FACULTAD 2

Análisis, Diseño e Implementación del módulo Transmisiones del Sistema de Investigación e Información Policial.

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

Autor:

Adonis Montoya Castillo

Tutores:

Ing. Humberto Rivero Guevara

Ing. Maylin Díaz Cabrera

Ciudad de la Habana, Junio 2011

"Año 53 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de_____ del año 2011.

_____

AUTOR

Adonis Montoya Castillo

_____ _____

TUTOR **TUTOR**

Ing. Humberto Rivero Guevara **Ing. Maylin Díaz Cabrera**

AGRADECIMIENTOS

A Fidel y a la Revolución por darnos la maravillosa oportunidad de estudiar en esta universidad y de crecer como hombres libres.

A mis padres, por ser tan comprensivos y atentos, por el amor, el cariño y la dedicación que me han brindado y por toda la confianza que han depositado en mí. A los que se lo debo todo y me han permitido ser quien soy. A ellos que se han convertido en mi inspiración y mi sentido de ser.

A mis hermanas, que no puedo quejarme de la agradable oportunidad que me ha dado la vida por tenerlas ahí siempre conmigo tan compresivas, tan sabias y tan amorosas, a ellas todo mi corazón.

A mis sobrinos, que aunque pequeños me han hecho ser una mejor persona por considerarme su ejemplo a seguir, los que se han convertido en mis hijos más traviesos.

A mi novia Sahyli, por estar ahí cuando la necesito y ser tan comprensiva y ayudarme en todo lo que ha podido.

A mi tutora Maylin, que no sé qué hubiese sido de mí en esta batalla sin su ayuda, sus continuos consejos y su confianza plena. A ella le agradezco mucho porque siempre estuvo ahí y se convirtió en mi ejemplo de luchadora, buena profesional y excelente persona. Con todo mi corazón gracias.

A mis amigos, que son mis otros hermanos muy queridos, que siempre me han dado fuerzas para emprender este largo camino, a lo que están aquí y a los que no pero sepan que a todos los tengo presente Osvel, Aldo, Jesús, Manuel Alejandro, Alejandro Lafourcade, Carlos Miguel, Daniel Bopelier, Rodolfo Roca, Franklin, Dalied, Suramis mi hermanita, Gerardo, Feliberto, Tomas Orlando, Omar, Armando Batista y a todos los demás.

A mi familia por ser tan unida y siempre tenerme presente. A mis tíos, a primos, a mis abuelos aunque algunos ya no estén entre nosotros, en fin, a todos.

A mis profes amigos y mi equipo de desarrollo Sustancias Químicas e Interpol que me han brindado un hogar en mi proyecto en especial a Kenny y Adrián

A todos aquellos que, de una forma u otra, han contribuido a mi formación tanto profesional como personal durante estos cinco años.

A mi tribunal en general, por ser tan exigente conmigo provocando que me prepare más ayudándome a ser un mejor profesional

DEDICATORIA

A las personas más especiales que he tenido en toda mi vida, mis padres: Inés María y Hugo Adonis y mis hermanas Yudanis y Yudania. Ellos han hecho de mí el hombre que soy, me han enseñado mucho de lo que sé y siempre han estado ahí para mí. Por eso, a ustedes va dedicado este trabajo que con tanto esfuerzo he realizado para darle un poco de alegría y orgullo. Los quiero con mi vida.

RESUMEN

El presente documento centra su atención en la investigación y proceso de desarrollo del módulo Transmisiones[1] del Sistema de Investigación e Información Policial (SIIPOL), un sistema de gestión policial que surge a raíz de los proyectos contemplados en el contrato de modernización del Cuerpo de Investigaciones Científicas Penales y Criminalísticas (CICPC).

El SIIPOL, es un sistema creado para automatizar todos los procesos que tienen lugar dentro de los despachos del CICPC de una forma óptima, utilizando tecnología de punta para la gestión y el procesamiento de la información que estos manejan. Se desea que el sistema contemple los procesos de todas las áreas operativas de la organización lo cual se está logrando paulatinamente y con esa finalidad se desarrolla la presente investigación que persigue el objetivo de incluir las funcionalidades que automatizarán los procesos del despacho de Transmisiones, el cual se encarga de gestionar y procesar la información generada por las comunicaciones que se producen entre los agentes desplegados en el terreno con el oficial de guardia para informar acerca de hechos delictivos ocurridos como robos, asaltos, asesinatos, y otros o verificar alguna información.

Este documento tiene plasmado la documentación necesaria del proceso de desarrollo del módulo Transmisiones del SIIPOL, módulo que pretende disminuir los tiempos de respuesta de las investigaciones desarrolladas en dicho despacho.

[1] Módulo que se encarga de gestionar y procesar toda la información generada por las Transmisiones realizadas en un turno de guardia.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El Cuerpo de Investigaciones Científicas, Penales y Criminalísticas (CICPC) es la Institución en la hermana República Bolivariana de Venezuela que garantiza la eficiencia en la investigación del delito, mediante su determinación científica, asegurando que el ejercicio de la acción penal conduzca a una administración de la justicia. Con la finalidad de alcanzar el más alto nivel de credibilidad nacional e internacional en la investigación del fenómeno delictivo y criminalidad violenta y con la visión de cumplir el decreto que instituye las competencias del CICPC como órgano principal de investigaciones penales al servicio del Estado, el gobierno bolivariano se empeña en superar las deficiencias que esta presenta en la actualidad. *(1)*

Atendiendo a un estudio de la situación general que presenta la Institución, se identifican un conjunto de problemáticas en los procesos que desarrollan y en los cuales están involucradas todas las áreas de la Institución. Por mencionar algunos ejemplos: *(2)*

1. Lentitud en la fluidez de la información entre las diferentes áreas de trabajo del CICPC, que deben coordinar su trabajo en la solución de los casos. (2)

2. Falta de información actualizada, oportuna y fiable para los entes de dirección del CICPC, que no permite el conocimiento táctico sobre el curso de las investigaciones de un caso, ni mejorar su contribución al desarrollo de políticas, estrategias y análisis sobre la criminalidad. (2)

3. Limitaciones en la diversidad de información que se requiere para la investigación de los hechos y en el uso de la que hoy está almacenada. (2)

4. Imposibilidad para acceder y utilizar información de interés de otras organizaciones como CNE (Consejo Nacional Electoral), Registro y Notarias, Penitenciarias, entre otras. (2)

Para dar solución a estos problemas en el marco de las relaciones entre Cuba y Venezuela por la colaboración entre los países de la Alternativa Bolivariana para las Américas (ALBA), se ha concebido el proyecto de Modernización del CICPC, el cual se centra en la construcción de un nuevo sistema que sustituya las actuales prestaciones del SIIPOL, mejore e incorpore las funcionalidades del resto de las áreas operativas y que contribuya a la disminución de los tiempos de respuesta de cada investigación.

Tras un previo análisis de los procesos en la institución se han detectado una serie de requisitos funcionales y no funcionales con los cuales el nuevo sistema debe cumplir para mejorar el nivel de respuesta a las necesidades de seguridad del ciudadano venezolano y facilitar la sistematización del trabajo en sus

dependencias. Como resultado se modeló un sistema informático dividido en módulos, donde uno de ellos es Transmisiones.

Dentro de este módulo se encuentra las funcionalidades asociadas al área de Trasmisiones, los cuales no están incorporados en el SIIPOL actual. En este sistema no existe un subsistema que gestione y procese la información generada por las Transmisiones[2] que se realizan en un turno de guardia. Los funcionarios que están de guardia se encargan del engorroso proceso de gestionar manualmente la información relativa a los eventos que ocurrían en la vida cotidiana de los venezolanos como: robos, asaltos, asesinatos, entre otros. Durante todo el proceso se manejan grandes volúmenes de información que se archivan físicamente, lo que trae como consecuencia que las búsquedas de información que se realicen se tornen lentas y engorrosas. También, estos archivos con el tiempo pueden deteriorarse o perderse, provocando que no exista un control adecuado de las Transmisiones archivadas en el CICPC. Además, se incurre en atrasos para la solución de los delitos, pérdida innecesaria del tiempo de los trabajadores y gastos monetarios en la compra de material de oficina para el archivo de la información.

Por todo lo planteado anteriormente se definió como **problema a resolver**: **¿Cómo garantizar el cumplimiento de los requisitos funcionales y no funcionales asociados al módulo Transmisiones del SIIPOL?**

Para lo cual se plantean los siguientes **objetivos:**

Objetivo general: Desarrollar un subsistema que cumpla con los requisitos funcionales y no funcionales asociados al módulo de Transmisiones en el SIIPOL.

Objetivos específicos:

- Analizar los Casos de Uso del módulo Transmisiones del SIIPOL.
- Analizar los procesos vinculados a la Gestión de las Transmisiones.

[2] Transmisiones: comunicaciones que se producen entre los agentes desplegados en el terreno con el oficial de guardia para informar acerca de hechos delictivos ocurridos como robos, asaltos, asesinatos, y otros o verificar alguna información.

- Diseñar cada una de las capas arquitectónicas del Módulo Transmisiones.
- Implementar cada una de las capas arquitectónicas de modo que sean capaces de dar respuesta a todos los requisitos funcionales y no funcionales del Módulo Transmisiones.
- Integrar el módulo al SIIPOL.

El objeto de estudio se define como el *Proceso de desarrollo del software de gestión SIIPOL*, y el **campo de acción** del mismo es el módulo Transmisiones.

Como **idea a defender:** Al realizar una correcta utilización de la metodología de desarrollo de software para el análisis, diseño e implementación del módulo Transmisiones, se podrá desarrollar un sistema que cumpla correctamente con los requisitos funcionales y no funcionales definidos para el mismo.

Tareas a desarrollar:

Para darle cumplimiento a los objetivos se han trazado las siguientes tareas**:**

- Elaboración del diseño teórico de la investigación.
- Definición de la Situación Problemática.
- Definición del Problema a Resolver.
- Definición del Objetivo General.
- Definición del Objeto de Estudio.
- Elaboración de la Fundamentación Teórica de la Investigación.
- Estudio del Estado del Arte.
- Investigación acerca de la metodología de desarrollo de software a utilizar.
- Análisis de la Descripción de los Casos de Uso de forma detallada.
- Realización de los estudios correspondientes sobre la arquitectura de software a utilizar para desarrollar el Módulo Trasmisiones.

- Análisis de los patrones de diseño que pueden aplicarse en el desarrollo del módulo.
- Diseño de las Clases pertinentes, paquetes y componentes definidos por la metodología utilizada para la solución.
- Implementación de las funcionalidades necesarias para dar solución a los requisitos funcionales y no funcionales de cada caso de uso.
- Validación de la propuesta de solución.

Para darle cumplimiento al objetivo general y respuesta al problema científico planteado fueron aplicados los siguientes **métodos teóricos**

- **Analítico-Sintético:** Para la realización del análisis de los elementos referentes a la investigación y la síntesis detallada de la misma.
- **Inductivo-Deductivo:** Para extraer regularidades y elaborar conclusiones de la tesis.
- **Histórico-Lógico:** Para comenzar con el trabajo se realizó un estudio de las investigaciones anteriormente hechas sobre el tema.

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

- **Capítulo 1. Fundamentación Teórica:** En este capítulo se abordan elementos teóricos relacionados con la investigación como son la metodología a utilizar, herramientas y elementos para el desarrollo y el estudio de los sistemas similares existentes en el campo de acción.
- **Capítulo 2. Análisis, diseño e implementación de la propuesta de solución:** En este capítulo se hace referencia a los flujos de trabajo de análisis, diseño e implementación, donde se expone el análisis de los principales patrones utilizados, se mencionan los elementos más importantes de la arquitectura y se tratan los principales artefactos generados a lo largo de estos flujos de trabajo.
- **Capítulo 3. Validación de la solución propuesta:** En este capítulo se pretende, mediante las pruebas que se le aplicarán a la solución final, validar la solución propuesta quedando evidenciado que el producto final es válido con respecto al objetivo general definido.

CAPÍTULO 1. Fundamentación Teórica

Introducción

Al iniciarse el desarrollo de un producto de software unas de las primeras actividades que se realizan, son en función de definir el tipo de aplicación y su ambiente de desarrollo. En este capítulo se aborda la fundamentación teórica del desarrollo del módulo Transmisiones del SIIPOL, el estudio de sistemas similares existentes vinculados al campo de acción; así como las tendencias, tecnologías, metodologías y herramientas actuales, realizando una selección de aquellas que serán utilizadas durante el desarrollo del proyecto.

Por cuestiones de seguridad los sistemas que serán descritos a continuación solo hacen alusión a algunos datos escuetos, debido a que las compañías desarrolladoras y las instituciones que utilizan estos sistemas se reservan sus propiedades y características para evitar ataques al sistema y proteger la integridad de los datos almacenados.

Software de Gestión Información

Un ***Sistema de Gestión de Información*** es un producto capaz de integrar todo un conjunto de actividades que se realizan dentro de una organización o empresa. Estos sistemas se utilizan para automatizar procesos y dar apoyo a otras acciones, con el objetivo de mejorar el tiempo de respuesta a las necesidades de los clientes u otras entidades, para acceder concurrentemente a documentos e información, minimizar el tiempo de acceso a información crucial, así como al análisis de estadísticas y resultados de investigaciones para la toma de decisiones, generar documentos automáticamente y dar solución a la problemática planteada, reducir riesgos de pérdida de documentos de vital importancia para la empresa, proporcionando grandes beneficios en todas las áreas de la misma. Recolectan gran cantidad de información la cual luego puede ser analizada y explotada. Un Sistema de Gestión de información de calidad debe ser de fácil utilización por los usuarios sin necesidad de intervención de un especialista en la rama para el desarrollo de los procesos que controla, con el uso de una interfaz visual de acuerdo a las necesidades del cliente final y la utilización de técnicas de programación avanzada para la implementación de las mismas. (2)

Sistema de Gestión de Información Policial

Un ***Sistema de Gestión de Información Policial*** se restringe solamente a gestionar los procesos en el área de las entidades policiales, incrementando la satisfacción del cliente y reduciendo los costos legales, protegiendo información sensible referente a la población y del país en general. El éxito de estos sistemas está dado por la confidencialidad, integridad y la seguridad de los datos que almacena, se conoce muy poco de las herramientas y metodologías de desarrollo de este tipo de software; es por ello que la información publicada sobre los mismos es escueta y básica. (2) Luego de un proceso de búsqueda detallada referente a algunos sistemas similares que pudieran apoyar la elaboración de la propuesta de solución, se encontraron los siguientes:

Sistema Territorial de Emergencias y Gestión Policial (STEGPOL)

Es un Sistema de Información Geográfica sobre una Plataforma Nacional Común de Información aplicada al "Sistema de Emergencias Nacionales" y al "Sistema Territorial de Gestión Policial" que actualmente operan en Chile. Se caracteriza por ser un sistema con tecnología de punta aplicado al "control territorial de la gestión policial", el cual identifica en forma veraz y efectiva dónde, geográficamente hablando, se están cometiendo o se han cometido actos delictivos a nivel territorial, apoyado por sistemas de información en línea desde el lugar de los hechos; lo cual permite la acción rápida y coordinada entre los diferentes actores encargados de la seguridad ciudadana a nivel nacional. Integra diferentes entidades como Carabineros, Investigaciones, Ministerio del Interior y Municipios, entre otros, en una Plataforma Nacional Común de Información que permite el intercambio de datos y que sirve de apoyo a la gestión operacional regional o comunal donde dichas instituciones están interconectadas entre sí, en especial con diferentes Divisiones o Departamentos de Carabineros, tales como Jefaturas de Zona, Prefecturas, Comisarías, Sub-Comisarías, Tenencias y por último los Retenes.

Vía la Intranet del Estado o vía comunicación en Banda Ancha - Internet se puede llegar a acceder a dicho Sistema Territorial de Emergencias y Gestión Policial (STEGPOL), montado en la Web con diferentes contraseñas de acceso restringido tanto para los Usuarios Operadores encargados de ingresar o modificar datos en línea, como para aquellos que solo tengan acceso a consultar dicho sistema. (3)

Sistema de Gestión Penitenciaria (SIGEP)

Proyecto productivo que es desarrollado por la Universidad de las Ciencias Informáticas, tributando al proceso de Informatización y modernización de las Instituciones de la República Bolivariana de Venezuela a raíz de los acuerdos del ALBA. Tiene gran similitud en cuanto a las tecnologías solicitadas por el cliente y el equipo de desarrollo, sus funcionalidades y procesos están dirigidas principalmente a las prisiones venezolanas, pero su arquitectura y herramientas constituyeron el punto de partida para el proceso investigativo previo para determinar cuáles usar en el desarrollo del SIIPOL. Para la elaboración del mismo utilizaron como plataforma de programación Java manejando los frameworks Acegi (actualmente SpringSecurity) para el manejo de la seguridad, Hibernate para la capa de Acceso a Datos, Spring para la de negocio y el módulo Spring MVC para la capa de presentación. (2)

EL Sistema Integrado de Información Policial

El Sistema Integrado de Información Policial (SIIPOL) es la aplicación informática que utilizaba el CICPC para realizar los procesos internos que desarrollan a diario en las dependencias, delegaciones y subdelegaciones. Está desarrollado sobre una tecnología actualmente obsoleta con servidores de Base de Datos SUN 6500 y base de datos Adabas[3] con lenguaje de programación Natural para el manejo de los datos; el acceso a este sistema actualmente se hace a través de un emulador, como Personal Comunication, o un Telnet. El servidor de la aplicación es con tecnología de Sun con Sistema Operativo Solaris. Utilizan un programa llamado Natural Security que es el que valida la gestión de los usuarios para la seguridad del SIIPOL.

Para mantener actualizado el sistema, el CICPC cuenta con una división de información policial y varias divisiones de análisis y seguimiento de la información; que se encargan de introducir o actualizar la

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

información que es recopilada por las dependencias, y ayudan a las que no tienen conexión directa con el SIIPOL. Además se encarga de tramitar los datos e informaciones que son requeridos por esas dependencias para hacer las averiguaciones y también suministra información a las fiscalías y tribunales. Aun así todos los casos que actualmente se trabajan en el CICPC no están registrados en la aplicación informática.

Además se utilizan datos de otras instituciones como la Base de Datos de SAIME (Servicio Automatizado de Inmigración y Extranjería), y la del INTT (Instituto Nacional de Transporte Terrestre), esta información se trae en soportes digitales para actualizar la Base de Datos del SIIPOL. El CICPC a su vez envía información para actualizar las Bases de Datos de esas instituciones. (2)

El Sistema de Investigación e Información Policial (SIIPOL)

La solución que se propuso por parte de la dirección del proyecto es construir un nuevo Sistema de Investigación e Información Policial (SIIPOL), tomando como referencia el Sistema Integrado de Información Policial existente, de forma tal que las funciones actuales sean mantenidas, incrementando las posibilidades de uso de la información y agregando nuevas funciones que el actual sistema no concibe, como es el caso de las Transmisiones.

