arma,String justificacion)
Descripción:	*Esta función se utiliza para eliminar un Arma*

Tabla 24. Descripción de la Clase ArmaFacadeImpl.

Nombre de la Clase:	ArmaServiceImpl
Tipo de Clase:	Controladora
Atributo:	**Tipo:**
armaDao	ArmaDao
carga	Carga
elementoDao	ElementoDao
Para Cada Responsabilidad:	
Nombre:	void salvar(Arma arma)
Descripción:	*Esta función se utiliza para guardar un Arma*
Nombre:	void actualizar(Arma arma)
Descripción:	*Esta función se utiliza para actualizar un Arma*

Nombre:	Arma cargarArmaPorId(Integer idObjeto)
Descripción:	*Esta función se utiliza para cargar un arma por id*
Nombre:	List<Arma> consultarArmaPorSerial(Arma arma)
Descripción:	*Esta función se utiliza para listar todos las armas que tengan el tipo de arma y algún número de serie según los datos de un arma*
Nombre:	List<Arma> buscarArmaPorCriterio(Arma arma, String letraCedula, String numeroCedula)
Descripción:	*Esta función se utiliza para realizar una búsqueda avanzada de una arma*
Nombre:	void eliminarArma(Arma arma,String justificacion)
Descripción:	*Esta función se utiliza para eliminar un Arma*

Tabla 25. Descripción de la Clase ArmaServiceImpl.

Nombre de la Clase:	VehiculoFacadeImpl
Tipo de Clase:	Controladora
Atributo:	**Tipo:**
vehiculoService	VehiculoService
Para Cada Responsabilidad:	
Nombre:	void salvar(Vehiculo vehiculo)
Descripción:	*Esta función se utiliza para guardar un vehículo*
Nombre:	void actualizar(Vehiculo vehiculo)
Descripción:	*Esta función se utiliza para actualizar un Arma*
Nombre:	Arma obtenerVehiculoPorId(Integer idObjeto)
Descripción:	*Esta función se utiliza para cargar un vehículo por id*
Nombre:	List<Vehiculo> buscarVehiculoPorCriterioAvanzada(VehiculoOtros vehiculoOtros, VehiculoMoto vehiculoMoto)
Descripción:	*Este método se utiliza para realizar una búsqueda avanzada de vehículos*
Nombre:	List<Vehiculo> listarCoincidencias(boolean moto, String noChapa, String serialCarroceria, MarcaVehiculo marca, ModeloVehiculo modelo)
Descripción:	*Este método se utiliza para listar todos los vehículos que cumplan con las características pasadas por parámetro*
Nombre:	boolean comprobarVehiculoValidado(Vehiculo vehiculo, Vehiculo vehiculoINTTT)
Descripción:	*Este método se utiliza para comprobar si un vehículo continua validado o no por INTTT en dependencia de algunos parámetros como número de placa, serial de carrocería,etc.*
Nombre:	boolean validarSerialCarroceria(String serialCarroceria)
Descripción:	*Este método se utiliza para validar el serial de una carrocería*
Nombre:	void eliminar(Vehiculo vehiculo, String justificacion)
Descripción:	*Esta función se utiliza para eliminar un vehículo*

Tabla 26. Descripción de la Clase VehiculoFacadeImpl.

Nombre de la Clase:	VehiculoServiceImpl
Tipo de Clase:	Controladora
Atributo:	**Tipo:**
vehiculoDao	VehiculoDao
carga	Carga

elementoDao	ElementoDao
Para Cada Responsabilidad:	
Nombre:	void salvar(Vehiculo vehiculo)
Descripción:	*Esta función se utiliza para guardar un vehículo*
Nombre:	void actualizar(Vehiculo vehiculo)
Descripción:	*Esta función se utiliza para actualizar un Arma*
Nombre:	Arma obtenerVehiculoPorId(Integer idObjeto)
Descripción:	*Esta función se utiliza para cargar un vehículo por id*
Nombre:	List<Vehiculo> buscarVehiculoPorCriterioAvanzada(VehiculoOtros vehiculoOtros, VehiculoMoto vehiculoMoto)
Descripción:	*Este método se utiliza para realizar una búsqueda avanzada de vehículos*
Nombre:	List<Vehiculo> listarCoincidencias(boolean moto, String noChapa, String serialCarroceria, MarcaVehiculo marca, ModeloVehiculo modelo)
Descripción:	*Este método se utiliza para listar todos los vehículos que cumplan con las características pasadas por parámetro*
Nombre:	boolean comprobarVehiculoValidado(Vehiculo vehiculo, Vehiculo vehiculoINTTT)
Descripción:	*Este método se utiliza para comprobar si un vehículo continua validado o no por INTTT en dependencia de algunos parámetros como número de placa, serial de carrocería,etc.*
Nombre:	boolean validarSerialCarroceria(String serialCarroceria)
Descripción:	*Este método se utiliza para validar el serial de una carrocería*
Nombre:	void eliminar(Vehiculo vehiculo, String justificacion)
Descripción:	*Esta función se utiliza para eliminar un vehículo*

Tabla 27. Descripción de la Clase VehiculoServiceImpl.

Nombre de la Clase:	ObjetoFacadeImpl
Tipo de Clase:	Controladora
Atributo:	**Tipo:**
objetoService	ObjetoService
Para Cada Responsabilidad:	
Nombre:	void salvar(Objeto objeto)
Descripción:	*Esta función se utiliza para salvar un Objeto*
Nombre:	void actualizar(Objeto objeto)
Descripción:	*Esta función se utiliza para actualizar un Objeto*
Nombre:	Objeto cargarObjetoPorId(Integer idObjeto)
Descripción:	*Esta función se utiliza para cargar un objeto por id*
Nombre:	List<Objeto> obtenerObjetoByExample(Objeto objeto)
Descripción:	*Esta función se utiliza para cargar una lista de objeto que cumplan con una determinada cantidad de parámetros*
Nombre:	List<Objeto> obtenerObjetosPorCaracteristicas(Objeto objeto)
Descripción:	*Esta función se utiliza para obtener todos los objetos que cumplan con las características contenidas en el objeto pasado como parámetro.*
Nombre:	void eliminar(Objeto objeto,String justificacion)
Descripción:	*Esta función se utiliza para eliminar un Objeto*

Tabla 28. Descripción de la Clase ObjetoFacadeImpl.

Nombre de la Clase:	ObjetoServiceImpl
Tipo de Clase:	Controladora
Atributo:	**Tipo:**
objetoDao	ObjetoDao
carga	Carga
elementoDao	ElementoDao
Para Cada Responsabilidad:	
Nombre:	void salvar(Objeto objeto)
Descripción:	*Esta función se utiliza para guardar un Objeto*
Nombre:	void actualizar(Objeto objeto)
Descripción:	*Esta función se utiliza para actualizar un Objeto*
Nombre:	Objeto cargarObjetoPorId(Integer idObjeto)
Descripción:	*Esta función se utiliza para cargar un objeto por id*
Nombre:	List<Objeto> obtenerObjetoByExample(Objeto objeto)
Descripción:	*Esta función se utiliza para cargar una lista de objeto que cumplan con una determinada cantidad de parámetros*
Nombre:	List<Objeto> obtenerObjetosPorCaracteristicas(Objeto objeto)
Descripción:	*Esta función se utiliza para obtener todos los objetos que cumplan con las características contenidas en el objeto pasado como parámetro.*
Nombre:	void eliminar(Objeto objeto,String justificacion)
Descripción:	*Esta función se utiliza para eliminar un Objeto*

Tabla 29. Descripción de la Clase ObjetoServiceImpl.