El sistema centralizará en varios módulos o subsistemas, la automatización de los procesos que se llevan a cabo en las dependencias que pertenecen al CICPC. La organización y cantidad de subsistemas estará en la medida de las necesidades, por lo que no deben coincidir exactamente con los que existen actualmente en el SIIPOL; en principio los subsistemas deberán organizarse por los procesos, y no por áreas de trabajo, es decir que por ejemplo: existirá un único subsistema para la toma de la denuncia, independientemente de que existan varias áreas que lleven a cabo este proceso con sus correspondientes particularidades; y un único subsistema para la sustanciación del expediente investigativo independientemente del tipo de delito investigado y el área que lleve a cabo la investigación. (2)

1.1.1. Metodología, Lenguajes y Herramientas de Desarrollo

Mundialmente las instituciones y empresas que se dedican a la producción de software utilizan en su proceso de desarrollo diversas metodologías, métodos y procedimientos para lograr un producto competente y con alta calidad. El propósito de esto es lograr una organización óptima del personal del equipo de desarrollo asegurando así la calidad de los artefactos generados. Las Metodología se refiere a los

métodos de investigación que se siguen para alcanzar una gama de objetivos en una ciencia. Son vías que facilitan el descubrimiento de conocimientos seguros y confiables para solucionar los problemas que la vida plantea. En resumen son el conjunto de métodos que se rigen en una investigación científica o en una exposición doctrinal.

Una metodología de desarrollo del software define quién está haciendo qué, cuándo, y cómo alcanzar un objetivo específico. Esta proporciona normas para el desarrollo eficiente del software con calidad al captar las mejores prácticas que el estado actual de la tecnología permite. (4)

1.1.2. Proceso Unificado de Desarrollo de Software (RUP)

El Proceso Unificado del Software (RUP, Rational Unified Process), es un marco de trabajo genérico que puede especializarse en una gran variedad de sistemas de software, incorporando una serie de actividades que transforman los requisitos de un usuario en un producto informático. Unifica completamente a un equipo de desarrollo de software y optimiza la productividad de cada uno de los miembros del equipo brindándoles la experiencia de los líderes de la industria y las lecciones aprendidas a través de miles de proyectos. Está fundamentada en un enfoque orientado a modelos de desarrollo basado en componentes, utilizando para ello el Lenguaje de Modelado Unificado (UML, Unified Modeling Language) el que define técnicas de análisis y diseño que ayudan a la confección de una solución sólida de software. (4)

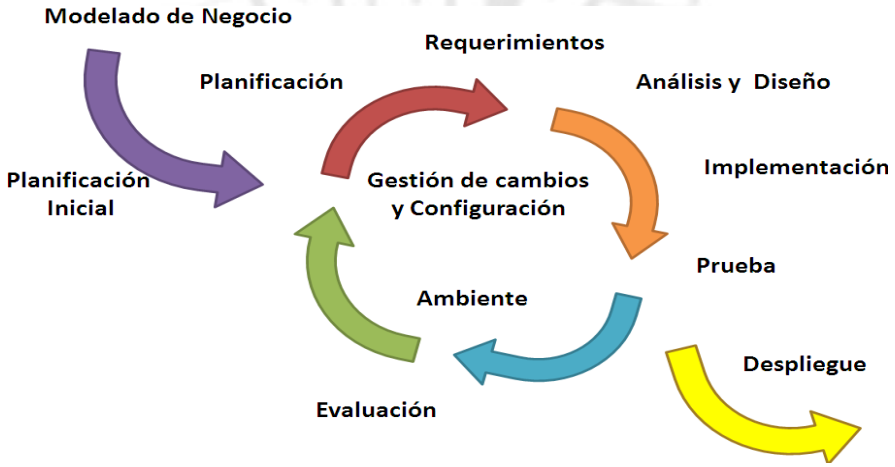


Ilustración 1. *Representación del proceso ingenieril del software planteado por RUP.*

Sus características principales son:

***Dirigido por casos de uso*:** Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requisitos. A partir de aquí los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso (cómo se llevan a cabo). (4)

Centrado en la arquitectura: La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. RUP se desarrolla mediante iteraciones, comenzando por los Casos de Uso (CU) relevantes desde el punto de vista de la arquitectura. El modelo de arquitectura se representa a través de vistas en las que se incluyen los diagramas de UML. (5)

Iterativo e incremental: La iteraciones referencian el transcurso entre los flujos de trabajos y los incrementos, al crecimiento del producto, permitiendo que en cada iteración se entregue una versión del software. La táctica de este proceso es alcanzar su objetivo por medio del orden y documentación, lo que lo clasifica como una metodología robusta. Está definido por cuatro fases (inicio, elaboración, construcción y transición), y nueve flujos de trabajo (Modelado del Negocio, Requisitos, Análisis y Diseño, Implementación, Pruebas, Despliegue, Gestión del Cambio y Configuraciones, Gestión de Proyecto y Entorno). (4)

RUP se define en 4 fases: (4)

- ***Inicio:*** Se describe el negocio y se delimita el alcance del proyecto con la determinación de los Casos de Uso del Sistema.

- ***Elaboración:*** Se define la arquitectura del sistema y se obtiene una aplicación ejecutable que responde a los casos de uso que la comprometen. A pesar de que se desarrolla a profundidad una parte del sistema, las decisiones sobre la arquitectura se hacen sobre la base de la comprensión del sistema completo y los requisitos (funcionales y no funcionales) identificados de acuerdo con el alcance definido.

- ***Construcción:*** Se logra un producto listo para su utilización que está documentado y tiene un manual de usuario. Se obtiene una o varias versiones del producto que han pasado las pruebas. Se ponen estos entregables a consideración de un subconjunto de usuarios.

- ***Transición:*** La versión ya está lista para su instalación en las condiciones reales. Puede implicar reparación de errores.

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

Los *flujos de Trabajo* que este define son los siguientes (ver Ilustración. 4):

- ***Modelación del negocio:*** Describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.

- **Requerimientos:** Define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.

- ***Análisis y diseño:*** Describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requisitos), por lo que indica con precisión lo que se debe programar.

- **Implementación:** Define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán, su ubicación en los componentes y la estructura de capas de la aplicación.

- ***Prueba:*** Busca los defectos a los largo del ciclo de vida.

- ***Instalación:*** Produce un entregable del producto y realiza actividades como empaque, instalación, asistencia a usuarios, etc. para entregar el software a los usuarios finales.

Flujos de soporte:

- ***Administración del proyecto:*** Involucra actividades con las que se busca construir un producto que satisfaga las necesidades de los clientes.

- ***Administración de configuración y cambios:*** Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización/actualización concurrente de elementos, control de versiones, etc.

- ***Ambiente:*** Contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto; así como el procedimiento para implementar el proceso en una organización.

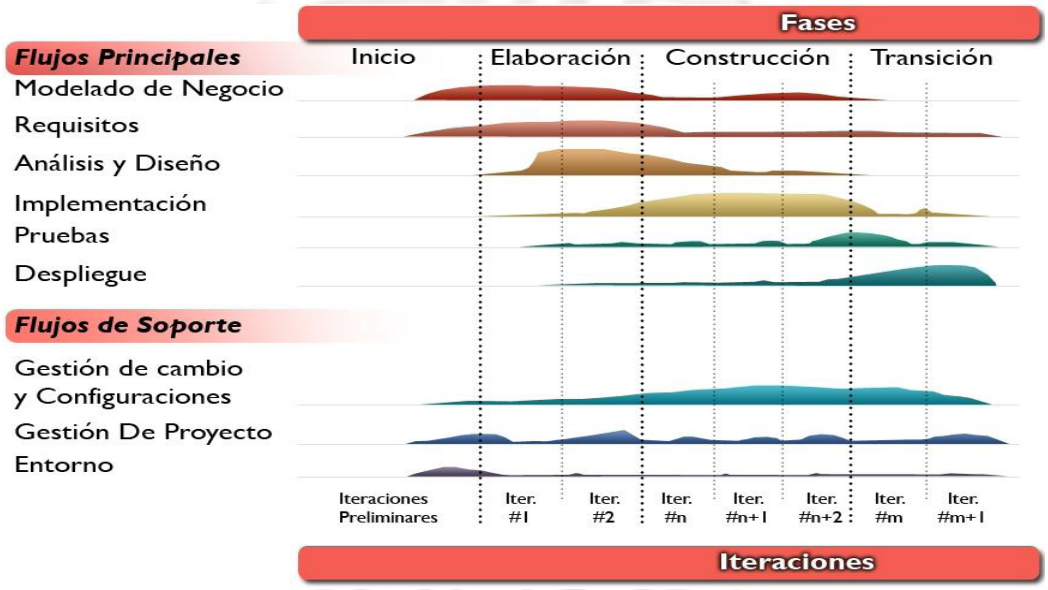


Ilustración 2. *RUP en 2 Dimensiones.*

1.1.3. Lenguaje de modelado UML

La modelación proporciona tres beneficios claves: visualización, complejidad de gestión y una comunicación clara. En la actualidad el auge en el desarrollo de software o aplicaciones de gran complejidad ha crecido hasta el punto de que un equipo de desarrollo ve la necesidad de la utilización de diagramas que ayuden al entendimiento pleno del software a desarrollar. Para ello, la solución se apoya en UML, lenguaje para la especificación, visualización, construcción y documentación de los artefactos de un proceso de sistema intensivo. Es de saber que UML es un "lenguaje" para especificar y no para describir métodos o procesos o sea es el lenguaje en el que está descrito el modelo. UML es un estándar del Grupo de Gestión de Objeto

(OMG)[4] y su utilización para el modelado es independiente del lenguaje de implementación, lo que permite graficar, documentar y construir los diseños necesarios y que éstos se puedan implementar en cualquier lenguaje que soporte las posibilidades de UML (como son casi todos los orientados a objeto), dando un apoyo significativo a los mismos. (6)

1.1.4. Herramientas Case

CASE sigla que corresponde a: Computer Aided Software Engineering; (Ingeniería de Software Asistida por Computación). El concepto de CASE es muy amplio; y una buena definición genérica, que pueda abarcar esa amplitud de conceptos, sería la de considerar a la Ingeniería de Software Asistida por Computación (CASE), como la aplicación de métodos y técnicas a través de las cuales se hacen útiles a las personas comprender las capacidades de las computadoras, por medio de programas, de procedimientos y su respectiva documentación. Concentrando nuestra atención en el uso de estas herramientas, para el desarrollo de proyectos informáticos que tengan como objetivo la automatización de procedimientos administrativos; se puede decir que: las herramientas CASE representan una forma que permite modelar los procesos de negocios de las empresas y desarrollar sistemas de información. En un sentido más amplio las Herramientas CASE se pueden ver como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software. (6)

Visual Paradigm

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: negocio, requerimiento, análisis y diseño orientados a objetos, construcción, pruebas y despliegue. (6)

- Ayuda a una rápida construcción de aplicaciones de calidad con un menor coste; permitiendo dibujar todos los tipos de diagramas así como generar código y documentación.

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

- Proporciona abundantes tutoriales de UML, demostraciones interactivas y proyectos.
- Cuenta con la posibilidad de diseñar los esquemas correspondientes al modelo Entidad-Relación.
- Apoya un conjunto de lenguajes, tanto en generación de código como ingeniería inversa: Java, C ++, CORBA IDL, PHP, XML Schema, Ada y Python, VB.NET, Flash ActionScript y archivos de mapeo de Hibernate.

1.1.5. Plataforma de desarrollo

Java Platform Enterprise Edition (JEE) es un conjunto de tecnologías que reduce significativamente el coste y la complejidad de desarrollo, despliegue y gestión de ambiente, centradas en un servidor de aplicaciones y una aplicación de cualquier tamaño a desarrollar. Sobre la base de la Plataforma Java, Standard Edition (Java SE), JEE añade las capacidades que proporcionan una plataforma completa para el desarrollo sin contratiempos, que es a la vez estable, segura y rápida. (7)

1.1.6. Lenguaje de programación

Java es un lenguaje de programación que está diseñado tanto para Aplicaciones Web como de Escritorio. Fue desarrollado por la compañía Sun Microsystems en 1995 con el propósito de cubrir las necesidades tecnológicas de punta. Su principal característica y a su vez lo que lo diferencia ampliamente de los demás lenguajes de programación es la independencia que tiene a nivel de plataforma; esto se debe a que se le ha creado una máquina virtual para cada sistema operativo (SO). Lo cual significa que el desarrollador de software que programa en dicho lenguaje obvia la parte de preocuparse por el Sistema Operativo. Java fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. El desarrollo de aplicaciones se apoya en un gran número de clases preexistentes. De algunas de ellas las ha podido hacer el propio usuario, otras pueden ser comerciales, pero siempre hay un número muy importante de clases que forman parte del propio lenguaje. Al ser un lenguaje puro Orientado a Objetos le otorga gran reusabilidad y la independencia de la plataforma permite que los programas escritos en el lenguaje Java puedan ejecutarse igualmente en cualquier tipo de hardware, lo que significa ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo. Estos son algunos de los motivos por los cuales los desarrolladores de software preferentemente web optan por este lenguaje dada la necesidad de que en la web los clientes pueden tener diferente aficiones en sus ordenadores, o sea, para una misma aplicación web unos trabajarán en Windows, Linux, Macintosh, etc. (6)

1.1.7. Entorno de Desarrollo Integrado

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

A medida que ha avanzado el proyecto de modernización del SIIPOL, se han utilizado varios entornos de desarrollo para obtener los beneficios propios del impulso de la tecnología. En la actualidad se utiliza el Eclipse Helios.

El Eclipse es una plataforma de desarrollo de software multilenguaje que contiene un IDE y un sistema de plug-ins para extenderlo. Está escrito en Java y brinda la funcionalidad de programar en este lenguaje por defecto a través del plug-in básico JDT (Java Development Tools). Además de extender los lenguajes que puede soportar el IDE, el sistema de plug-ins permite adicionar otras funcionalidades útiles como editores visuales de distintos tipos de archivos, internacionalización, conexión con repositorios de control de versiones, etc. (8)

El Eclipse se distribuye bajo la Eclipse Public License, y es por tanto código abierto y software libre. Debido a este sistema, el IDE Eclipse es muy popular y existen plug-ins de muchos tipos para disimiles funciones. Las principales compañías detrás de los frameworks utilizados (fundamentalmente SpringSource y JBoss) proveen plug-ins que aceleran el desarrollo de software utilizando Eclipse como base. (8)

El SDK de Eclipse incluye por defecto el JDT, que ofrece un compilador incremental de Java y un modelo completo de archivos Java. Estas funcionalidades son muy importantes, dado que permiten aplicar técnicas avanzadas de refactorización y análisis de código.

La versión del IDE utilizada en el proyecto reúne un conjunto de plugins además de los provistos por defectos, como el WTP (permite desarrollo web, adicionando soporte para servidores), SpringIDE (para trabajo con los contexto de Spring), Subeclipse para integración con el control de versiones, etc. (8)

1.1.8. Frameworks Utilizados

Un ***framework*** (la traducción aproximada es marco de trabajo) es una estructura de soporte compuesta por componentes genéricos, personalizables, intercambiables y configurables para la organización de un proyecto de software. La solución que se desarrolla es parte de la modernización del SIIPOL por lo cual serán utilizados para esto los mismos frameworks ya definidos por el equipo de desarrollo para la implementación del sistema.

Capa de Presentación

Java Server Faces Framework o Java Server Faces (JSF) es un estándar creado por la SUN y su framework oficial para aplicaciones web basadas en Java, facilitando la construcción de interfaces de usuario (UI) para aplicaciones del lado del servidor con tecnología Java siguiendo el Patrón Modelo- Vista-Controlador. Diseñado para ser flexible, posee un modelo de componentes orientado a objetos, conversión de tipos de datos, separación de responsabilidades, desarrolladores de componentes, desarrolladores de lógica de aplicación, montadores de páginas, un poderoso sistema de navegación declarativa, uso de simples clases java como controladores, fácil incorporación de potencialidades AJAX. JSF provee un conjunto de librerías de tags personalizables para las JavaServer Pages (JSP) como interfaz entre JSF y la página JSP. (2)
Posee un grupo de bondades factibles para el desarrollador como son:

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

- Es más fácil de manejar que otros frameworks como Struts[5].

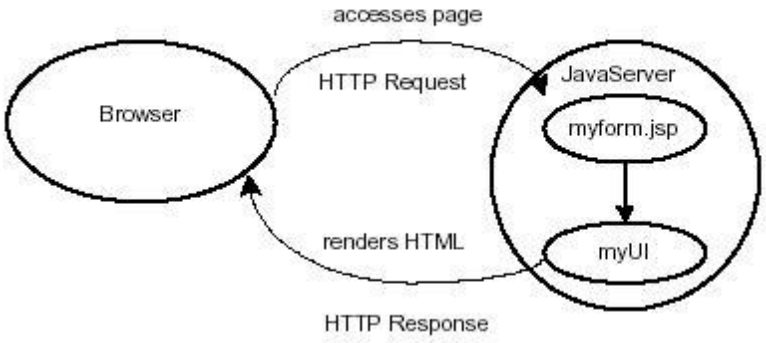


Ilustración 3. *Ciclo de vida bajo la tecnología JSF.*

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

Ajax4JSF es una extensión del estándar JSF que se integra a este con gran facilidad, evita escribir código Java Script al brindar una amplia gama de componentes, permite agregar capacidades AJAX incluso a aplicaciones JSF ya creadas. En fin, las prestaciones que brinda la poderosa combinación de JSF con AJAX, hacen que sea más atractiva por la comodidad que supone para el desarrollador y además por integrarse eficientemente con otros frameworks. (2)

[5] ***Struts*** es un *framework* de la capa de presentación que implementa el patrón MVC en Java Este separa muy bien la gestión del ciclo de vida de la aplicación, del modelo de objetos de negocio y de la generación de la interfaz de usuario (*Scribd*). Uno de los mayores aciertos que tuvo Struts fue reducir la repetición innecesaria de tareas y código común.

Capa de lógica de negocio

Spring es un framework de código abierto, es el más popular y el más ambicioso de todos los framework de peso ligero, interviene en todas las capas arquitectónicas de una aplicación JEE, brinda soporte a varios frameworks de presentación, entre ellos Java Server Faces (JSF), brinda soporte a Hibernate. (9)

Spring soporta la programación orientada a aspecto, técnica que complementa la programación orientada a objeto, logrando dividir el programa en preocupaciones separadas de la lógica del negocio y que se ejecutan de manera transversal a la aplicación; esto evita la dispersión de código o código enmarañando, posibilitando una mejor separación de conceptos y reduciendo la dependencia entre cada uno de los módulos. Otra de sus características al igual que los frameworks de peso ligero es que implementa el patrón inyección de dependencia mediante la técnica inversión de control, promoviendo el bajo acoplamiento entre los objetos relacionados. Spring provee un contexto apropiado para el desarrollo de aplicaciones web e integración con frameworks como hibernate (Acceso a Datos), IBATIS (Acceso a Datos), Struts (Presentación), JSF (Presentación), Tapestry (Presentación), JUnit (Test), Asegi (Seguridad). (6)

Capa de Acceso a Datos

La forma en que Java maneja las conexiones a una Base de datos es abriendo una conexión mediante la Interfaz de Programación de Aplicaciones (API, Application Programming Interface) de JDBC, y las consultas en Lenguaje de Consultas Estructurado (SQL, *Structured Query Language*); debido a que son objetos que se van a hacer persistentes en una base de datos o en ficheros a través del tiempo lo que provoca que este proceso sea altamente repetitivo y por lo tanto muy propenso a errores. Una aplicación con su modelo de dominio orientado a objetos no interactúa directamente con las columnas y tablas; por lo que se hizo uso de una forma muy elegante y novedosa de realizar las acciones sobre la base de datos a través de entidades encargadas de esta función, haciendo uso del Mapeo Relacional de Objetos (ORM). (2)

Hibernate es un Framework que utiliza Mapeo Objeto-Relacional (ORM, Object-Relational Mapping) para la relación entre el modelo objetual y el modelo relacional. Permite una fácil integración con cualquier otro Framework de Java y para su uso no requiere de muchas reglas específicas o patrones, lo que evita que se hagan grandes cambios en la arquitectura de la aplicación. Resuelve un gran problema de portabilidad dado el hecho que es muy configurable mediante el uso de XML, permitiendo configurar, entre otras cosas el dialecto del gestor de base de datos con el que se trabaja. Automatiza de forma eficiente el trabajo con

operaciones altamente repetitivas como crear, actualizar, consultar y eliminar. Permite trabajar con objetos persistentes haciendo uso de polimorfismo y herencias y relaciones y la mayoría de los tipos de datos que soporta Java. Es una solución no intrusiva, que no obliga a seguir determinadas reglas o patrones de diseño cuando se desarrolla la capa de lógica de negocio o las clases del dominio. Las clases persistentes no requieren implementar la interfaz serializable, aunque sí deben cumplir con la característica de tener un constructor sin parámetros. (2)

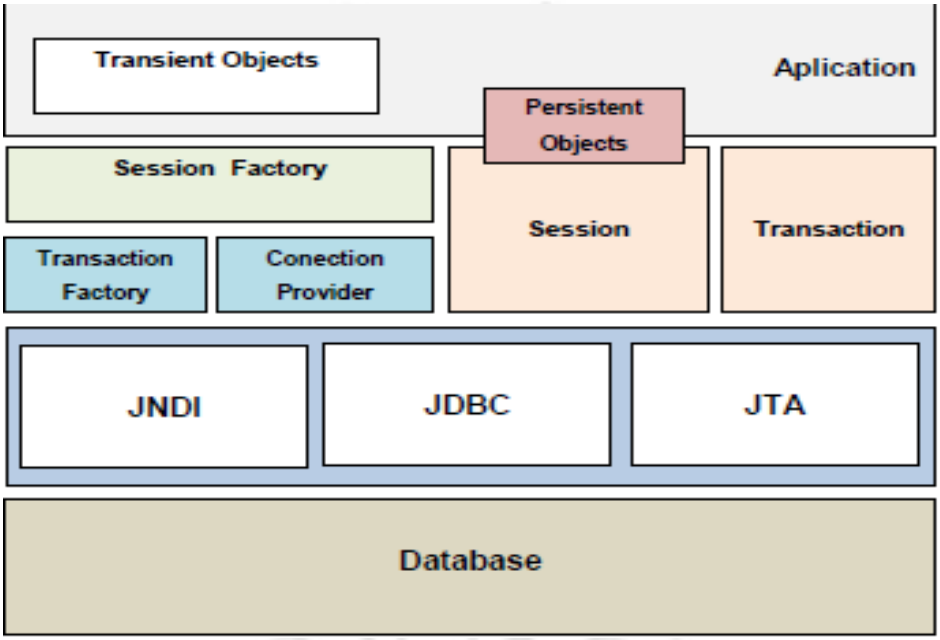


Ilustración 4. *Arquitectura por capas del Framework Hibernate.*

1.1.9. Sistema Gestor de base de datos

Oracle es un sistema de gestión de base de datos relacional fabricado por Oracle Corporation. Se considera uno de los sistemas de bases de datos más completos, destacando su soporte de transacciones, estabilidad, escalabilidad, y que es multiplataforma. Es un producto vendido a nivel mundial, aunque la gran potencia que tiene y su elevado precio hacen que, por norma general, sólo se use en empresas muy grandes y/o multinacionales.

1.2. Propuesta de solución

Después del estudio y análisis de la Metodología, lenguajes y herramientas de desarrollo antes expuesto, con el fin de cumplir todos los requisitos funcionales y no funcionales que la modernización del SIIPOL requiere, se ha decidido por parte de la dirección del proyecto construir una aplicación cliente-servidor donde las pc clientes se conectaran mediante protocolo HTTPS al servidor de aplicaciones y este a su vez al de base datos mediante protocolo TCP/IP. Dicha aplicación web será implementada en Java y haciendo uso de los frameworks JSF 1.2, Spring 3.05 e Hibernate 3.6 como entorno de desarrollo integrado se usará el Eclipse Helios y Visual Paradigm como herramienta CASE junto a UML para el modelado. Todo esto con RUP como metodología de desarrollo.

Ilustración 5. *Propuesta de Solución.*

Dicha aplicación será construida siguiendo el patrón arquitectónico n capas, donde cada capa solo conoce a su inmediata inferior.

Ilustración 6. *Patrón n capas.*

Conclusiones

Después de haber realizado un estudio de los sistemas de gestión policial similares al SIIPOL como son: STEGPOL y SIGEP, se detectó la ausencia de un subsistema o componente que lleve a cabo la gestión y procesamiento de las Transmisiones generadas en un departamento policial lo cual impide la reutilización de código o la reutilización de componentes y nos obliga a la implementación de un subsistema que se encargue de automatizar todos estos procesos que sería el módulo de Transmisiones, cuya solución sería integrada al SIIPOL .

Debido al estudio previo realizado por parte de los directivos del proyecto en el que se ha llevado a cabo un análisis detallado de las tecnologías y herramientas más usadas en el campo de la informática expuestas anteriormente, se opta por la decisión de desarrollar una aplicación web, sobre el lenguaje Java, con la integración de los frameworks Java Server Faces (JSF), Spring e Hibernate, lo cual daría como ventaja velocidad de desarrollo, además del uso del patrón n capas con tres niveles, facilitando el desarrollo del producto y disminuyendo el acoplamiento. Para lograr tal resultado se propone por parte de la dirección del proyecto el uso de las herramientas Eclipse Helios y Visual Paradigm por las facilidades que estas brindan. El proceso como tal será orientado por la metodología RUP, la cual constituye una guía de cómo se debe desarrollar una aplicación de tal escala.

CAPÍTULO 2. Diseño e Implementación de la Propuesta de Sistema.

Introducción

En este capítulo se abordarán los temas referentes al diseño e implementación de la propuesta de solución del módulo Transmisiones, para ello se comenzará por una descripción de los casos de usos que resultaron de la aplicación de la Ingeniería de Requerimientos y que constituyen la entrada para realizar el análisis, diseño e implementación. Además se explicarán temas relacionados a cuestiones propias del sistema, que se adaptaron a la arquitectura definida por los arquitectos del proyecto.

2.1. Diseño

En el diseño se modela el sistema y encontramos su forma (incluida la arquitectura) para que soporte todos los requisitos incluyendo los requisitos no funcionales y otras restricciones que se le suponen. Una entrada esencial en el diseño es el resultado del análisis. El modelo de análisis proporciona una comprensión detallada de los requisitos, y lo que es más importante, impone una estructura del sistema que debe conservar lo más fielmente posible cuando demos forma al sistema. En concreto, los propósitos del diseño son: (4)

- Adquirir una comprensión a profundidad de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables sistemas operativos, tecnologías de distribución y concurrencia.
- Crear una entrada apropiada y un punto de partida para las actividades de implementación subsiguientes capturando los requisitos o subsistemas individuales. (4)

2.1.1. Modelo de Diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de usos 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 como abstracción de la implementación del sistema y es, de este modo, utilizada como una entrada fundamental de las actividades de implementación. (4)

2.1.2. Descripción de la Arquitectura del Sistema

En un software es muy importante el uso de una arquitectura de software que asegure el cumplimiento de todos los requisitos ya sean funcionales y no funcionales de la forma más óptima posible. En una aplicación del alcance que se requiere para cubrir los requisitos del SIIPOL, que cuenta numerosas clases, se ha de contar con el empleo de patrones para garantizar un diseño claro y lo más simple posible. El módulo Transmisiones como parte de este sistema utiliza e implementa patrones de diseño heredados de los *frameworks* utilizados y otros impuestos por la arquitectura del proyecto de los cuales mencionaremos a continuación los más usados.

Patrón Facade: Este es un patrón de diseño clásico que se ocupa de separar la lógica de una operación del código cliente, interfiriendo en sus llamadas mediante una fachada o clase abstracta que es lo único visible al exterior del componente (11). Este patrón se usa en la capa de negocio creando un punto de acceso único al módulo Transmisiones.

DAO: Concentran la lógica especial de persistencia de las entidades del dominio de la aplicación. Los DAO contienen todas las sentencias de interacción de la aplicación con la base de datos, así como su transformación para la presentación. (8) Este se utiliza en la capa de acceso a datos, con el fin de provocar el menor cambio posible en las clases de la solución de la solución, cuando se realice un cambio en la fuente de los datos.

Patrón Domain Model: Este patrón descrito por Fowler consiste básicamente en usar las técnicas de diseño orientado a objetos para modelar mediante clases el dominio del problema. Un modelo de dominio consiste en una trama de objetos interconectados, una gran parte de los cuales presentan estado y comportamiento y cada uno de ellos modela un concepto del problema. El Modelo de Dominio es una vía para enfrentar con éxito negocios complejos y constituye el núcleo de la capa de negocio y de todo el

sistema. La implementación de dicho Modelo de Dominio usa POJOs para permitir el uso más sencillo de los *frameworks* elegidos y la migración de estos a versiones superiores a medida que el proyecto avance, así como la introducción de nuevas tecnologías y el desfasaje de las obsoletas. (8) Este patrón se ve en las clases persistentes que tiene el módulo Transmisiones las cuales encapsula el dominio del problema.

Patrón Alta Cohesión: más que un diseño directamente implementable en código, se trata de un principio que nos guiará en el diseño, es un objetivo subyacente a tener en cuenta continuamente. Es un principio evaluativo que aplica un diseñador mientras evalúa todas las decisiones de diseño. Se puede medir el nivel de cohesión en una clase cuanto más enfocado sea su comportamiento. Cada elemento de nuestro diseño debe realizar una labor única dentro del sistema, que no se desempeñe en ninguno de los de los demás elementos del sistema. (12)
Por otra parte pensar en interfaces nos fuerza a que nuestros sistemas sigan los principios de alta cohesión. Un diseño cohesionado facilita el cambio (objetivo principal de todos los patrones de diseño). Al realizar un cambio en una clase muy cohesionada, todos los métodos que pueden verse afectados, toda la información que necesitamos controlar, estará a la vista, en el mismo fichero.

Patrón Bajo Acoplamiento: le da respuesta a la problemática de soportar bajas dependencias, bajo impacto del cambio e incremento de la reutilización. El acoplamiento es una medida de la fuerza con que un elemento está conectado a, tiene conocimiento de, confía en, otros elementos. Un elemento con bajo (o débil) acoplamiento no depende de demasiados otros elementos. Estos elementos pueden ser clases, subsistemas, sistemas entre otros, de ahí la importancia que se lleve a cabo el desempeño de este patrón, para de esta manera obtener una aplicación lo más flexible posible. (12)

Patrón Experto: no es más que asignar una responsabilidad al experto en información, la clase que tiene la información necesaria para realizar la responsabilidad. (12) Este patrón muy utilizado a la hora de modelar las entidades persistentes de la solución donde las funcionalidades propias de cada información re implementan en la clase que contiene toda la información o sea la clase experta en la información.

Composite view: Un objeto vista que está compuesto de otros objetos vista, es usado por ejemplo cuando se quiere incluir un subview.jsp dentro de una página .jsp. (Sánchez, 2009) Utilizado en las interfaces de usuario donde se descomponen las interfaces complejas y redundantes en una vista la cual es incluida donde se hace necesario su uso.

Bridge (Puente): Desacopla una abstracción de su implementación. Es el principio seguido para todas las implementaciones con respecto a las interfaces de la presente solución. (Sánchez, 2009)

A continuación mostramos una serie de patrones heredados o impuestos por los frameworks utilizados:

Patrón MVC: El MVC es un patrón arquitectónico reconocido y descrito en uno de los sistemas de clasificación de patrones de arquitectura más extendidos en el mundo: Pattern of Enterprise Application Architecture, Este sistema de patrones fue propuesto por Martin Fowler en el 2003, y en este el MVC es ubicado en la categoría de Patrones de Presentación Web (Web Presentation Patterns). El MVC divide una aplicación en 3 componentes: Modelo, Vista y Controlador. (8) Este patrón es implementado por el framework de la capa de presentación (JSF).

Observer: Define una dependencia de uno a muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él. (Sánchez, 2009)

Front Controller: Un objeto que acepta todos las peticiones de un cliente y los direcciona a manejadores apropiados, es usado en la capa de presentación. (12) También implementado por el framework de la capa de presentación (JSF).

Patrón Controlador: aumenta el potencial de reutilización, y asegura que la lógica de la aplicación no se maneja en la capa de interfaz. Un Controlador es un objeto que no pertenece a la interfaz de usuario, responsable de recibir o manejar un evento del sistema y define el método para la operación del sistema. (12) Este patrón aunque es definido por el framework de presentación es implementado por los programadores del proyecto cuando atienden las peticiones del cliente en los beanes manejados.

Abstract Factory (Fábrica abstracta): Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí, es usado por el framework spring. (Sánchez, 2009)

Builder (Constructor virtual): Abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto, es usado por el framework spring. (12)

La aplicación define varios roles para familias específicas de elementos. Cada una de estas familias o roles, se encarga de tareas particulares, se encuentra en lugares específicos y se nombra de una forma específica. Algunas de estas clases serían las siguientes: (11)

Tipo de Clase	Función
Beans Manejados	Manejan la lógica por detrás de las páginas de presentación, o sea, todo lo referente al formateo de datos para la página
Convertidores	Un convertidor es un elemento cuya función consiste en convertir la entrada de datos de la interface a formas más legibles para la aplicación, y viceversa.
Validadores	EL objetivo de un validador es asegurarse de que la información provista por el usuario es válida según las reglas de negocio reflejadas en el sistema.
Fachadas	Las fachadas son los elementos encargados de proveer el punto básico de aplicación de aspectos, transacciones de la aplicación. Además sirven para separar la lógica de cada módulo de las llamadas de los demás.
Servicios	Los servicios son las clases encargadas de llevar a cabo cualquier lógica de negocio especialmente difícil o complicado o largo, que no tenga que ver directamente con el acceso a datos o la presentación.
Daos	Encargados de manejar la persistencia de una entidad cualquiera, tanto para leer como para escribir en la BD
Entidades	Las entidades son el mecanismo básico de movimiento e información dentro de la aplicación. Las entidades son leídas de la BD por los Daos y

	luego se mueven por todas las capas de la aplicación hasta la presentación.
Nomencladores	Esta es una clase particular del sistema, que representa el conjunto de valores propios de los conceptos asociados a las entidades del sistema. Es especial debido a que unifica todos los conceptos que pueden modelarse como clasificadores. (8)
Útiles	Estas clases son colecciones de métodos que encapsulan lógica útil para varios lugares de la aplicación. Funcionan como librerías de funcionalidades en vez de librerías de componentes

También se definió una arquitectura en paquetes para la organización de las clases dentro de los paquetes de la aplicación quedando la siguiente estructura:

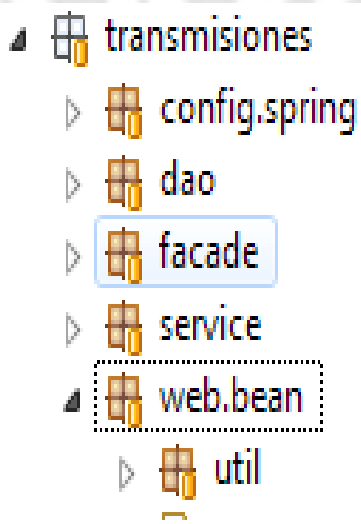


Ilustración 7.*Arquitectura Modular del sistema.*

La definición de arquitectura no es sencilla, dado que no existe consenso absoluto sobre este concepto a nivel mundial. Para dar basamento conceptual a las actividades concebidas dentro del marco de la actividad arquitectónica del proyecto SIIPOL, la dirección del proyecto se basó en el concepto siguiente:

"Arquitectura es la organización estructural de un sistema, representada por sus componentes, las relaciones entre los mismos, el ambiente y los principios que gobiernan el diseño y su evolución. La arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. " (8)

La arquitectura definida, ha sido comprobada durante los años de desarrollo del SIIPOL y ha soportado todos los requisitos funcionales y no funcionales del sistema, manteniendo sus principales principios de fiabilidad, integridad de los datos, seguridad, rendimiento etc.