ANEXO 5: Diagramas de Componentes.

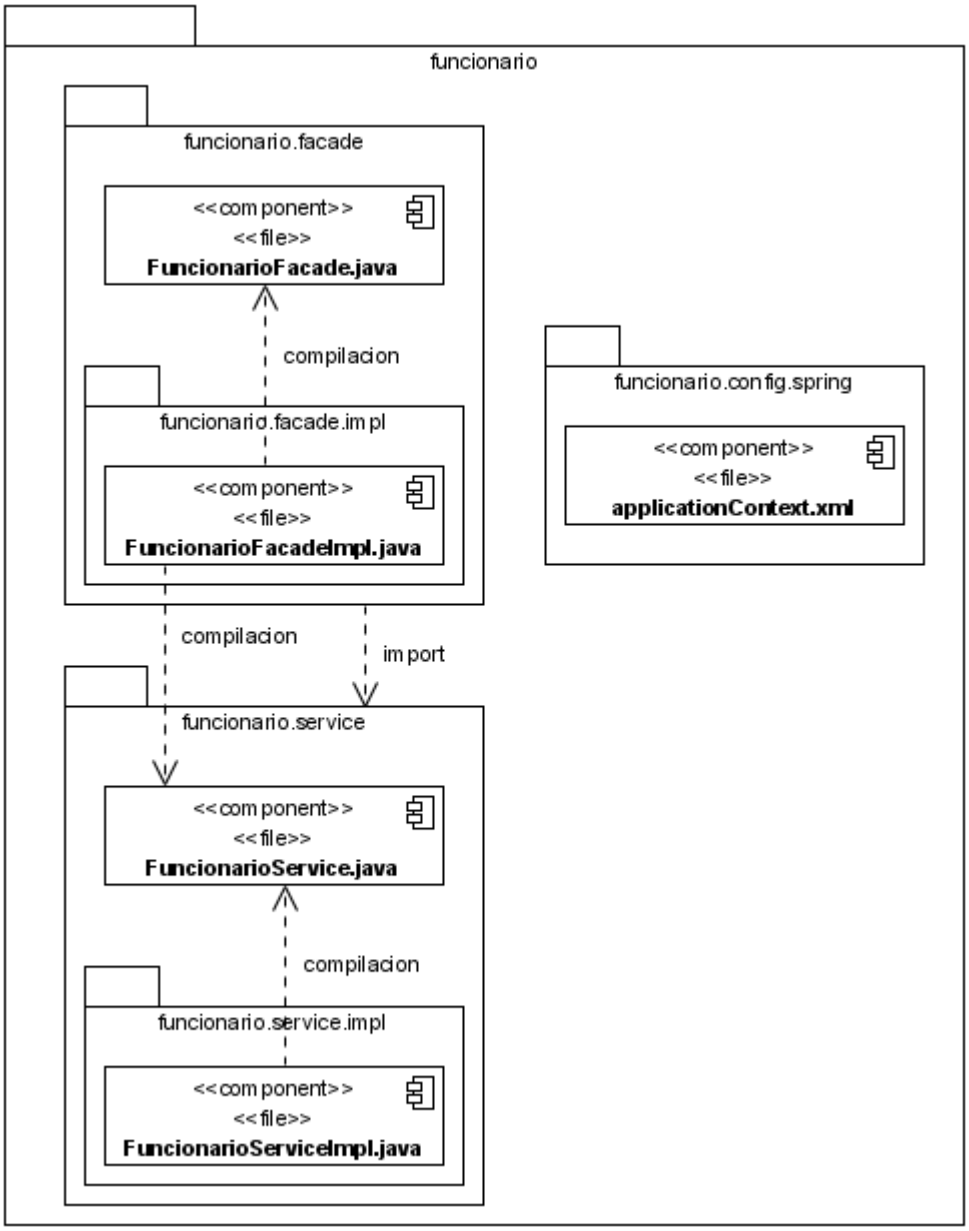

Fig 68. Sub-módulo Funcionario.