2.1.3. Diagrama de clases del diseño

Los diagramas de clase muestran la estructura estática del sistema, especialmente, lo que existe como clases, su estructura interna y sus relaciones con otras clases. Los diagramas de clase no muestran información temporal. Un diagrama de clase se presenta como una recopilación de elementos de modelo declarativo (estático), como clases, paquetes y sus relaciones, conectados como gráficos entre si y a su contenido. Los diagramas de clase se pueden organizar en (y ser propiedad de) paquetes, mostrando sólo lo que es relevante en un paquete concreto. (5)

A continuación se muestran los diagramas de las clases relevantes creadas y utilizadas en la solución propuesta:

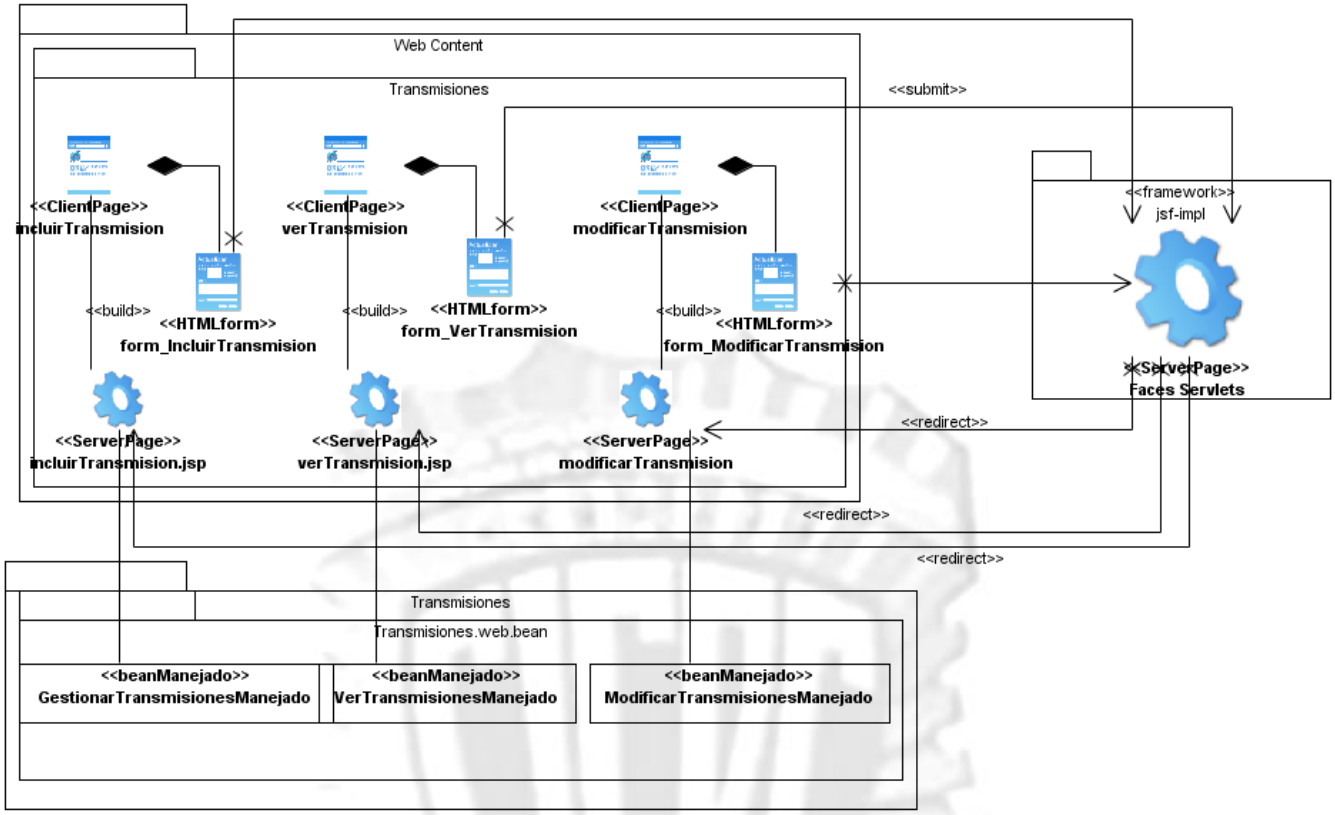


Ilustración 8.*Diseño. Gestionar Transmisiones parte 1 .Diagrama de Clases*

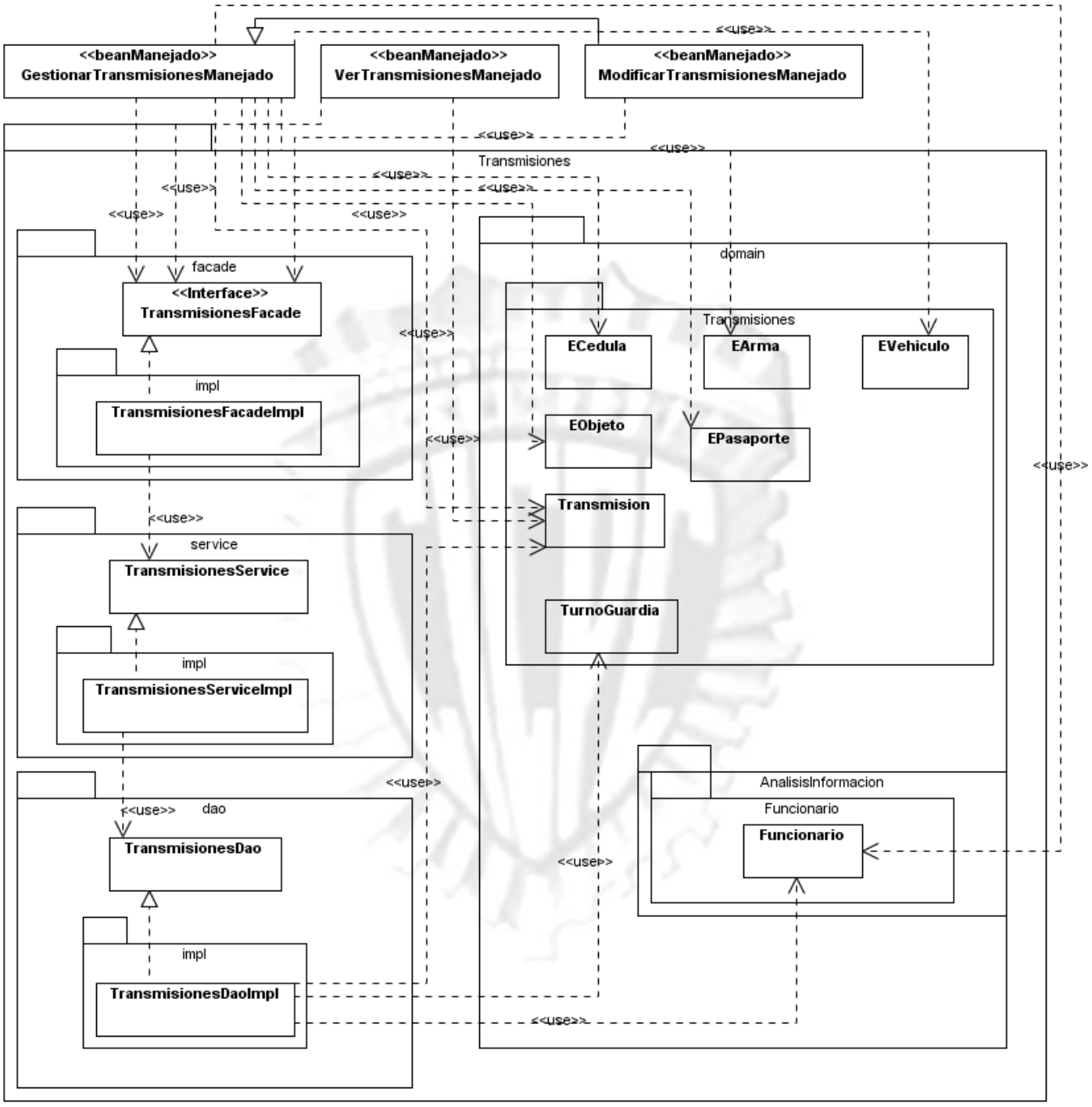


Ilustración 9.*Diseño. Gestionar Transmisiones parte 2 .Diagrama de Clases*

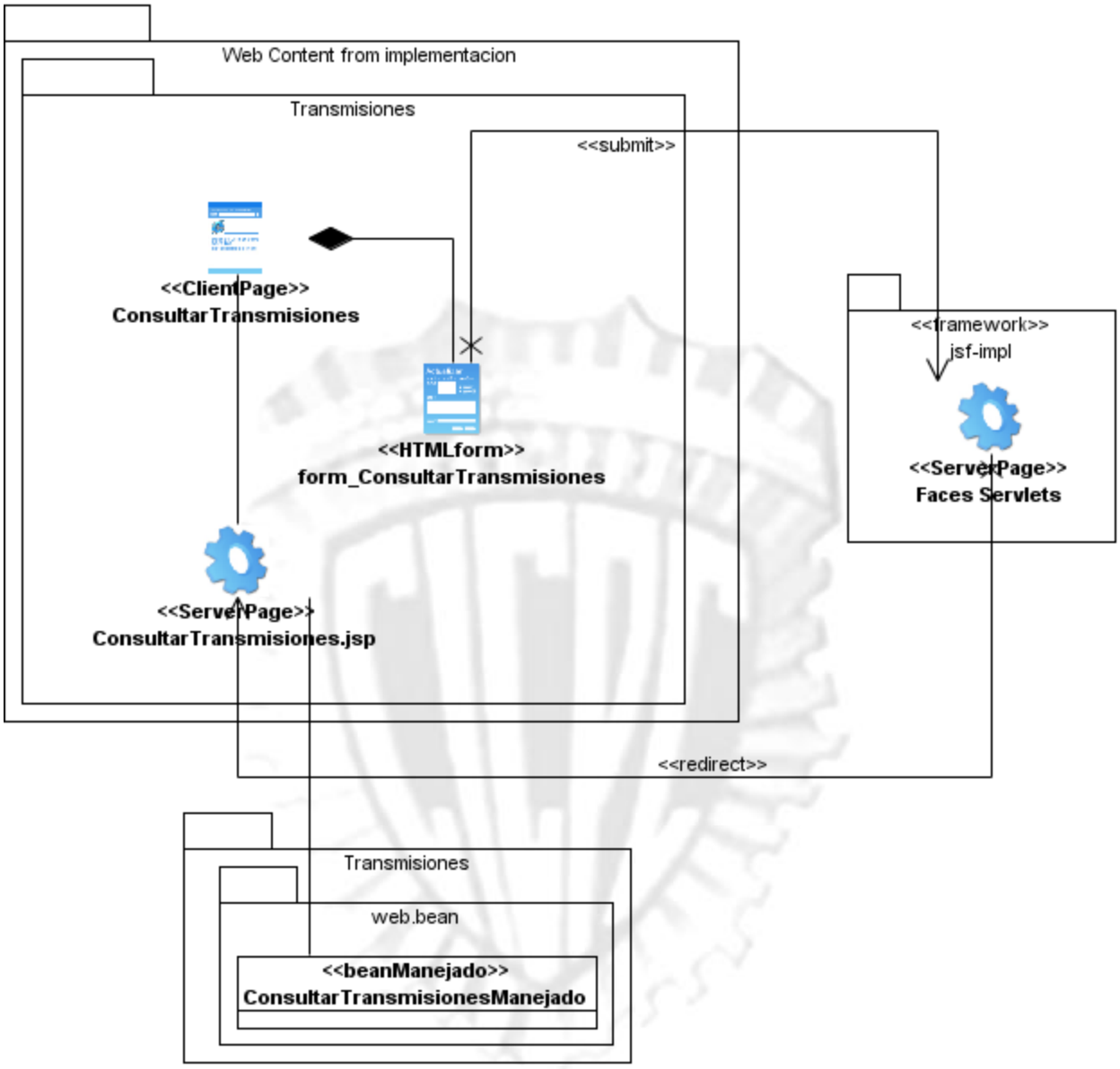


Ilustración 10.*Diseño. Consultar Transmisiones parte 1 .Diagrama de Clases*

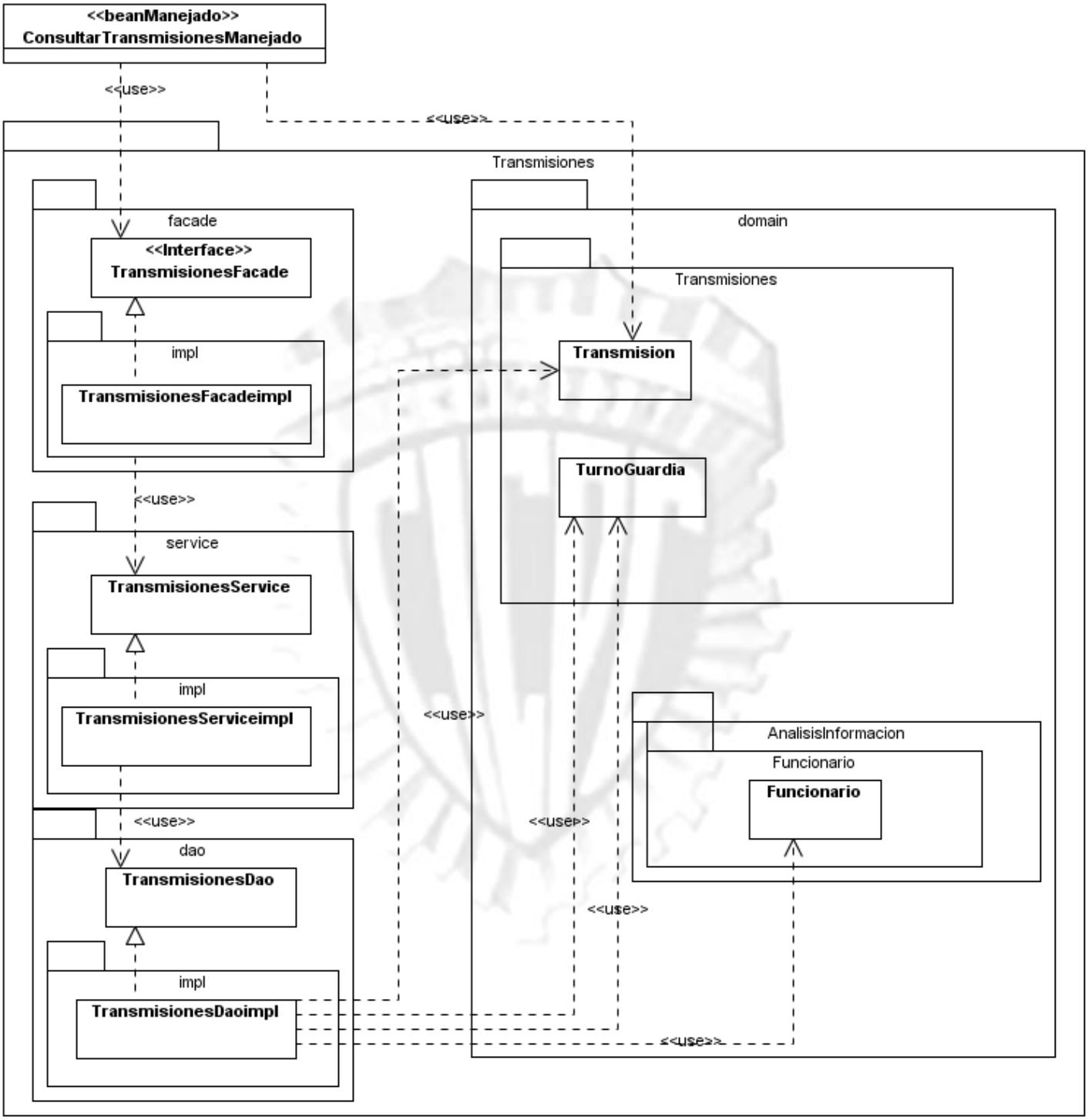


Ilustración 11. *Diseño. Consultar Transmisiones parte 2 .Diagrama de Clases*

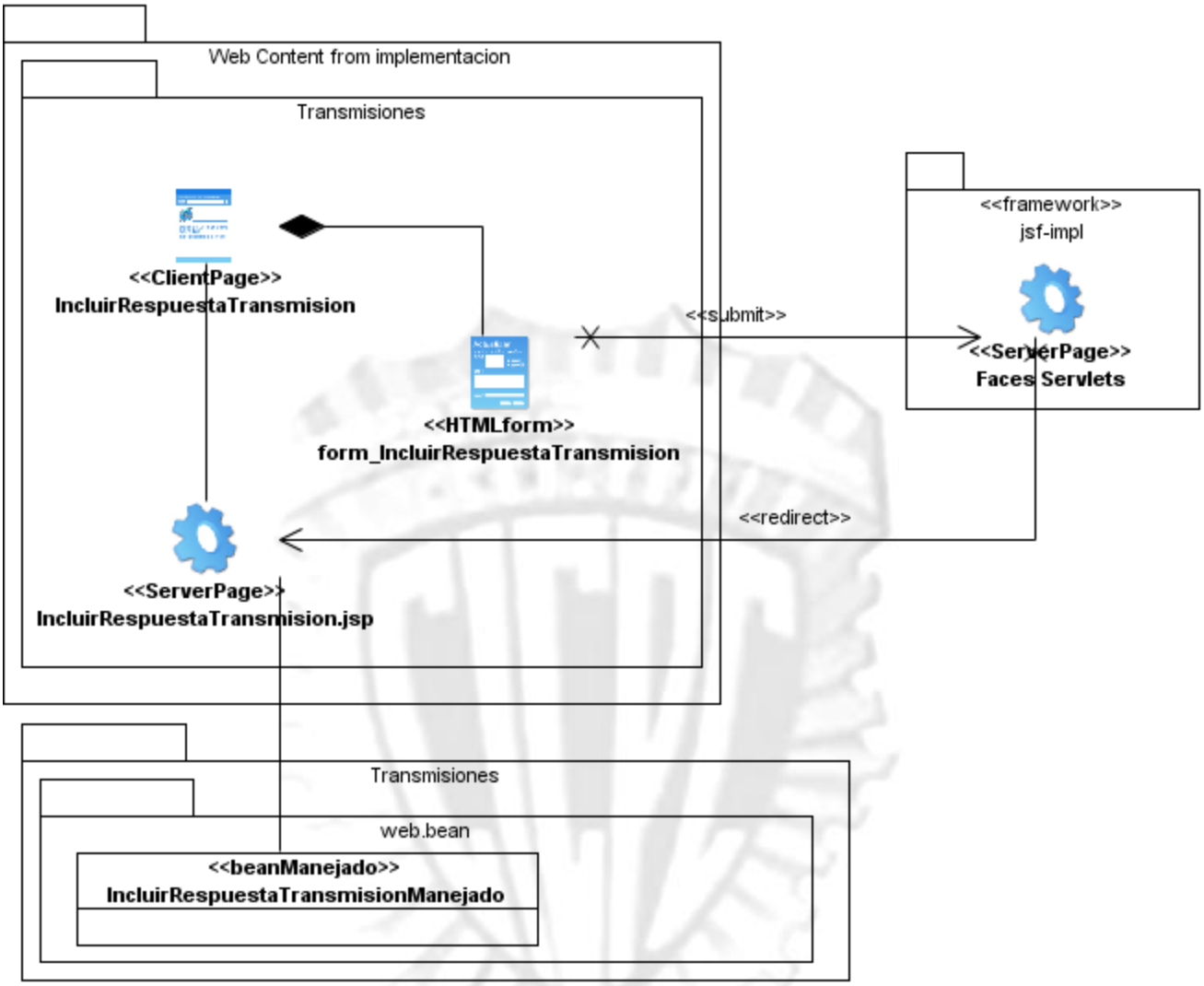


Ilustración 12. *Diseño. Incluir Respuesta de Dependencia parte 1 .Diagrama de Clases*