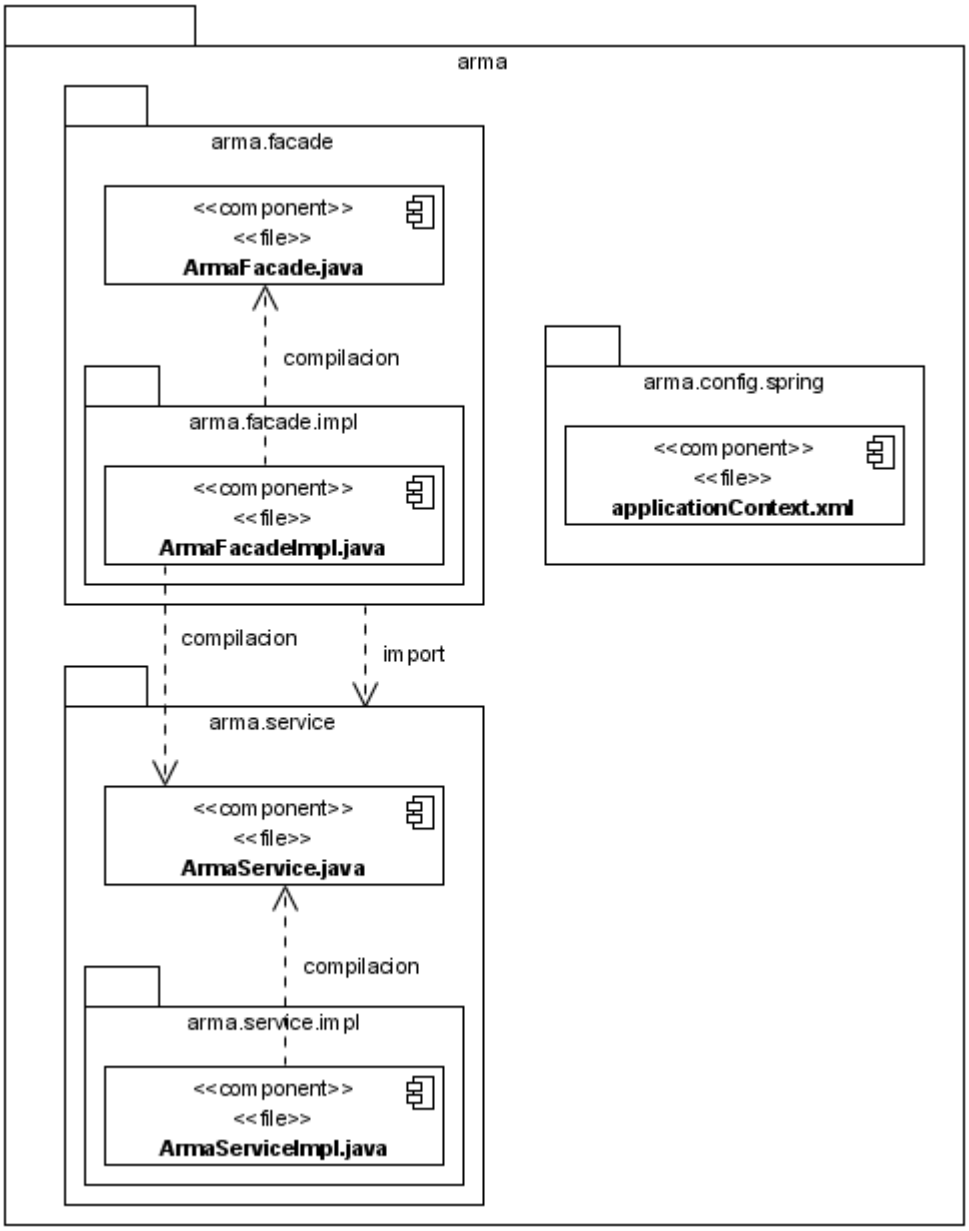

Fig 69. Sub-módulo Arma.

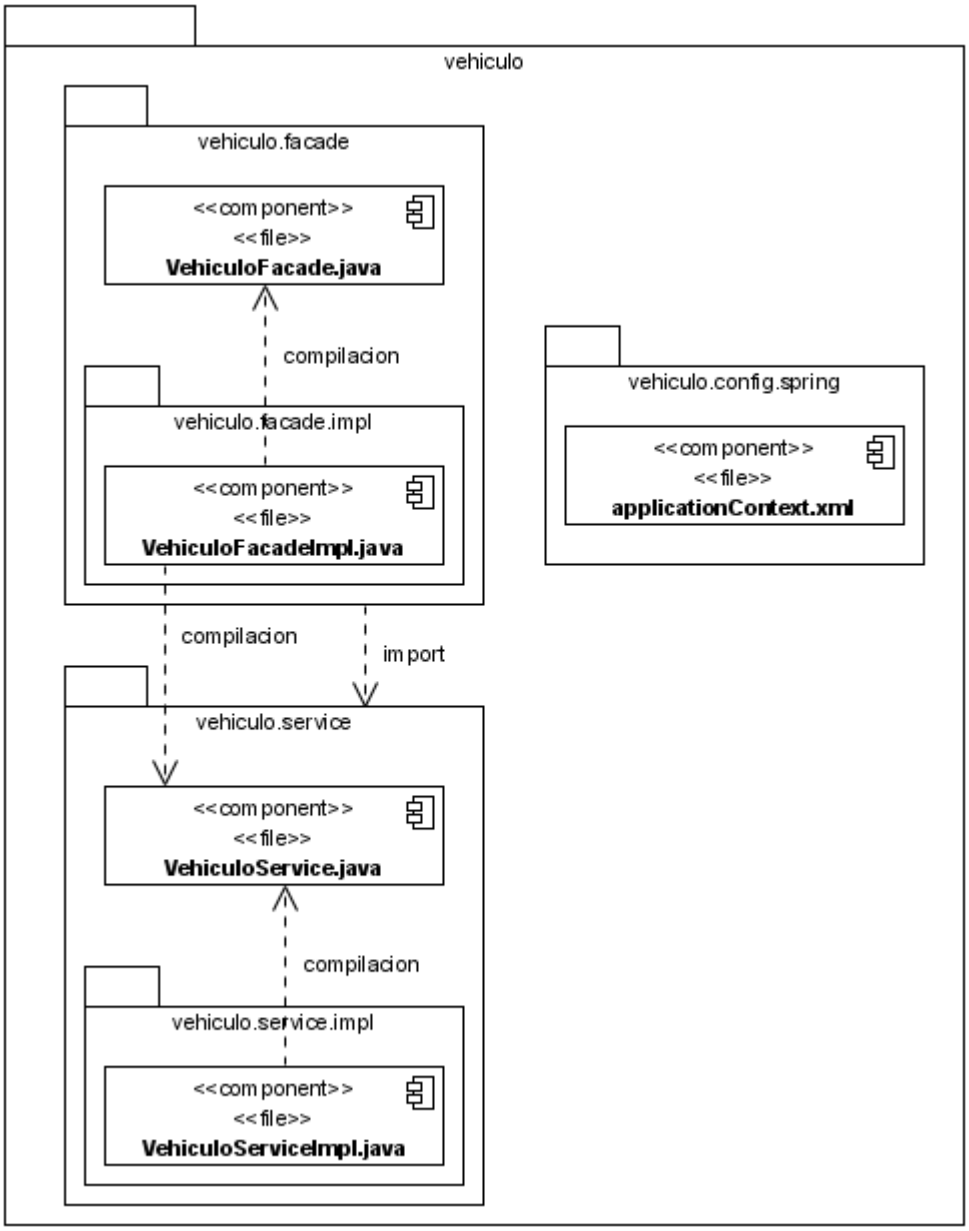

Fig 70. Sub-módulo Vehículo.