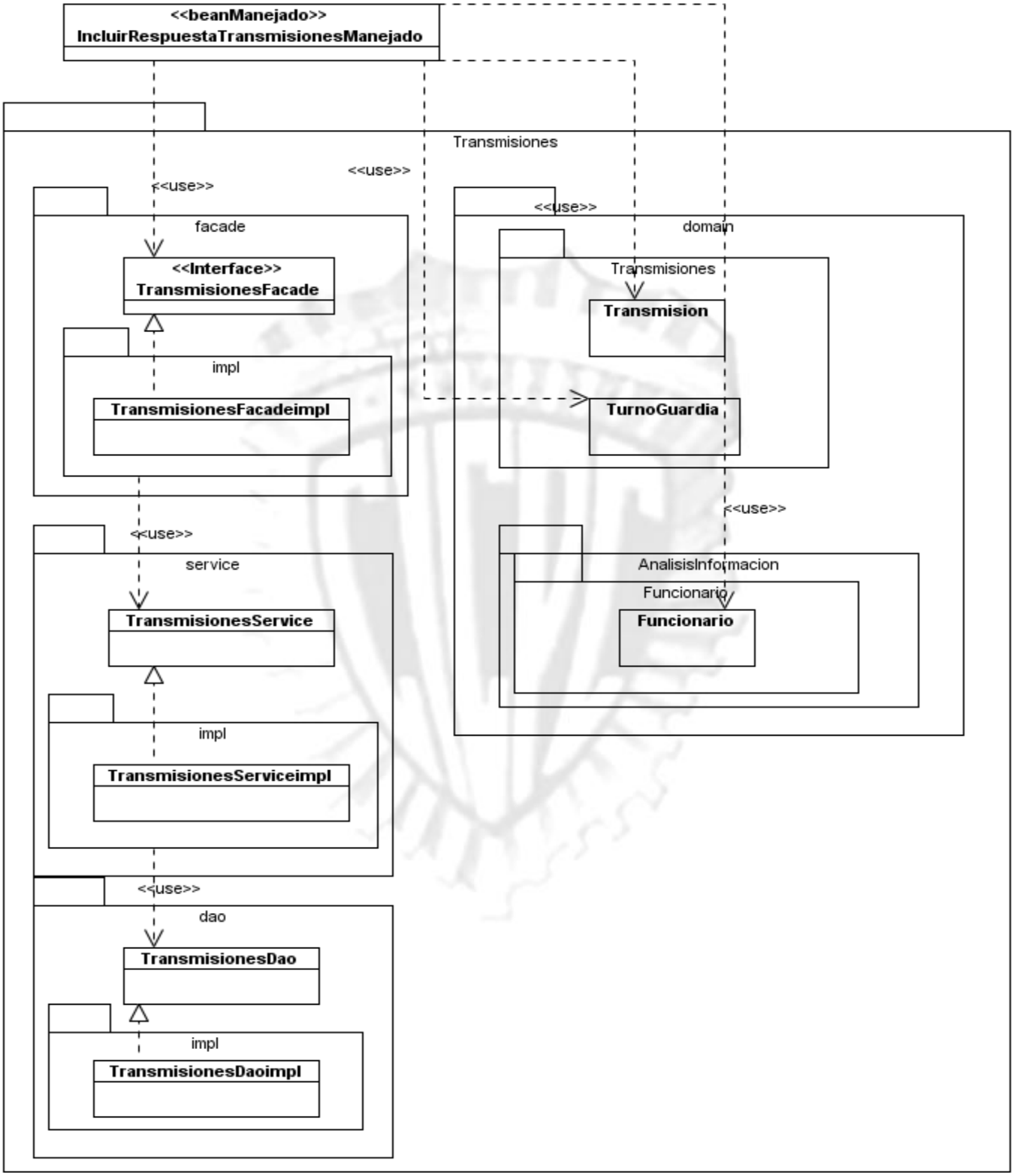

<<beanManejado>>
IncluirRespuestaTransmisionesManejado
Transmisiones
<<use>>
<<use>>
<<use>>
facade
<<Interface>>
TransmisionesFacade
impl
TransmisionesFacadeimpl
domain
Transmisiones
Transmision
TurnoGuardia
<<use>>
AnalisisInformacion
Funcionario
Funcionario
<<use>>
service
TransmisionesService
impl
TransmisionesServiceimpl
<<use>>
dao
TransmisionesDao
impl
TransmisionesDaoimpl

Ilustración 13.*Diseño. Incluir Respuesta de Dependencia parte 2.Diagrama de Clases*

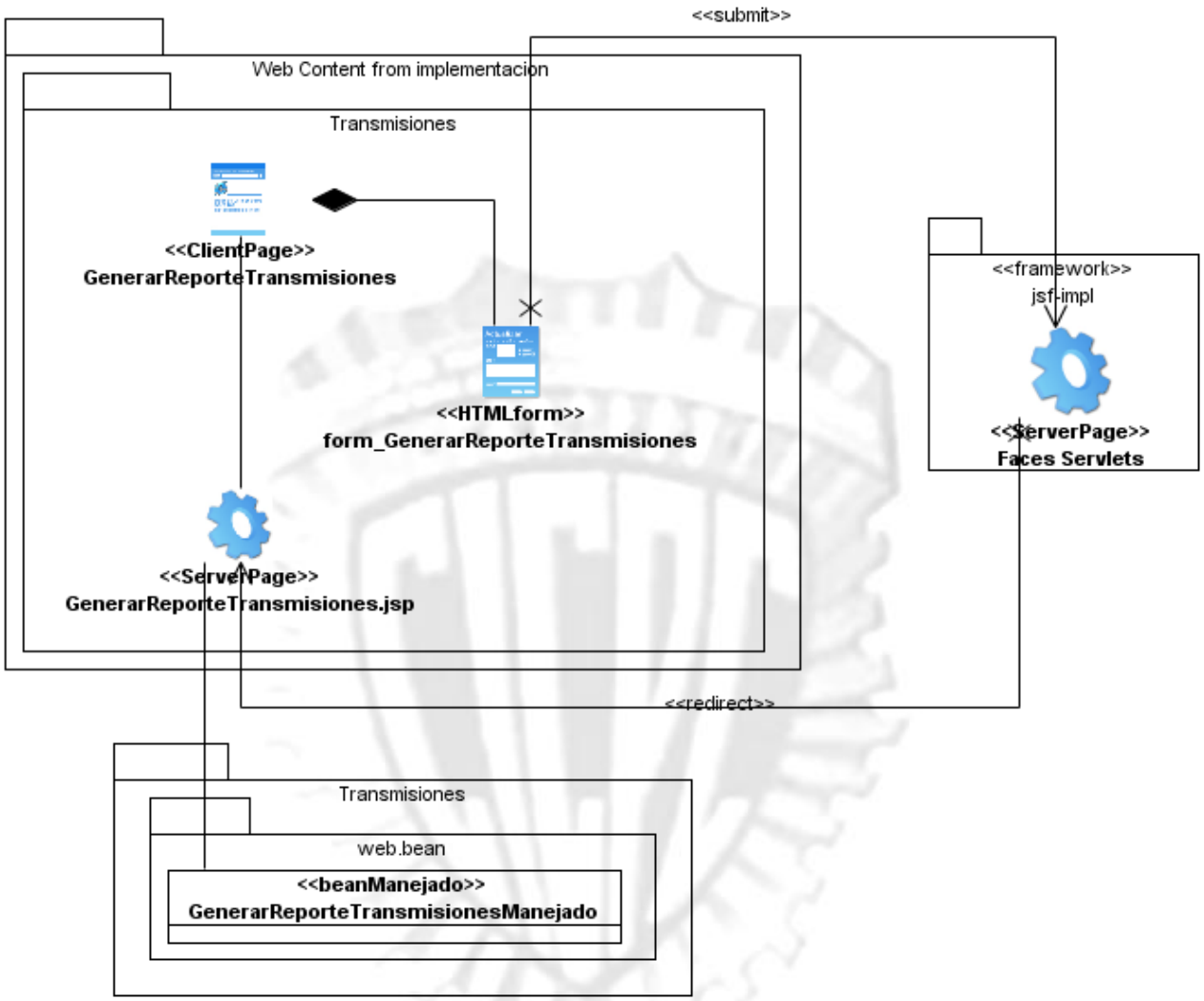


Ilustración 14.*Diseño. Generar Reporte por Turnos de Guardia parte 1.Diagrama de Clases*

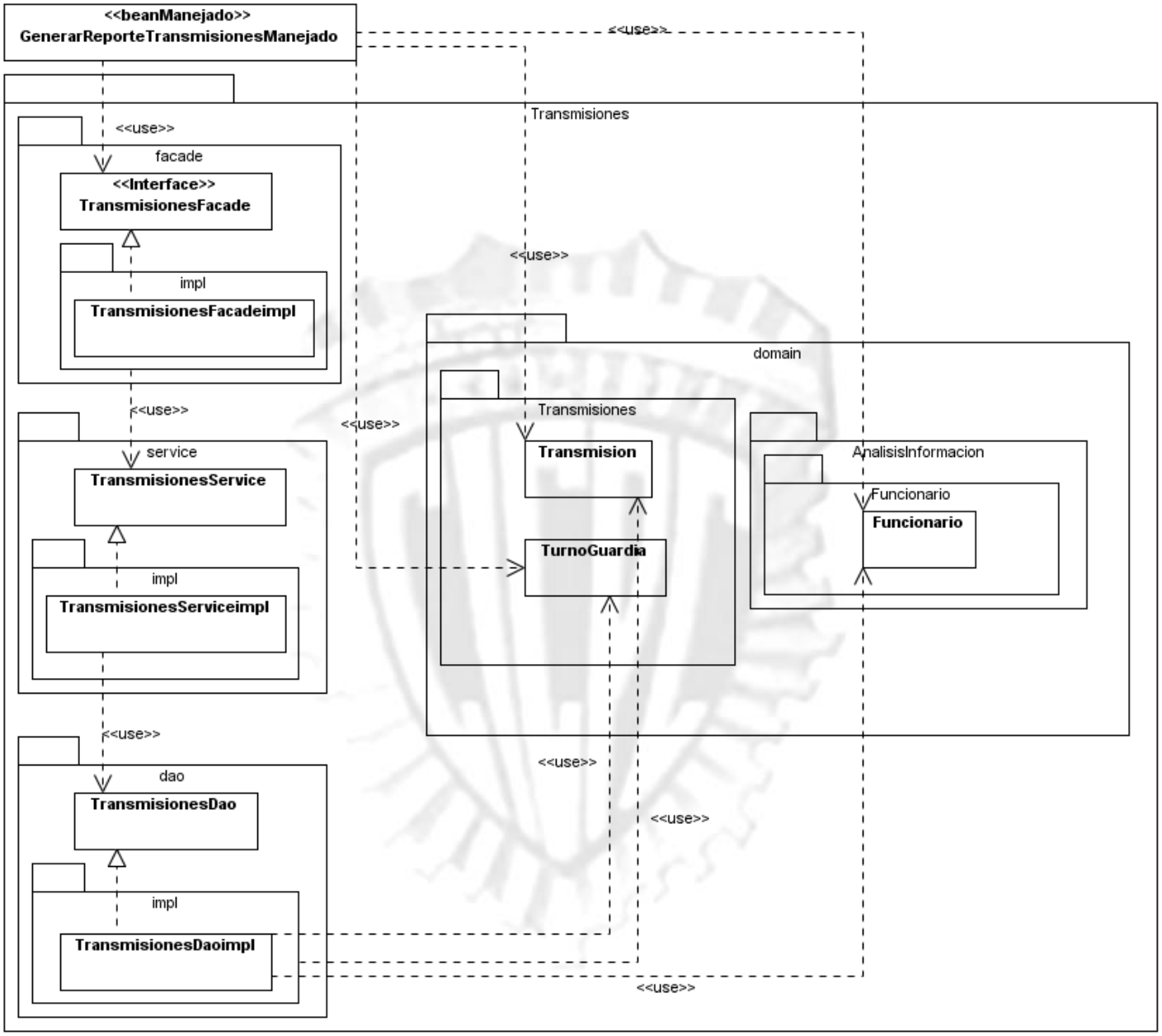


Ilustración 15.*Diseño. Generar Reporte por Turnos de Guardia parte 1.Diagrama de Clases*

2.1.4. Clases significativas propias de la solución

Después de haber dado una breve reseña de los patrones más importantes que se usaron para la realización del diseño de la solución y la descripción de la arquitectura definida se dará una descripción de las clases significativas para cada caso de uso.

Nombre: Transmision	
Tipo de clase: Entidad Persistente	
Atributo	***Tipo***
fechaHora	Date
fechaCreacion	Date
tipoTransmision	Nomenclador
sitioSuceso	Direccion
estadoTransmision	Nomenclador
turnoGuardia	TurnoGuardia
funcCreador	Funcionario
fechaFinalizacion	Date
numero	String
observaciones	String
Para cada responsabilidad:	

Nombre: TurnoGuardia	
Tipo de clase: Entidad Persistente	
Atributo	***Tipo***
fechaInicial	Date

fechaTerminacion	Date
funcCreador	Funcionario
funcJefe	Funcionario
anotaciones	String
dependenciaInterna	DependenciaInterna
fecha	Date
integrantes	Set<Funcionario>
Para cada responsabilidad:	
Nombre:	toString
Descripción:	Retorna un String que identifica el turno de guardia
Nombre:	equals
Descripción:	Compara el objeto pasado por parámetro y retorna falso o verdadero

Nombre: DependenciaRespuesta	
Tipo de clase: Entidad Persistente	
Atributo	***Tipo***
respuesta	String
dependencia	DependenciaInterna
funcionarioLogin	Funcionario
funcReceptor	Funcionario
fechaCreacion	Date
Para cada responsabilidad:	
Nombre:	toString
Descripción:	Retorna un String que identifica la Dependencia que respondió
Nombre:	equals
Descripción:	Compara el objeto pasado por parámetro y retorna falso o verdadero

Nombre: ElementoVerficar	
Tipo de clase: Entidad Persistente	
Atributo	***Tipo***
elementoVerificar	Nomenclador
estadoVerificar	Nomenclador
Para cada responsabilidad:	
Nombre:	toString
Descripción:	Retorna un String que identifica el Elemento Verificado
Nombre:	equals
Descripción:	Compara el objeto pasado por parámetro y retorna falso o verdadero

Nombre: TransmisionesManejado	
Tipo de clase: Controladora	
Atributo	***Tipo***
cedula	ECedula
arma	EArma
pasaporte	EPasaporte
vehiculo	EVehiculo
objeto	EObjeto
tipotransmision	Nomenclador
infoCiudadano	PersonaExterna
obsJustificacion	String
causa	Nomenclador
cantCausa	Integer
cantFuncMuertos	Integer

cantFuncOPoliciasMuertos	Integer
cantCiudadanosMuertos	Integer
showCantFuncMuertos	boolean
showCantFuncOPoliciasMuertos	boolean
showCantCiudadanosMuertos	boolean
numeroValidado	boolean
letraCaso	String
noCaso	String
tipoSuceso	Nomenclador
tipoInformante	Nomenclador
horaRecepcion	Date
estadoVerificar	Nomenclador
elementoVerificar	Nomenclador
validElement	boolean
horaSalida	Date
portatil	String
unidad	String
despachoUnidadOperativa	DependenciaInterna
tripulacion	Funcionario
direccionSuceso	Direccion
direccionInspeccion	Direccion
datoInformanteTipo	String
nombreInstitucionInfo	String

credencial	String
datoInformanteDespacho	DependenciaInterna
datoInformanteFuncionario	Funcionario
datoFuncionario	Funcionario
despachosANotificar	DependenciaInterna
tipoEntidadNTRobo	Nomenclador
otrosDatosEntidadNTRobo	String
otrosDatosInformante	String
observaciones	String
pasaporteValidar	String
letraCedula	String
numeroCedulaValidar	String
numeroPlacaValidar	String
serialCarroceriaValidar	String
serialPrimarioValidar	String
serialSecundarioValidar	String
serialTerciarioValidar	String
serialObjeto	String
listEVehiculosVerificados	List<EVehiculo>
listECedulasVerificadas	List<ECedula>
listEPasaportesVerificados	List<EPasaporte>
listEArmasVerificadas	List<EArma>
listEObjetoVeri	List<EObjeto>

listTipotransmisiones	List<SelectItem>
listCausaMuertes	List<SelectItem>
despachos	List<SelectItem>
listTiposSuceso	List<SelectItem>
tipoEntidades	List<SelectItem>
listEstadoVerificar	List<SelectItem>
listElementosVerificar	List<SelectItem>
listTiposInformantes	List<SelectItem>
funcionariosDadoDependencia	List<SelectItem>
funcionariosDadoDependenciaUnidadOperativa	List<SelectItem>
listaMuertesCusa	List<MuerteCausa>
listaDelegacionesNotificadas	List<DependenciaRespuesta>
listFuncionariosTripulacion	List<Funcionario>
listFuncionarioMarked	List<SelectableItemDTO<Funcionario>>
informanteCredencial	String
vehiculoArea	Vehiculo
igestionarVehiculo	IGestionarVehiculoManejado

Para cada responsabilidad:

Nombre:	verfMuertosInputs
Descripción:	Verifica que la cantidad de muertos que se introducen no superen la cantidad de los muertos notificados.
Nombre:	cancelarTransmision
Descripción:	Cancela la operación y envía al escritorio de trabajo

Nombre:	auxAddMuertesCausas
Descripción:	Adiciona a la lista de causas de muerte cuantos muertos se debieron a dicha causa.
Nombre:	addMuertesCausa
Descripción:	Se agregan las causas y cantidad de muertes a la tabla
Nombre:	addAreasDelegacionesNotificacion
Descripción:	Adiciona a la tabla Áreas Delegaciones Notificadas
Nombre:	validarElementos
Descripción:	Valida los elementos para el tipo de Transmisión
Nombre:	addElementos
Descripción:	Adiciona elementos a la lista de elementos verificados
Nombre:	deleteElementosPer
Descripción:	Elimina elementos de la lista de elementos verificados
Nombre:	seleccionarPersona
Descripción:	Gestiona el tipo de informante ciudadano
Nombre:	adicionarTripulacion
Descripción:	Agrega la Tripulación a la Transmisión de control de unidades en el área
Nombre:	eliminarTripulacion
Descripción:	Elimina la Tripulación de la Transmisión de Control de Unidades en el Área
Nombre:	incluirTransmision
Descripción:	Incluye una transmisión
Nombre:	buscarFuncionarioInformantePorCredencial
Descripción:	Busca un funcionario por credencial, se usa para asociar un funcionario como informante

	de la Transmisión.
Nombre:	validarActaProcesal
Descripción:	Valida el Acta Procesal
Nombre:	cancelarActaProcesal
Descripción:	Cancelar el Acta Procesal

Nombre: GenerarReporteTransmisionesManejado	
Tipo de clase: Controladora	
Atributo	***Tipo***
fechaInicial	Date
fechaFinal	Date
turnoGuardia	TurnoGuardia
listaTransmisiones	List<Transmision>
selectItemTurnoGuardia	List<SelectItem>
Para cada responsabilidad:	
Nombre:	consultar
Descripción:	Consulta los reporte de Transmisiones
Nombre:	nuevaBusqueda
Descripción:	Realiza una nueva búsqueda de las Transmisiones
Nombre:	buscarFuncionariosGuardiaRango
Descripción:	Busca los funcionarios de guardia