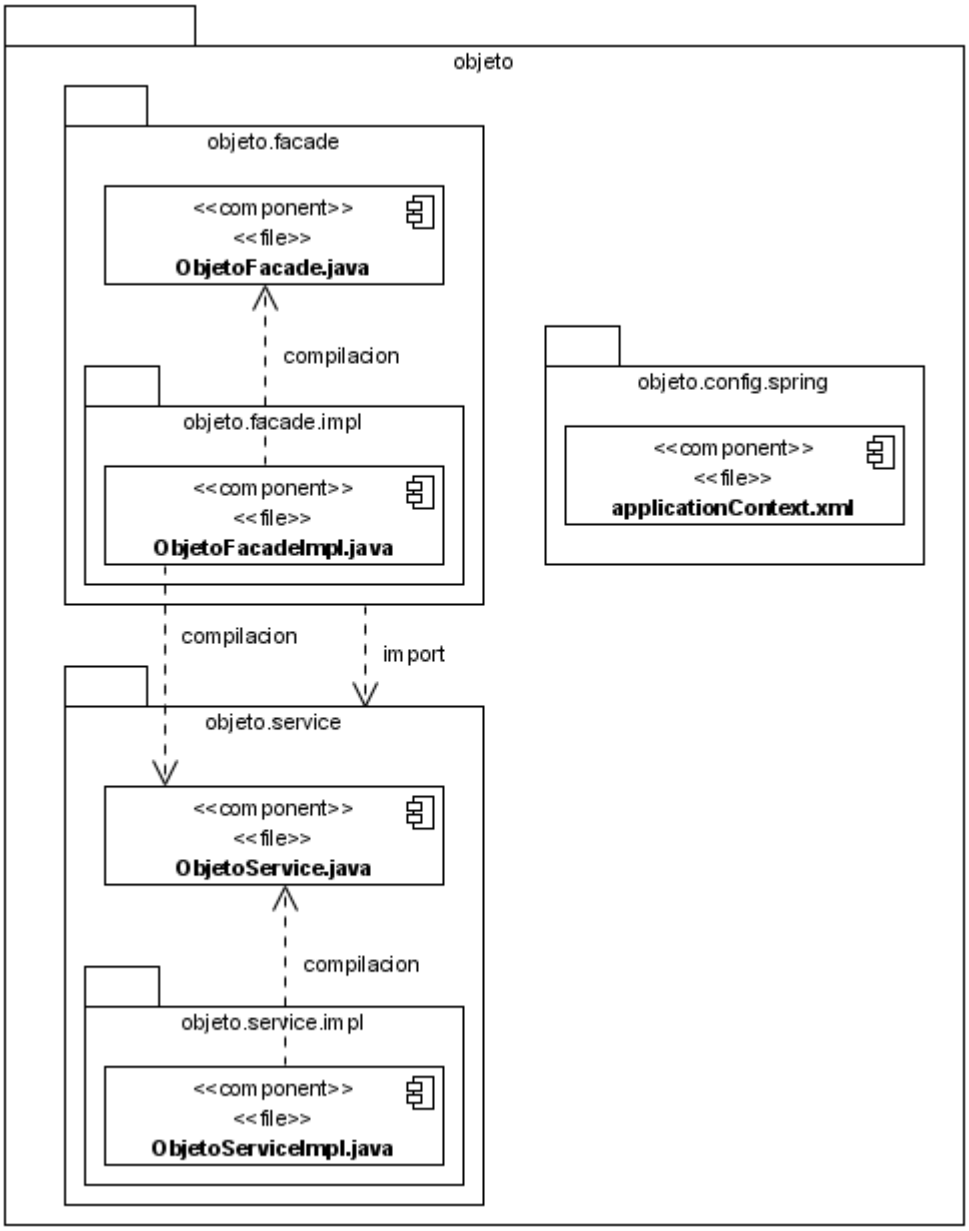

Fig 71. Sub-módulo Objeto.

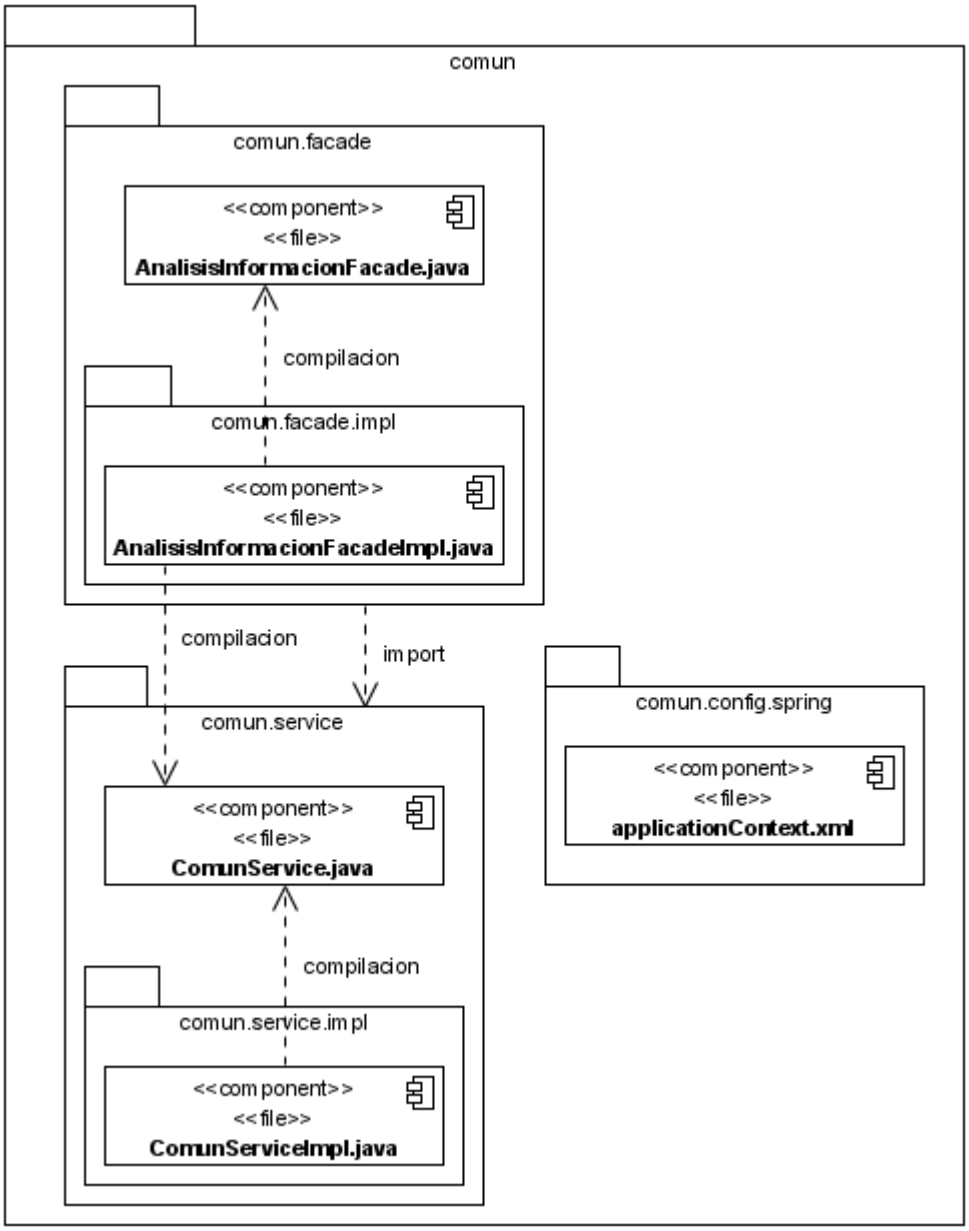

Fig 72. Paquete Común.

ANEXO 6: Uso de Nomencladores.