Nombre: IncluirRespuestaTransmisionManejado	
Tipo de clase: Controladora	
Atributo	***Tipo***
respuesta	String
anotaciones	String
urlVer	String
estadoTransmision	Nomenclador
dependeciaRespp	DependenciaRespuesta
transmision	Transmision
dependenciasRespuestas	List<SelectItem>
Para cada responsabilidad:	
Nombre:	modificar
Descripción:	Modifica la respuesta de Transmisión

Nombre: TransmisionesFacadeImpl	
Tipo de clase: Controladora	
Atributo	***Tipo***
transmisionesService	TransmisionesService
Para cada responsabilidad:	
Nombre:	salvarTurnoGuadia
Descripción:	Salva un tuno de guardia si no existen turnos registrados con anterioridad que coincidan con el horario del turno
Nombre:	consultarTurnoGuardia

Descripción:	Consulta un Turno Guardia de acuerdo con los criterios dados
Nombre:	actualizarTurnoGuadia
Descripción:	Actualiza un Turno de Guardia
Nombre:	consultarTransmisiones
Descripción:	Consulta una Transmisión de acuerdo con los criterios dados
Nombre:	salvarTransmision
Descripción:	Salva una Transmisión
Nombre:	actualizarTransmision
Descripción:	Actualiza una Transmisión
Nombre:	buscarTurnosDadoRangoFecha
Descripción:	Busca los Turnos de Guardia que se encuentran en un rango de fecha determinado

Nombre: TransmisionesServiceImpl	
Tipo de clase: Controladora	
Atributo	***Tipo***
transmisionesDao	TransmisionesDAO
correspondenciaFacade	CorrespondenciaFacade
Para cada responsabilidad:	
Nombre:	salvarTurnoGuadia
Descripción:	Salva un tuno de guardia si no existen turnos registrados con anterioridad que coincidan con el horario del turno
Nombre:	consultarTurnoGuardia

Descripción:	Consulta un Turno Guardia de acuerdo con los criterios dados
Nombre:	actualizarTurnoGuadia
Descripción:	Actualiza un Turno de Guardia
Nombre:	consultarTransmisiones
Descripción:	Consulta una Transmisión de acuerdo con los criterios dados
Nombre:	salvarTransmision
Descripción:	Salva una Transmisión
Nombre:	actualizarTransmision
Descripción:	Actualiza una Transmisión
Nombre:	buscarTurnosDadoRangoFecha
Descripción:	Busca los Turnos de Guardia que se encuentran en un rango de fecha determinado

Nombre: TransmisionesDaoImpl	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	salvarTurnoGuadia
Descripción:	Salva un tuno de guardia si no existen turnos registrados con anterioridad que coincidan con el horario del turno
Nombre:	consultarTurnoGuardia
Descripción:	Consulta un Turno Guardia de acuerdo con los criterios dados
Nombre:	actualizarTurnoGuadia
Descripción:	Actualiza un Turno de Guardia

Nombre:	consultarTransmisiones
Descripción:	Consulta una Transmisión de acuerdo con los criterios dados
Nombre:	salvarTransmision
Descripción:	Salva una Transmisión
Nombre:	actualizarTransmision
Descripción:	Actualiza una Transmisión
Nombre:	listarTurnosDadoFechas
Descripción:	Busca los Turnos de Guardia que se encuentran en un rango de fecha determinado
Nombre:	obtenertTurnoGuardiaVigente
Descripción:	Obtiene el Turno de Guardia que se encuentra vigente si existe

2.1.5. Realizaciones de casos de uso

Una realización de caso de uso del diseño es una colaboración del modelo de diseño que describe cómo se realiza un caso de uso específico y cómo se ejecuta, en términos de clases de diseño y sus objetos. (4)

Por la complejidad que mostraba realizar un diagrama de secuencia mediante la interacción entre objetos, la dirección del proyecto decidió realizar estos diagramas mediante la interacción entre subsistemas a lo cual denominaron “Diagrama de contrato entre paquetes”, convirtiéndose de esta forma en un artefacto del proyecto. Es decir son diagramas de secuencia que contienen las instancias de actores, subsistemas, y trasmisiones de mensajes entre estos y que se diferencian en los siguientes aspectos. (4)

- La línea de vida en los diagramas de secuencia denota subsistemas en lugar de objetos del diseño.
- Cada subsistema identificado debería tener al menos una línea de vida que lo denote en un diagrama de secuencia.

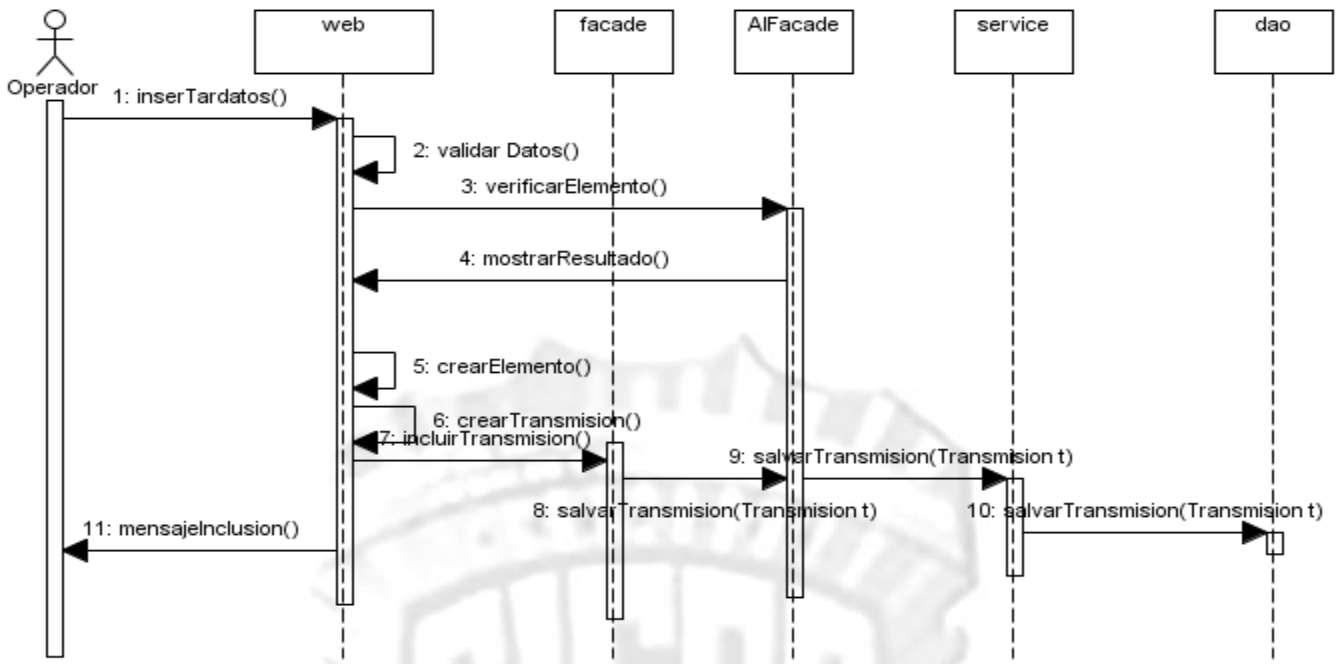


Ilustración 16.*Diseño. CU Gestionar Transmisión. Diagrama de Contratos entre Paquetes. Escenario Incluir.*

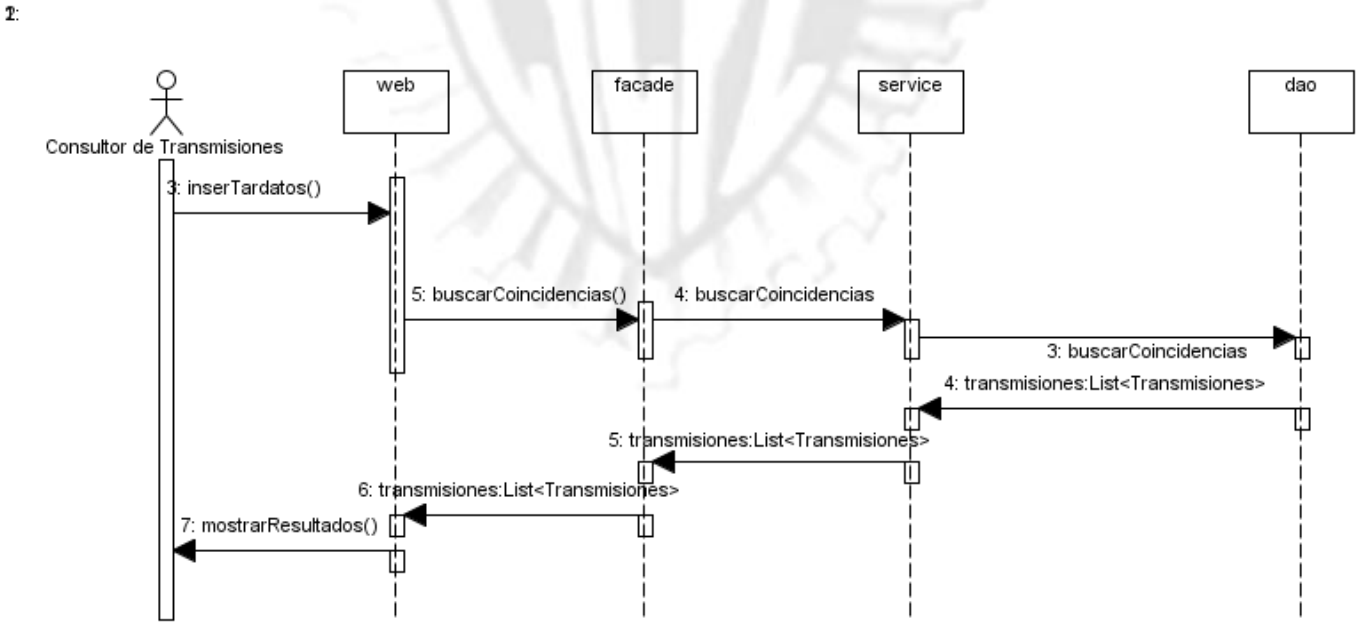


Ilustración 17.*Diseño. CU Consultar Transmisiones. Diagrama de Contratos entre Paquetes.*

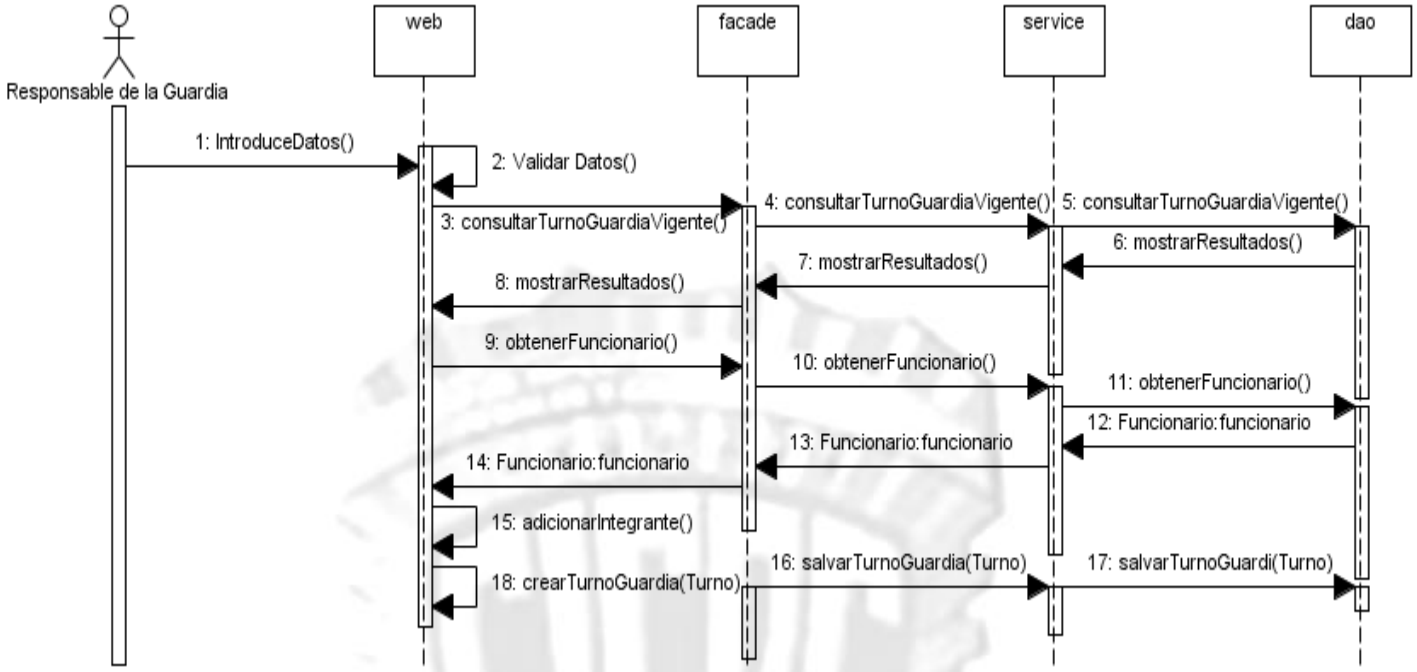


Ilustración 18.*Diseño. CU Gestionar Turno de Guardia. Diagrama de Contratos entre Paquetes. Escenario Incluir.*

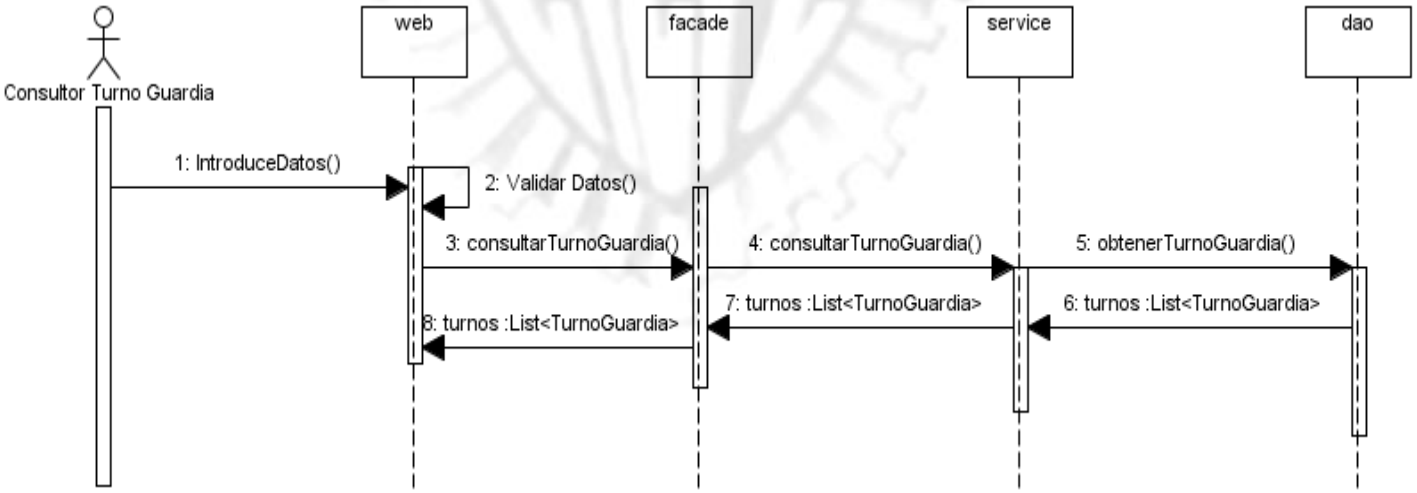


Ilustración 19.*Diseño. CU Consultar Turno de Guardia. Diagrama de Contratos entre Paquetes.*

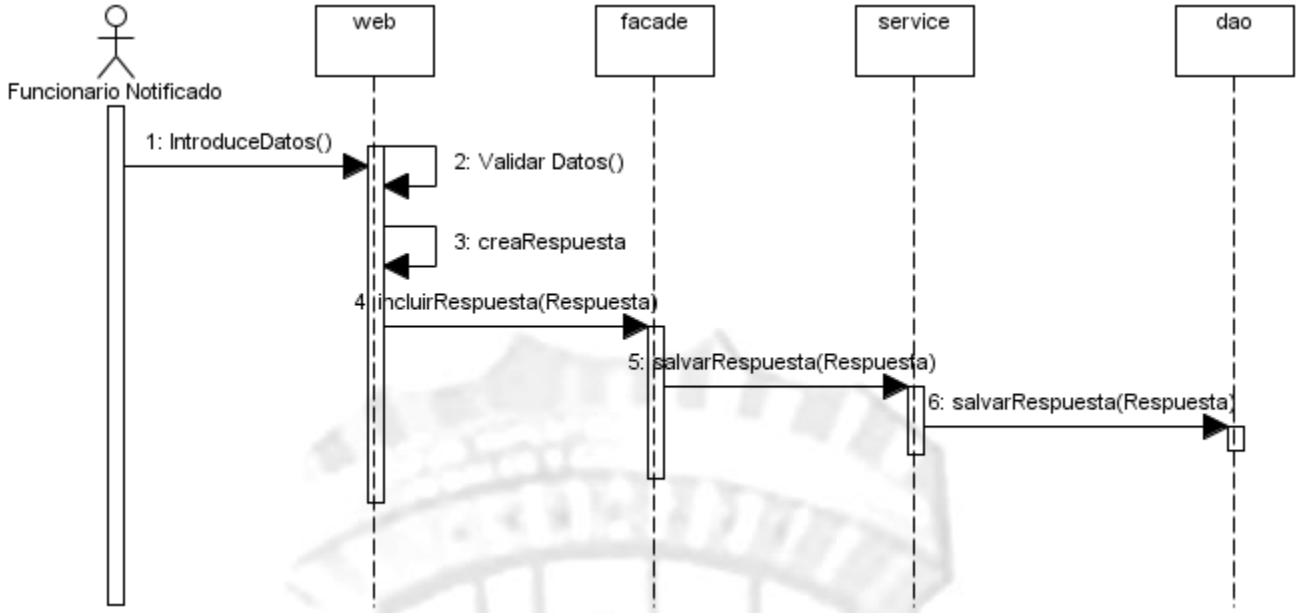


Ilustración 20.*Diseño. CU Incluir Respuesta Transmisiones. Diagrama de Contratos entre Paquetes.*

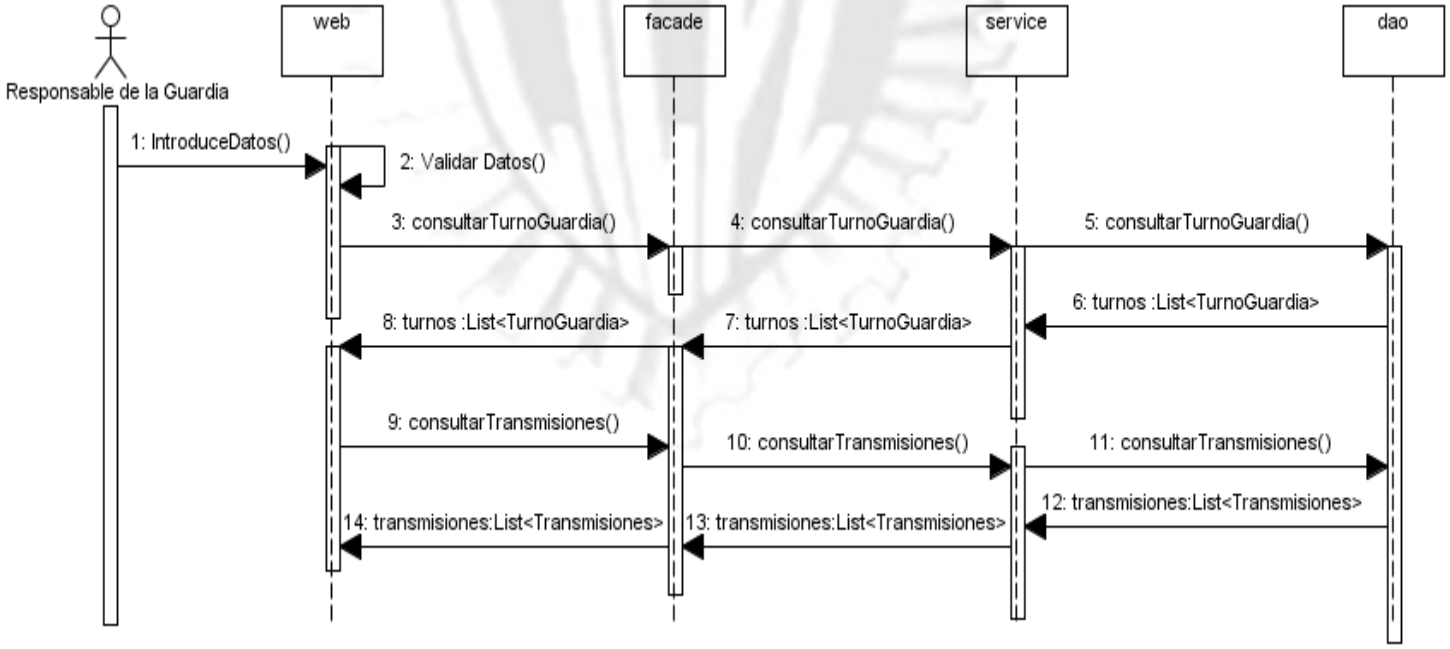


Ilustración 21.*Diseño. CU Incluir Respuesta Transmisiones. Diagrama de Contratos entre Paquetes.*

2.2. Modelo de Datos

En la informática, un modelo de datos es un lenguaje utilizado para la descripción de una base de datos. Por lo general, un modelo de datos permite describir las estructuras de datos de la base (el tipo de los datos que incluye, la base de datos y la forma en que se relacionan), las restricciones de integridad (las condiciones que los datos deben cumplir para reflejar correctamente la realidad deseada) y las operaciones de manipulación de los datos (agregado, borrado, modificación y recuperación de los datos de la base de datos).

El modelado de los datos de las Transmisiones se representará mediante dos diagramas fundamentales, el de clases persistentes, obtenidas del modelo de objetos del negocio y de aplicar el diseño a los diferentes casos de usos y el de las tablas del modelo relacional obtenidas del diagrama de clases persistentes.

2.2.1. Diagrama de Clases Persistentes

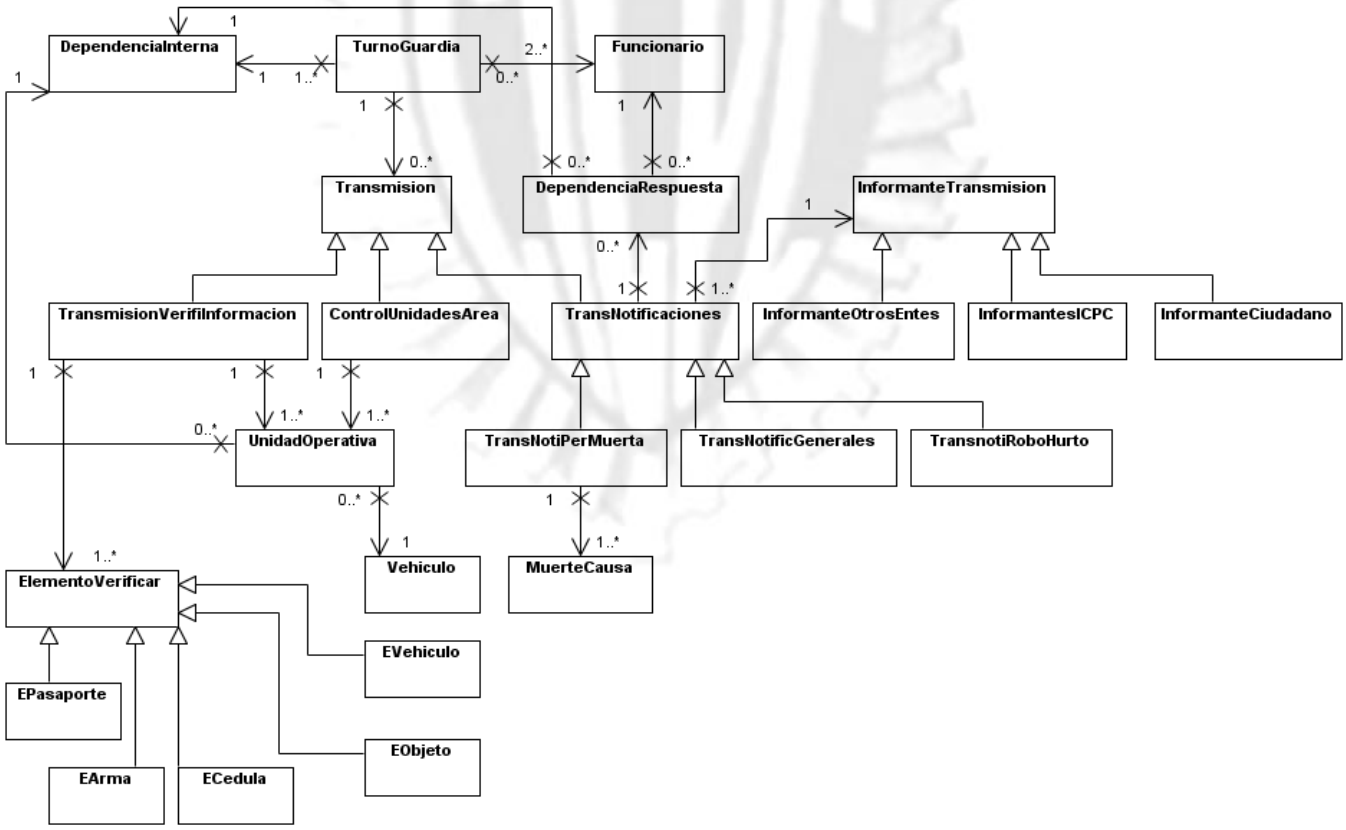


Ilustración 22.*Diagrama de clases persistentes*

2.2.2. Diagramas de Tablas del Modelo Relacional

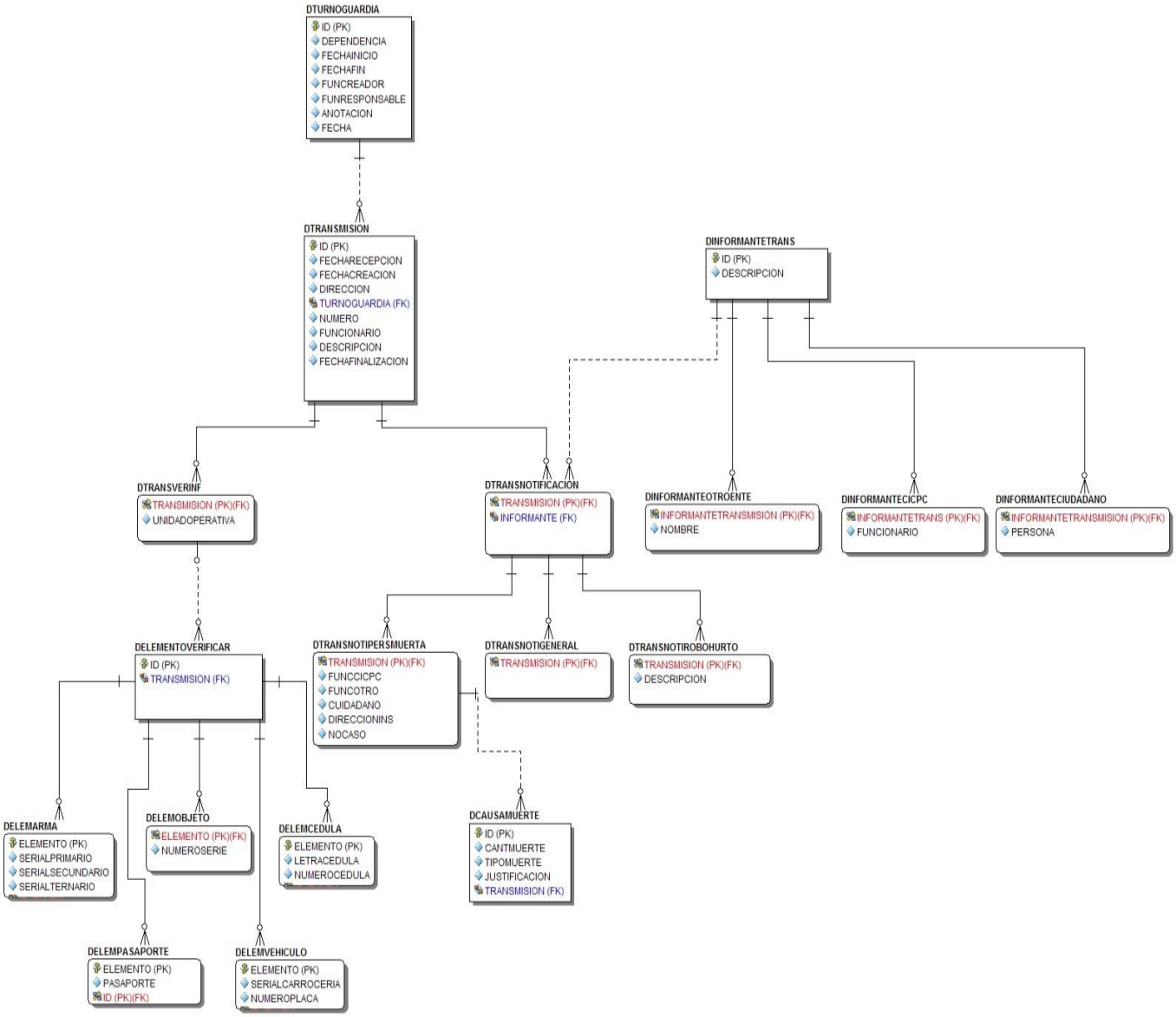


Ilustración 23.*Diagrama de tablas del modelo relacional*

2.3. Implementación

En la implementación empezamos con el resultado del diseño e implementamos el sistema en términos de componentes, es decir ficheros de código fuente, scripts, ficheros de códigos binario, ejecutables etc. Siendo uno de los resultados más importante del diseño la captura y definición de la arquitectura se convierte en factor indispensable para este flujo de trabajo el desarrollo de la misma y la implementación del sistema. Los propósitos de la implementación son: (4)

- Planificar las integraciones de sistema necesarias en cada iteración. Seguimos para ello un enfoque incremental, lo que da lugar a un sistema que se implementa en una sucesión de pasos pequeños y manejables. (4)

- Distribuir el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue. Esto se basa fundamentalmente en las clases activas encontradas durante el diseño. (4)

- Implementar las clases y subsistemas encontrados durante el diseño. En particular, las clases se implementan como componentes de fichero que contienen código fuente. (4)

- Probar los componentes individualmente, y a continuación integrarlos compilándolos y enlazándolos en uno o más ejecutables, antes de ser envidaos para ser integrados y llevar a cabo las comprobaciones de sistema. (4)

2.3.1. Modelo de Implementación

El modelo de implementación describe como los elementos del modelo de diseño, como las clases, se implementan en términos de componentes, como ficheros de código fuente, ejecutables, etc. El modelo de implementación describe también como se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y como dependen los componentes unos de otros. (4)

2.3.2. Diagramas de subsistemas de implementación

Los subsistemas de implementación proporcionan una forma de organizar los artefactos del modelo de implementación en trozos más manejables. Un subsistema puede estar formado por componentes, interfaces y otros subsistemas (recursivamente). Además estos pueden implementar y así proporcionar las interfaces que representan la funcionalidad que exportan en forma de operaciones. (4)

Después de explicada la función de los diagramas de subsistemas de implementación a continuación se muestra la organización de los artefactos generados en la implementación del módulo de Transmisiones.

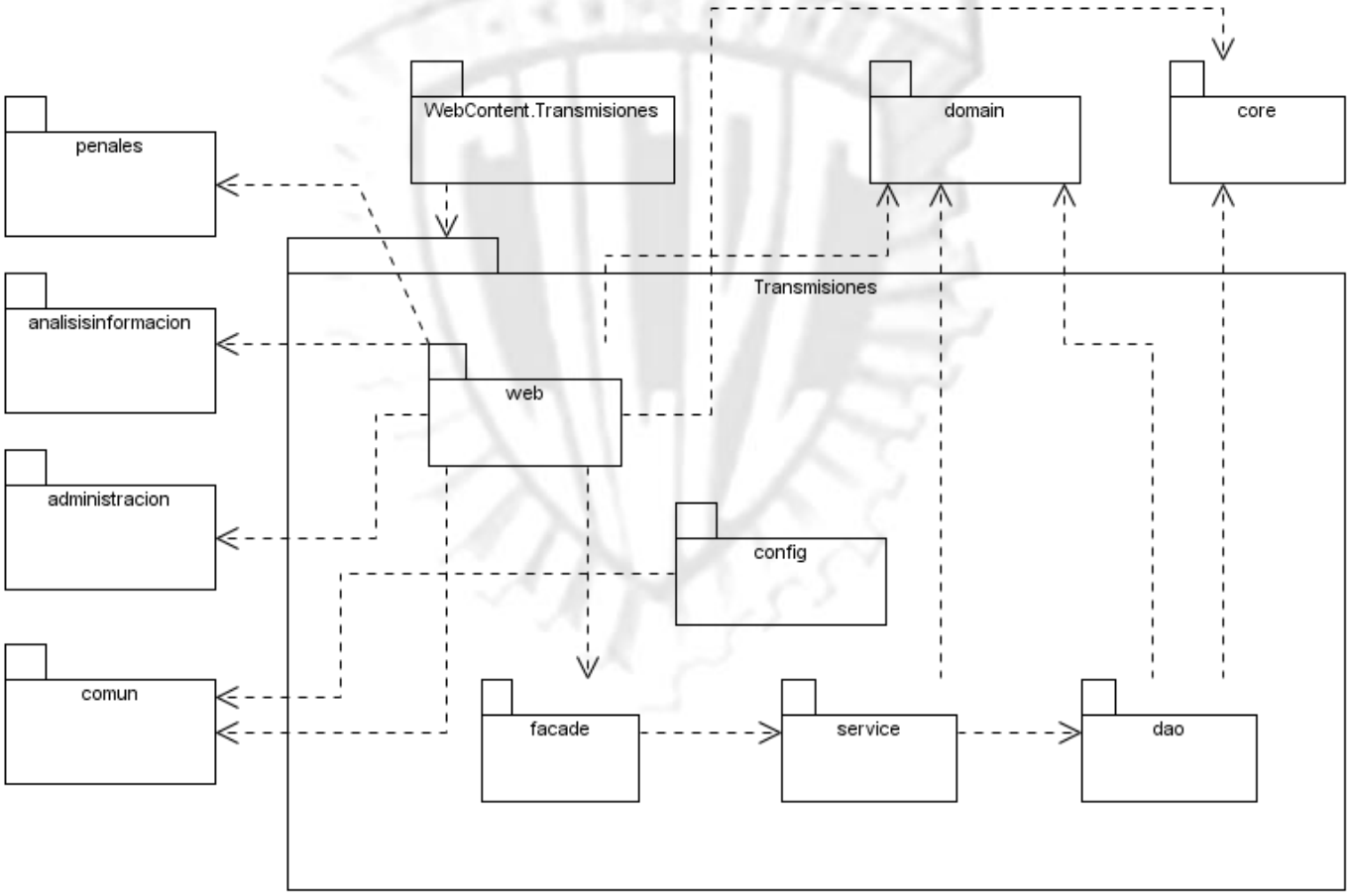


Ilustración 24.*Implementación. Diagramas de Subsistemas de Implementación.*

2.3.3. Diagrama de componentes

Un componente es el empaquetamiento físico de los elementos de un modelo, como son las clases en el modelo de diseño. (4) En un diagrama de componentes se representan las relaciones que existen entre los componentes que interactúan en el sistema. A continuación representamos los diagramas de componentes que contienen las relaciones de los subsistemas del módulo Transmisiones.