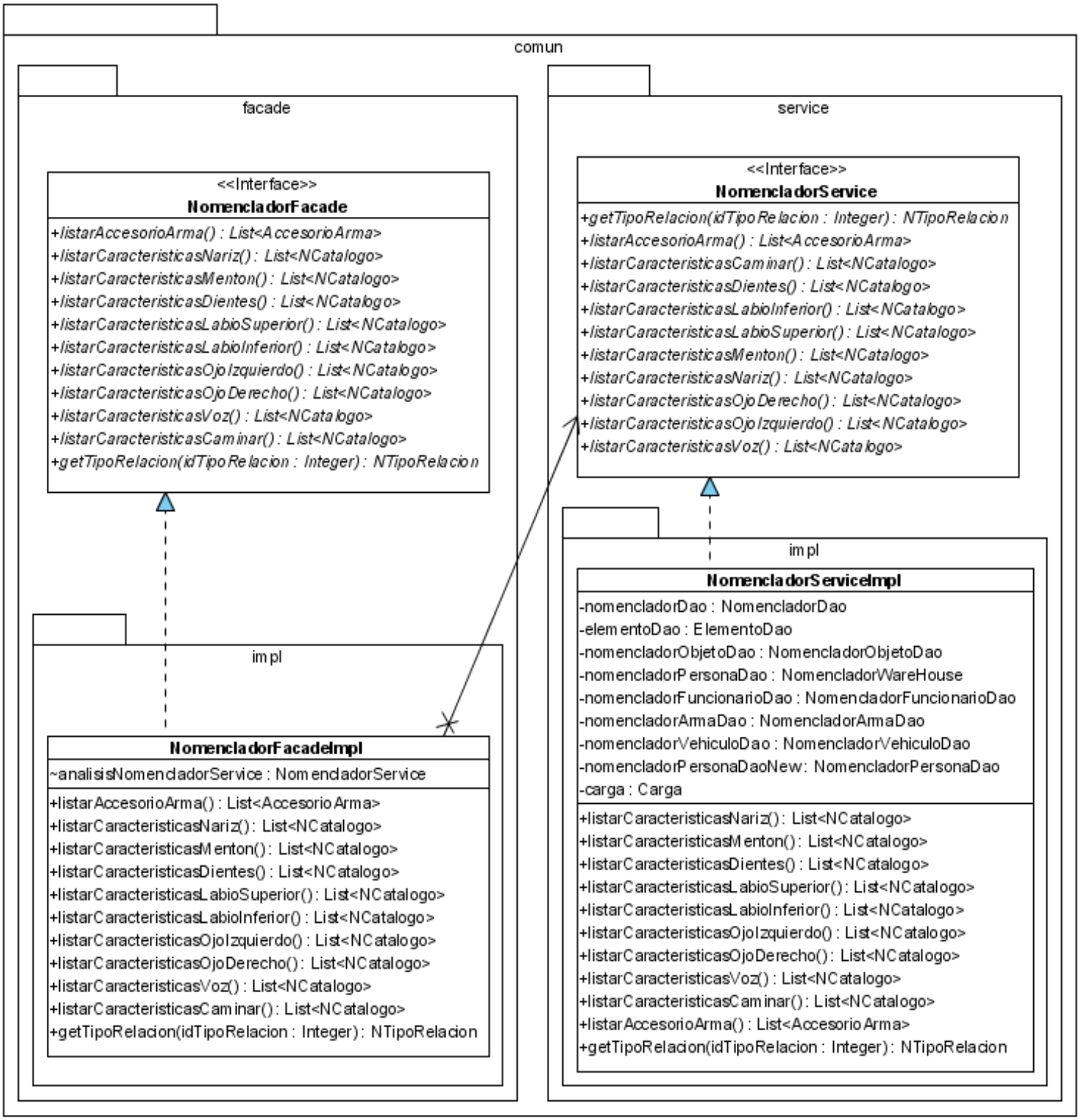

Fig 73. Fragmento del diagrama de Clases del Diseño sobre el uso de Nomencladores.

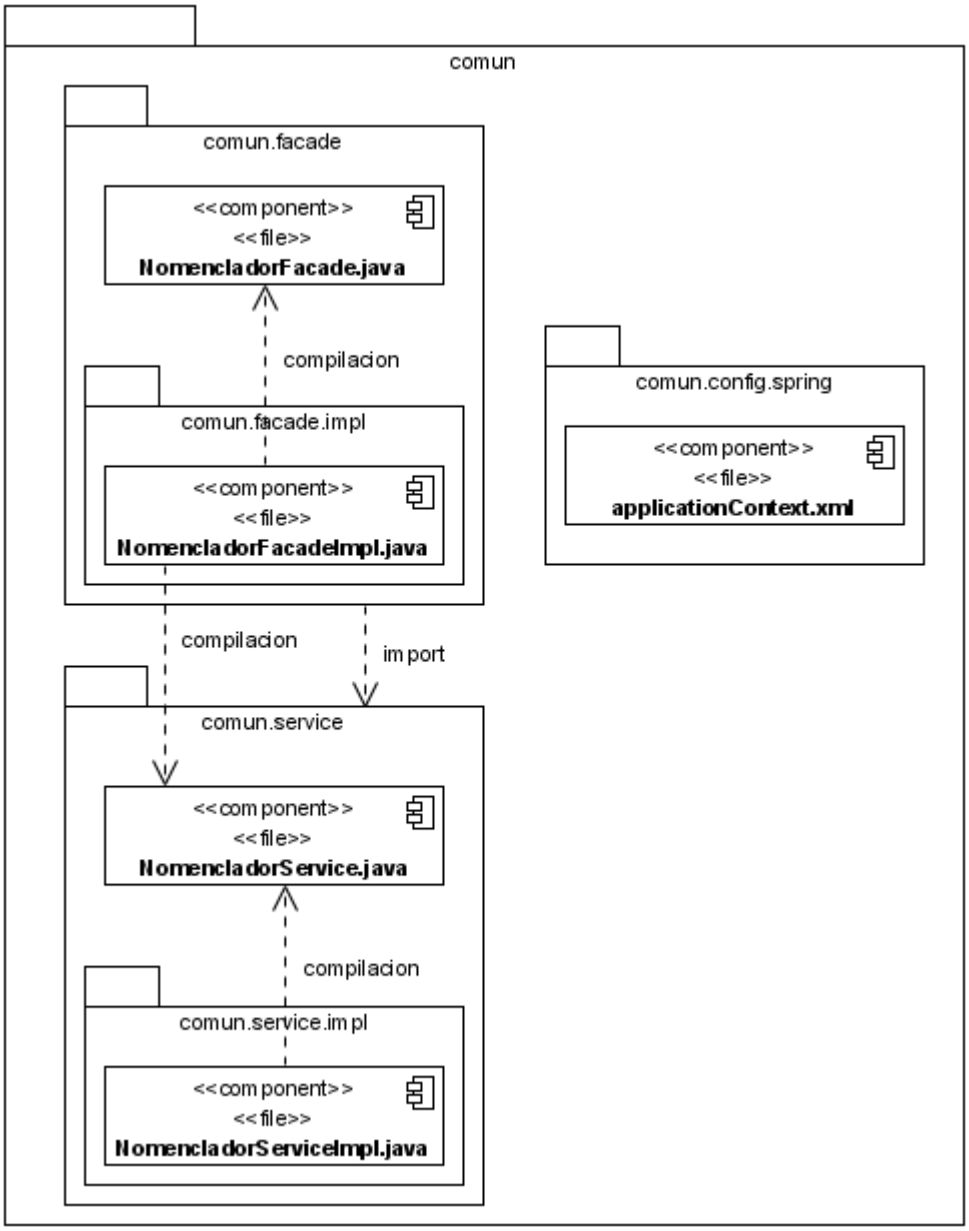

Fig 74. Diagrama de Componentes sobre el uso de Nomencladores.

Nombre de la Clase:	NomencladorFacadeImpl
Tipo de Clase:	Controladora
Atributo:	**Tipo:**
analisisNomencladorService	NomencladorService
Para Cada Responsabilidad:	
Nombre:	List<ColorObjeto> listarColor()
Descripción:	*Esta función se utiliza para listar todos los colores posibles*
Nombre:	List<Estado> listarEstados()
Descripción:	*Esta función se utiliza para listar todas los estados que puede tener un objeto*
Nombre:	List<EstadoMateriaObjeto> listarEstadoMateria()
Descripción:	*Esta función se utiliza para listar todos los estados de materia que puede tener un objeto*
Nombre:	List<FormaObjeto> listarForma()
Descripción:	*Esta función se utiliza para listar todos los formas que puede tener un objeto*
Nombre:	List<GrupoObjeto> listarGrupo()
Descripción:	*Esta función se utiliza para listar todos los grupos que puede tener un objeto*
Nombre:	List<UMCantidad> listarUnidadMedidaCantidad()
Descripción:	*Esta función se utiliza para listar todos las unidades de medidas de cantidad que puede tener un objeto*
Nombre:	List<UMPeso> listarUnidadMedidaPeso()
Descripción:	*Esta función se utiliza para listar todos las unidades de medidas del peso que puede tener un objeto*
Nombre:	List<UMTamanno> listarUnidadMedidaTamanno()
Descripción:	*Esta función se utiliza para listar todos las unidades de medidas del tamaño que puede tener un objeto*
Nombre:	List<UMValor> listarUnidadMedidaValor()
Descripción:	*Esta función se utiliza para listar todos las unidades de medida del valor que puede tener un objeto*
Nombre:	Estado obtenerEstadoInicialElemento()
Descripción:	*Esta función retorna el estado inicial de un Elemento*
Nombre:	List<AccesorioArma> listarAccesorioArma()
Descripción:	*La función retorna la lista de todos los posibles accesorios que puede poseer un arma*
Nombre:	List<MarcaArma> listarMarcaArma()
Descripción:	*Esta función se utiliza para listar todas las marcas que puede poseer un arma*
Nombre:	List<MarcaArma> listarMarcaArmaConModelos()
Descripción:	*Esta función se utiliza para listar todos las marcas que puede poseer un arma, con la lista de los modelos asociados a las mismas*
Nombre:	List<ModeloArma> listarModeloArmaPorMarca(MarcaArma marca)
Descripción:	*Esta función se utiliza para listar todos los modelos que puede tener la marca de un arma*
Nombre:	List<TipoArma> listarTipoArma()
Descripción:	*Esta función se utiliza para listar todas los tipos de armas*
Nombre:	List<TipoSerial> listarTipoSerial()
Descripción:	*Esta función se utiliza para listar todas los tipos de seriales que puede tener un arma*
Nombre:	RazonSerialModificado obtenerRazonJustificacion()
Descripción:	*Esta función se utiliza para obtener la razón para la cual es obligatoria una justificación*
Nombre:	List<RazonSerialModificado> listarRazonSerialModificado()
Descripción:	*Esta función se utiliza para listar todas las razones para modificar un serial del arma*

Nombre:	List<Caucho> listarCauchos()
Descripción:	*Listar cauchos de Vehiculo*
Nombre:	List<EstadoPlaca> listarEstadoPlaca()
Descripción:	*Esta función se utiliza para listar todos los estados que puede tener la placa de un vehículo*
Nombre:	List<EstadoMotorCarroceria> listarEstadoMotorCarroceria()
Descripción:	*Esta función se utiliza para listar todos los estados que puede tener un motor y la carrocería de un vehículo*
Nombre:	List<UsoPlaca> listarUsoPlaca()
Descripción:	*Esta función se utiliza para listar todos los usos de la placa de un vehículo*
Nombre:	List<TipoVehiculo> listarTipoVehiculo()
Descripción:	*Esta función se utiliza para listar todos los tipos de vehículo*
Nombre:	List<ClaseVehiculo> listarClaseVehiculo()
Descripción:	*Esta función se utiliza para listar todos las clases de vehículos*
Nombre:	List<MaterialRines> listarMaterialRines()
Descripción:	*Esta función se utiliza para listar todos los materiales rines*
Nombre:	List<TipoDispositivo> listarTiposDispositivos()
Descripción:	*Esta función se utiliza para la listar todos los dispositivo vehículo*
Nombre:	List<MarcaVehiculo> listarMarcasVehiculosConModelos()
Descripción:	*Esta función se utiliza para listar marcas con modelo vehículo*
Nombre:	List<EstadoRecuperacion> listarEstadosRecuperacionObjeto()
Descripción:	*Esta función se utiliza para listar los estados de recuperación de un Objeto*
Nombre:	List<EstadoRecuperacion> listarEstadosRecuperacionArma()
Descripción:	*Esta función se utiliza para listar los estados de recuperación de un Arma*
Nombre:	List<EstadoRecuperacion> listarEstadosRecuperacionVehiculo()
Descripción:	*Esta función se utiliza para listar los estados de recuperación de un Vehiculo*
Nombre:	List<EstadoVenezolano> listarEstadosVenezolanos()
Descripción:	*Esta función se utiliza para listar estados venezolanos*
Nombre:	List<Municipio> listarMunicipios(EstadoVenezolano estado)
Descripción:	*Esta función se utiliza para listar municipios dado un estado*
Nombre:	List<Parroquia> listarParroquias(Municipio municipio)
Descripción:	*Esta función se utiliza para listar parroquias dado un municipio*
Nombre:	List<Pais> listarPaises()
Descripción:	*Esta función se utiliza para listar países*
Nombre:	List<NCatalogo> listarSexoPersona()
Descripción:	*Esta función se utiliza para listar sexos*
Nombre:	List<NCatalogo> listarRangoEdad()
Descripción:	*Esta función se utiliza para listar rangos de edad*
Nombre:	List<NCatalogo> listarRangoEstatura()
Descripción:	*Esta función se utiliza para listar rangos de estatura de la persona*
Nombre:	List<NCatalogo> listarRangoPeso()
Descripción:	*Esta función se utiliza para listar rangos de peso de la persona*
Nombre:	List<NCatalogo> listarEstadoCivil()
Descripción:	*Esta función se utiliza para listar estado civil de la persona*
Nombre:	List<NCatalogo> listarGradoInstruccion()
Descripción:	*Esta función se utiliza para listar grados de instrucción de la persona*
Nombre:	List<NCatalogo> listarReligiones()

Descripción:	*Esta función se utiliza para listar religiones de la persona*
Nombre:	List<NCatalogo> listarNacionalidades()
Descripción:	*Esta función se utiliza para listar nacionalidades de la persona*
Nombre:	List<NTipoNombreAsociado> listarTipoNombresAsociados()
Descripción:	*Esta función se utiliza para listar nombres asociados de la persona*
Nombre:	List<NCarrera> listarCarreras()
Descripción:	*Esta función se utiliza para listar carreras de la persona*
Nombre:	List<NEspecialidad> listarEspecialidades(NCarrera carrera)
Descripción:	*Esta función se utiliza para listar especialidades dada una carrera de la persona*
Nombre:	List<NProfesion> listarProfesiones(NEspecialidad especialidad)
Descripción:	*Esta función se utiliza para listar profesiones dada una especialidad de la persona*
Nombre:	List<NEsfera> listarEsferas()
Descripción:	*Esta función se utiliza para listar esferas de la persona*
Nombre:	List<NOficio> listarOficios(NEsfera esfera)
Descripción:	*Esta función se utiliza para listar oficios dada una esfera de la persona*
Nombre:	List<NTipoTelefono> listarTipoTelefonos()
Descripción:	*Esta función se utiliza para listar tipos de teléfonos de la persona*
Nombre:	List<NCompanniaTelefono> listarCompanniasTelefono()
Descripción:	*Esta función se utiliza para listar compañías de teléfono de la persona*
Nombre:	List<TipoDireccion> listarTipoDireccion()
Descripción:	*Esta función se utiliza para listar tipos de dirección de la persona*
Nombre:	List<TipoResidencia> listarTipoResidencia()
Descripción:	*Esta función se utiliza para listar tipos de residencia de la persona*
Nombre:	List<NTipoRelacion> listarTiposRelacionesFiliales()
Descripción:	*Esta función se utiliza para listar tipos de relaciones filiales de la persona*
Nombre:	List<NCatalogo> listarTamannoCara()
Descripción:	*Esta función se utiliza para listar tamaños de la cara de la persona*
Nombre:	List<NCatalogo> listarFormaCara()
Descripción:	*Esta función se utiliza para listar formas de la cara de la persona*
Nombre:	List<NCatalogo> listarTamannoFrente()
Descripción:	*Esta función se utiliza para listar tamaños de la frente de la persona*
Nombre:	List<NCatalogo> listarFormaFrente()
Descripción:	*Esta función se utiliza para listar forma de la frente de la persona*
Nombre:	List<NCatalogo> listarSeparacionCejas()
Descripción:	*Esta función se utiliza para listar separación de las cejas de la persona*
Nombre:	List<NCatalogo> listarTipoCejaDerecha()
Descripción:	*Esta función se utiliza para listar tipos de cejas derecha de la persona*
Nombre:	List<NCatalogo> listarTipoCejaIzquierda()
Descripción:	*Esta función se utiliza para listar tipos de cejas izquierda de la persona*
Nombre:	List<NCatalogo> listarTamannoCejaIzquierda()
Descripción:	*Esta función se utiliza para listar tamaños de cejas izquierda de la persona*
Nombre:	List<NCatalogo> listarEspesorCejaIzquierda()
Descripción:	*Esta función se utiliza para listar espesores de cejas izquierda de la persona*
Nombre:	List<NCatalogo> listarTamannoCejaDerecha()
Descripción:	*Esta función se utiliza para listar tamaños de cejas derecha de la persona*

Nombre:	List<NCatalogo> listarEspesorCejaDerecha()
Descripción:	*Esta función se utiliza para listar espesores de cejas derecha de la persona*
Nombre:	List<NCatalogo> listarLobuloOrejaIzquierda()
Descripción:	*Esta función se utiliza para listar lóbulos de oreja izquierda de la persona*
Nombre:	List<NCatalogo> listarTamannoOrejaIzquierda()
Descripción:	*Esta función se utiliza para listar tamaños de oreja izquierda de la persona*
Nombre:	List<NCatalogo> listarLobuloOrejaDerecha()
Descripción:	*Esta función se utiliza para listar lóbulos de oreja derecha de la persona*
Nombre:	List<NCatalogo> listarTamannoOrejaDerecha()
Descripción:	*Esta función se utiliza para listar tamaños de oreja derecha de la persona*
Nombre:	List<NCatalogo> listarTamannoNariz()
Descripción:	*Esta función se utiliza para listar tamaños de nariz de la persona*
Nombre:	List<NCatalogo> listarFormaNariz()
Descripción:	*Esta función se utiliza para listar formas de nariz de la persona*
Nombre:	List<NCatalogo> listarCaracteristicasNariz()
Descripción:	*Esta función se utiliza para listar características de la nariz de la persona*
Nombre:	List<NCatalogo> listarCaracteristicasMenton()
Descripción:	*Esta función se utiliza para listar características del mentón de la persona*
Nombre:	List<NCatalogo> listarFormaMenton()
Descripción:	*Esta función se utiliza para listar formas del mentón de la persona*
Nombre:	List<NCatalogo> listarNaturalezaCabello()
Descripción:	*Esta función se utiliza para listar la naturaleza del cabello de la persona*
Nombre:	List<NCatalogo> listarLargoCabello()
Descripción:	*Esta función se utiliza para listar el largo del cabello de la persona*
Nombre:	List<NCatalogo> listarTipoCabello()
Descripción:	*Esta función se utiliza para listar el tipo de cabello de la persona*
Nombre:	List<NCatalogo> listarTipoCalvicie()
Descripción:	*Esta función se utiliza para listar el tipo de calvicie de la persona*
Nombre:	List<NCatalogo> listarColorCabello()
Descripción:	*Esta función se utiliza para listar el color del cabello de la persona*
Nombre:	List<NCatalogo> listarPlanchas()
Descripción:	*Esta función se utiliza para listar los tipos de planchas de la persona*
Nombre:	List<NCatalogo> listarDesdentado()
Descripción:	*Esta función se utiliza para listar los tipos de desdentado de la persona*
Nombre:	List<NCatalogo> listarPuentes()
Descripción:	*Esta función se utiliza para listar los tipos de puentes de la persona*
Nombre:	List<NCatalogo> listarCaracteristicasDientes()
Descripción:	*Esta función se utiliza para listar las características de los dientes de la persona*
Nombre:	List<NCatalogo> listarColorPiel()
Descripción:	*Esta función se utiliza para listar los colores de la piel de la persona*
Nombre:	List<NCatalogo> listarTipoPiel()
Descripción:	*Esta función se utiliza para listar los tipos de piel de la persona*
Nombre:	List<NCatalogo> listarEspesorBarba()
Descripción:	*Esta función se utiliza para listar espesores de barba de la persona*
Nombre:	List<NCatalogo> listarEspesorBigote()
Descripción:	*Esta función se utiliza para listar espesores de bigote de la persona*

Nombre:	List<NCatalogo> listarLargoPatillas()
Descripción:	*Esta función se utiliza para listar largos de la patilla de la persona*
Nombre:	List<NCatalogo> listarTamannoBoca()
Descripción:	*Esta función se utiliza para listar tamaños de la boca de la persona*
Nombre:	List<NCatalogo> listarEspesorLabioSuperior()
Descripción:	*Esta función se utiliza para listar espesores del labio superior de la persona*
Nombre:	List<NCatalogo> listarCaracteristicasLabioSuperior()
Descripción:	*Esta función se utiliza para listar características del labio superior de la persona*
Nombre:	List<NCatalogo> listarEspesorLabioInferior()
Descripción:	*Esta función se utiliza para listar espesores del labio inferior de la persona*
Nombre:	List<NCatalogo> listarCaracteristicasLabioInferior()
Descripción:	*Esta función se utiliza para listar características del labio inferior de la persona*
Nombre:	List<NCatalogo> listarColorIrisOjoIzquierdo()
Descripción:	*Esta función se utiliza para listar colores del iris del ojo izquierdo de la persona*
Nombre:	List<NCatalogo> listarTipoOjoIzquierdo()
Descripción:	*Esta función se utiliza para listar tipos del ojo izquierdo de la persona*
Nombre:	List<NCatalogo> listarTamannoOjoIzquierdo()
Descripción:	*Esta función se utiliza para listar tamaños del ojo izquierdo de la persona*
Nombre:	List<NCatalogo> listarCaracteristicasOjoIzquierdo()
Descripción:	*Esta función se utiliza para listar características del ojo izquierdo de la persona*
Nombre:	List<NCatalogo> listarColorIrisOjoDerecho()
Descripción:	*Esta función se utiliza para listar tamaños del ojo derecho de la persona*
Nombre:	List<NCatalogo> listarTipoOjoDerecho()
Descripción:	*Esta función se utiliza para listar tipos del ojo derecho de la persona*
Nombre:	List<NCatalogo> listarTamannoOjoDerecho()
Descripción:	*Esta función se utiliza para listar tamaños del ojo derecho de la persona*
Nombre:	List<NCatalogo> listarCaracteristicasOjoDerecho()
Descripción:	*Esta función se utiliza para listar características del ojo derecho de la persona*
Nombre:	List<NTipoSenna> listarTipoSennas()
Descripción:	*Esta función se utiliza para listar tipos de señas de la persona*
Nombre:	List<NZona> listarZonaMorfologica(NTipoSenna tipoSenna)
Descripción:	*Esta función se utiliza para listar zonas morfológicas dado un tipo de seña de la persona*
Nombre:	List<NLugar> listarLugarMorfologico(NZona zona, NTipoSenna tipoSenna)
Descripción:	*Esta función se utiliza para listar lugares morfológicas dado una zona y un tipo de seña de la persona*
Nombre:	List<NLocalizacion> listarLocalizacionMorfologica(NLugar lugar)
Descripción:	*Esta función se utiliza para listar localizaciones morfológicas dado un lugar de la persona*
Nombre:	List<NCatalogo> listarManeraHablar()
Descripción:	*Esta función se utiliza para listar las maneras de hablar de la persona*
Nombre:	List<NCatalogo> listarTipoVoz()
Descripción:	*Esta función se utiliza para listar los tipos de voz de la persona*
Nombre:	List<NCatalogo> listarCaracteristicasVoz()
Descripción:	*Esta función se utiliza para listar las características de voz de la persona*
Nombre:	List<NCatalogo> listarTics()
Descripción:	*Esta función se utiliza para listar tics de la persona*

Nombre:	List<NCatalogo> listarHabitos()
Descripción:	*Esta función se utiliza para listar hábitos de la persona*
Nombre:	List<NCatalogo> listarFormaCaminar()
Descripción:	*Esta función se utiliza para listar formas de caminar de la persona*
Nombre:	List<NCatalogo> listarCaracteristicasCaminar()
Descripción:	*Esta función se utiliza para listar características de caminar de la persona*
Nombre:	List<NCatalogo> listarCondicionesFisicas()
Descripción:	*Esta función se utiliza para listar condiciones físicas de la persona*
Nombre:	List<NCatalogo> listarContextura()
Descripción:	*Esta función se utiliza para listar contexturas de la persona*
Nombre:	List<NCatalogo> listarOrientacionSexual()
Descripción:	*Esta función se utiliza para listar orientaciones sexuales de la persona*
Nombre:	List<NCatalogo> listarOtrasCondiciones()
Descripción:	*Esta función se utiliza para listar otras condiciones de la persona*
Nombre:	EstadoVenezolano obtenerEstadoPorId(Integer idEstadoVenezolano)
Descripción:	*Esta función se utiliza para listar estados venezolanos dado su id*
Nombre:	Municipio obtenerMunicipioPorIdEstadoIdMunicipio(Integer idEstado, Integer idMunicipio);
Descripción:	*Esta función se utiliza para listar municipios dados los id de estado y municipio*
Nombre:	Parroquia obtenerParroquiaPorIdestadoIdMunicipioIdParroquia(Integer idMunicipio, Integer idParroquia)
Descripción:	*Esta función se utiliza para listar parroquias dados los id de municipio y parroquia*
Nombre:	TipoDireccion obtenerTipoDireccionHabita()
Descripción:	*Esta función se utiliza para listar tipos de direcciones de tipo habita de la persona*
Nombre:	List<NEstadoPersona> listarEstadosPersona()
Descripción:	*Esta función se utiliza para listar status legales de la persona*
Nombre:	NEstadoPersona obtenerEstadoInicialPersona()
Descripción:	*Esta función se utiliza para listar estados iníciales de la persona*
Nombre:	List<Cargo> listarCargos()
Descripción:	*Esta función se utiliza para listar cargos del funcionario*
Nombre:	List<Rango> listarRangos()
Descripción:	*Esta función se utiliza para listar rangos del funcionario*
Nombre:	List<EstadoRecuperacion> listarEstadosRecuperacionObjeto()
Descripción:	*Esta función se utiliza para listar los posibles estados de recuperación de un objeto*
Nombre:	List<EstadoRecuperacion> listarEstadosRecuperacionVehiculo()
Descripción:	*Esta función se utiliza para listar los posibles estados de recuperación de un vehículo*
Nombre:	public List<EstadoRecuperacion> listarEstadosRecuperacionArma()
Descripción:	*Esta función se utiliza para listar los posibles estados de recuperación de un arma*
Nombre:	List<NCatalogo> obtenerCatalogo(String nombre)
Descripción:	*Esta función se utiliza para obtener un lista de catalogo por nombre*
Nombre:	NCatalogo obtenerCatalogo(String nombre, String valor)
Descripción:	*Esta función se utiliza para obtener un catalogo por nombre y valor*

Tabla 30. Descripción de la Clase NomencladorFacadeImpl.

Nombre de la Clase:	NomencladorServiceImpl
Tipo de Clase:	Controladora
Atributo:	**Tipo:**
nomencladorDao	NomencladorDao
elementoDao	ElementoDao
nomencladorObjetoDao	NomencladorObjetoDao
nomencladorPersonaDao	NomencladorWareHouse
nomencladorFuncionarioDao	NomencladorFuncionarioDao
nomencladorArmaDao	NomencladorArmaDao
nomencladorVehiculoDao	NomencladorVehiculoDao
nomencladorPersonaDaoNew	NomencladorPersonaDao
carga	Carga
Para Cada Responsabilidad:	
Nombre:	Ídem nombres de las operaciones de NomencladorFacadeImpl
Descripción:	*Ídem descripciones de las operaciones de NomencladorFacadeImpl*

Tabla 31. Descripción de la Clase NomencladorServiceImpl.