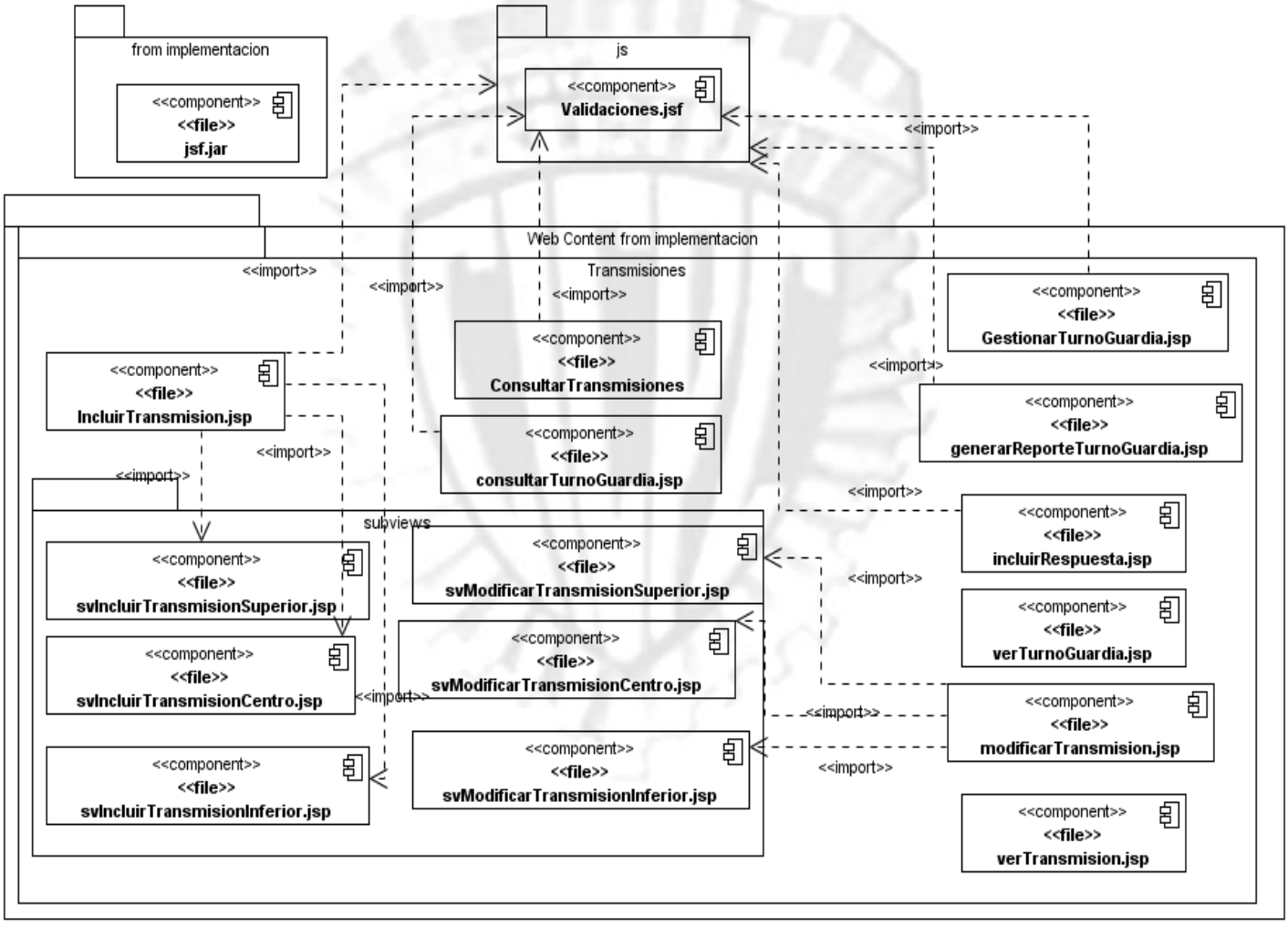


Ilustración 25. *Implementación. Diagrama de componentes. Vista de la capa de presentación. Páginas JSP*

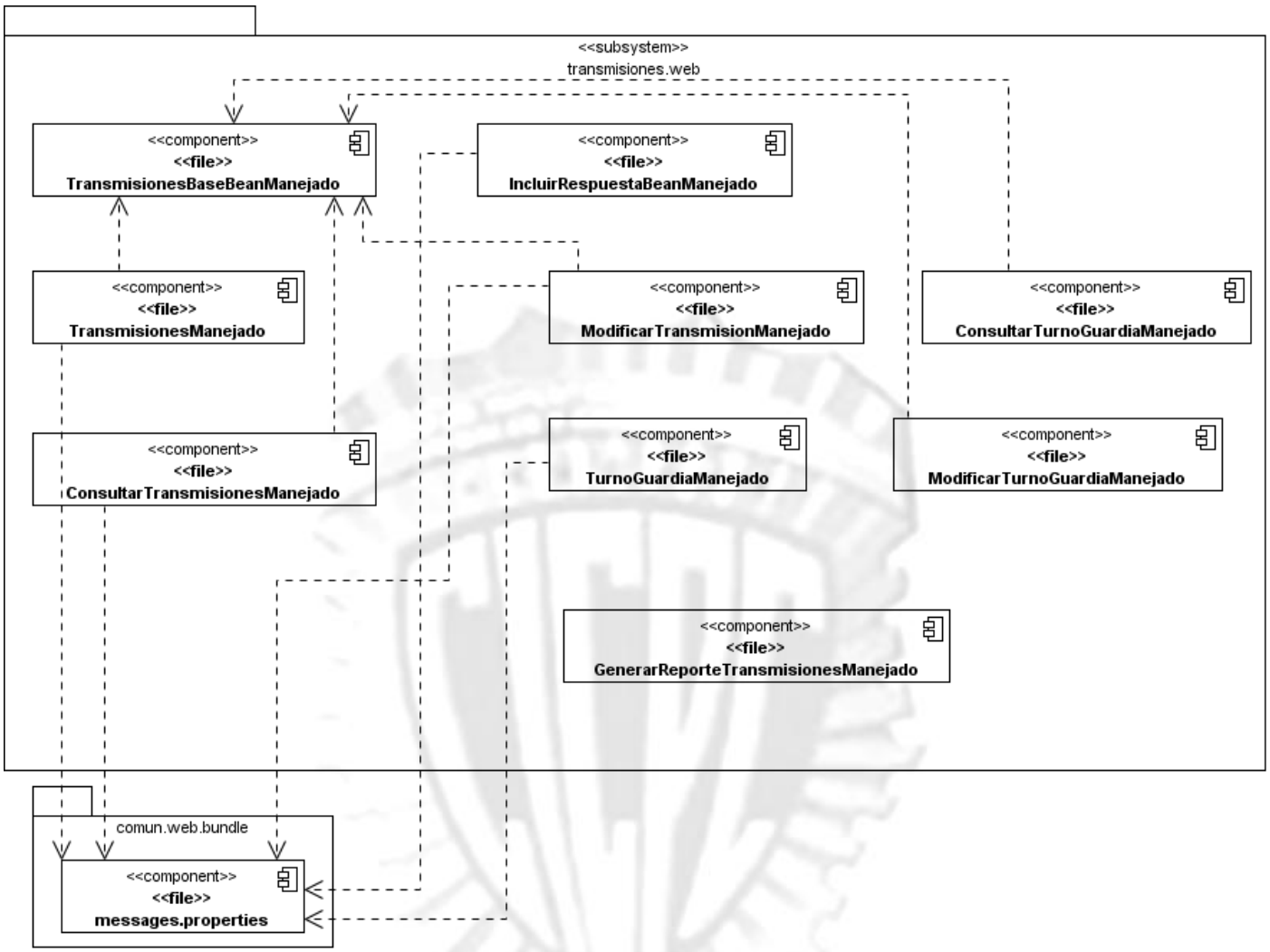


Ilustración 26.*Implementación. Diagrama de componentes. Vista de la capa de presentación. Beans Manejados*

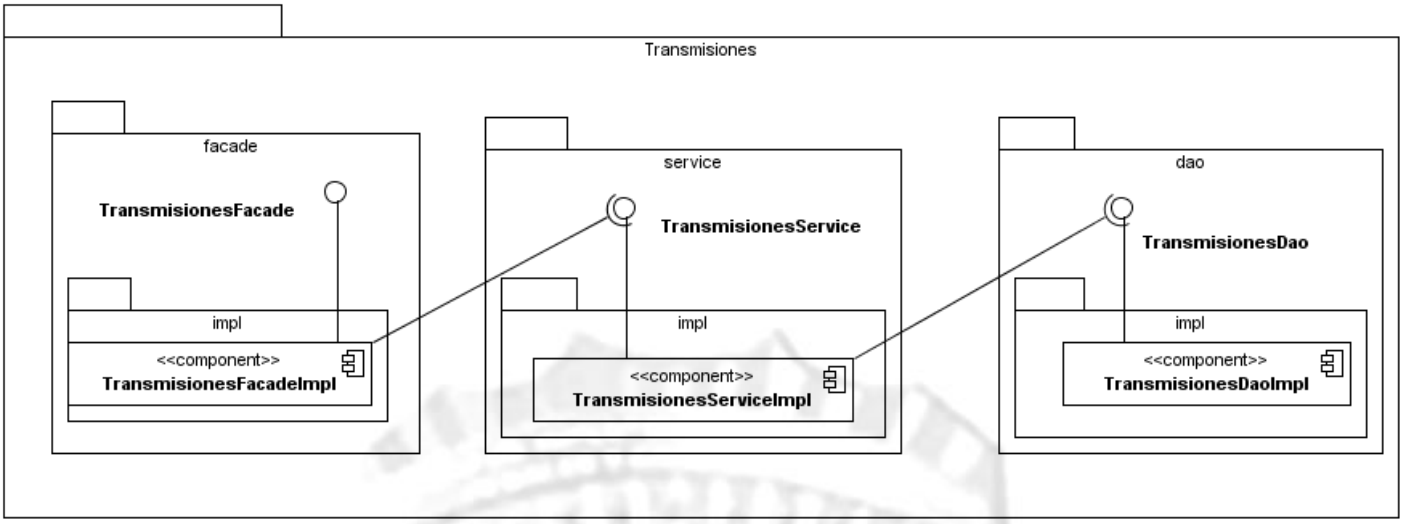


Ilustración 27.*Implementación. Diagrama de componentes. Vista de fachada, servicios y daos.*

Conclusiones

En este capítulo se abordó acerca de lo realizado en los flujos de trabajo análisis, diseño e implementación para el desarrollo de la solución propuesta, teniendo en cuenta las descripciones de los casos de uso como principal entrada al desarrollo del capítulo, obteniéndose así los principales artefactos generados en cada flujo, según lo propuesto por la metodología utilizada RUP estos son el modelo de diseño, modelo de datos y modelo de implementación, quedando la aplicación lista para entrar al flujo de pruebas. En el proceso solo se generó la documentación necesaria con el fin de agilizar el desarrollo de la solución. Otro aspecto significativo fue el uso de patrones de diseño, para evitar problemas frecuentes, estos ayudaron a realizar diseño estándar, facilitando el entendimiento y la reutilización de elementos de la solución propuesta.

El producto final de la investigación: el módulo Transmisiones fue construido dando cumplimiento gran parte de los objetivos específicos planteados. Para el mejor entendimiento de la solución propuesta se documentaron los procesos fundamentales y los artefactos generados.

CAPITULO 3. VALIDACIÓN DE LA SOLUCIÓN PROPUESTA.

Introducción

Este capítulo se enfoca en demostrar la validez de la solución propuesta para resolver el problema planteado. Con el objetivo de realizar esta actividad y siguiendo la metodología seleccionada para el desarrollo de la solución, se realizó el flujo de trabajo de pruebas, con el fin de validar cada artefacto generado en los flujos de trabajos anteriores.

En el flujo de trabajo de prueba se verifica el resultado de la implementación probando cada parte del sistema, ya sean internas o externas, para comprobar que el sistema implementado o producto cumple con todos los requisitos funcionales y no funcionales, así como con los estándares internacionales que deben cumplir los productos de su tipo. Como resultado este flujo se obtiene un producto con excelente calidad que cumple con las expectativas del cliente.

3.1. Métodos de Pruebas

Para llevar a cabo las pruebas a la solución propuesta se usaron varios métodos de pruebas entre los cuales se encuentran:

3.1.1. Prueba de Caja Blanca

Este método permite examinar la estructura interna del programa, comprobando los caminos lógicos del software y examinando el estado del programa en varios puntos para determinar si el estado real coincide con el esperado.(6) Estas pruebas fueron realizadas por el propio desarrollador de la solución, verificándose y corrigiéndose los errores encontrados, comprobándose así que la solución básicamente funcionaba como se deseaba, debido a que estas pruebas fueron realizadas por el propio desarrollador durante el proceso de implementación, de estas pruebas no existe ningún tipo de documentación, solo la constancia del correcto funcionamiento lógico de las funcionalidades.

3.1.2. Prueba de Caja Negra

Se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. Una prueba de caja negra examina algunos aspectos del modelo fundamental del sistema sin tener mucho en cuenta la estructura lógica interna del software. (6) Estas son las pruebas más comunes y conocidas en el caso de Transmisiones también son las más aplicadas ya que son realizadas por todo el equipo de desarrollo del software y además por el equipo de calidad UCI, son guiadas por los casos de uso y los conocidos casos de pruebas, se enfocan en verificar que la aplicación cumple con lo especificado en la descripción de los casos de uso y en que la interfaz de la aplicación sea agradable al usuario y cumpla con las pautas de diseño impuestas por el proyecto.

3.2. Niveles de pruebas

Todos los métodos de pruebas mencionados anteriormente son eficientes para lograr la validez de la solución en cuanto a lo especificado en los casos de uso pero no demuestran totalmente la ausencia de defectos, ni la correcta integración de la solución con el sistema y la arquitectura ya implementada debido a este motivo a la solución propuesta se le realizaron varias pruebas de nivel de las cuales se describen a continuación las más importantes:

3.2.1. Pruebas Unitarias

Esta prueba la hace el mismo programador y consiste en ir ejecutando el código para convencerse de que "básicamente, funciona". Esta prueba suele consistir en pequeños ejemplos que se intentan ejecutar. Si el módulo falla, se suele utilizar un depurador para observar la evolución dinámica del sistema, localizar el fallo y repararlo; a esto se refiere las pruebas de unidad. (6) La automatización de estas pruebas es fundamental para agilizar el proceso, por tal motivo en el caso del módulo Transmisiones estas pruebas se automatizaron mediante la implementación de una clase main que probara básicamente el código asegurando de que cada funcionalidad se ejecutara según lo planeado, obteniéndose como resultado algunos errores dados al probar con casos críticos (valores extremos) los cuales fueron corregidos en la medida del desarrollo.

3.2.2. Pruebas de Integración.

Las pruebas de integración se llevan a cabo durante la construcción del sistema, involucran a un número creciente de módulos y terminan probando el sistema como conjunto. Las pruebas funcionales de integración son similares a las pruebas de caja negra. Aquí se trata de encontrar fallos en la respuesta de un

módulo cuando su operación depende de los servicios prestados por otro(s) módulo(s). (6) Estas pruebas fueron realizadas por el equipo de desarrollo al que el módulo de Transmisiones pertenece, fueron orientadas a probar las dependencias de Transmisiones con otros módulos tales como Análisis de la Información y Penales y el módulo de Estadísticas.

A pesar de que no existe documentación oficial generada durante estas pruebas, se pueden mencionar los principales errores detectados y corregidos durante esta etapa.

- Cuando se asocia en el CU Gestionar Trasmisiones, un informante (CU Gestionar Persona módulo Análisis de Información), y se presiona el botón cancelar no regresa a la vista anterior.

  Solución al problema: No se estaba implementando bien la interfaz de comunicación IGestionarPersonaManejado donde se especifica en la implementación del método cancelar si se desea regresar o no.

- Cuando se valida un acta procesal en el CU Gestionar Trasmisiones el sistema lanza un error.

  Solución al problema: la fachada del módulo investigación penal encargado de realizar esta validación, se estaba inyectando en el constructor, debido a las recientes refactorizaciones de la arquitectura esto ya no funciona, se quitó ya que la inyección se realiza por reflexión mediante las anotaciones de spring.

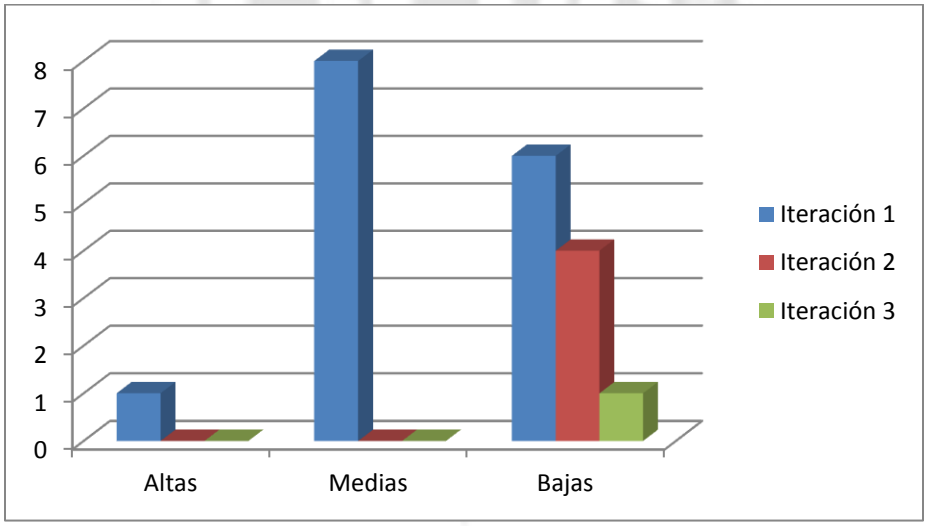
- En las Trasmisiones de Verificación de Información en el CU Gestionar Trasmisiones no realizaba la verificación de los elementos (dígase Arma, Objeto, Vehículo y Persona entidades del módulo Análisis de información)

  Solución al problema: Con la refactorización de los nomencladores y cambio de id en nomencladores que Trasmisiones usaba con valores fijos, las condiciones ya no se cumplían, evidenciando errores lógicos. Se corrigió cambiando los valores fijos a las constantes definidas en la nueva clase nomenclador.

3.2.3. Pruebas de Sistema

Estas pruebas verifican que cada elemento encaja de forma adecuada y que se alcanza la funcionalidad y el rendimiento del sistema total. Para este nivel de pruebas se utilizan principalmente los métodos de pruebas de caja negra, (11) que en el caso de la solución, fueron realizadas por el equipo de calidad interna del proyecto y el equipo de desarrollo del proyecto en tres iteraciones de las cuales, las primeras dos iteraciones, las realizó el equipo de desarrolladores del proyecto, estas pruebas se denominaron Pruebas Cruzadas, debido a que cada equipo de desarrollo prueba un módulo distinto al que le pertenece, para garantizar que sean encontrados nuevos errores, la otra iteración fue realizada por el equipo de calidad interna del proyecto. Estas pruebas son realizadas basándose en los casos de pruebas diseñados debidamente para la solución propuesta los cuales se muestran en los anexos.

Los resultados obtenidos al final de la aplicación de todas estas pruebas se presentan en la siguiente gráfica:



Al ver los la gráfica de los resultados obtenidos de las pruebas realizadas, se puede apreciar como las no conformidades va disminuyendo en cuanto a cantidad y nivel de complejidad a lo largo de las iteraciones lo cual deja claro el avance de la calidad de la solución. Al final de cada iteración de prueba, se realizó una etapa conocida como la etapa de Respuestas de no Conformidades, las no conformidades encontradas en el sistema se arreglaron dando paso a una nueva iteración de pruebas, en la cual se verificó si los errores detectados anteriormente habían sido corregidos, posteriormente se vuelve a probar el sistema en busca de

nuevos errores. Este proceso normalmente fue repetido hasta no encontrar no conformidades significativas en la solución.

3.2.4. Pruebas de Aceptación

Finalmente se realizaron las pruebas de aceptación, las cuales son llevadas a cabo por el cliente con el objetivo probar las funcionalidades del sistema. Estas pruebas buscan una cobertura de la especificación de requisitos y del manual del usuario. Dichas pruebas no se realizan durante el desarrollo, pues sería impresentable de cara al cliente; sino que se realizan una vez pasada todas las pruebas de integración por partes de los desarrolladores. En estas como resultado se tuvo 0 no conformidades y varios pedidos de cambio.

Aunque estas pruebas no tributan directamente a la validación de la investigación, pues se va del alcance de esta y se centra en probar que los requisitos fueron bien especificados, si constituyen un elemento de solidez de los resultados obtenidos, quedando satisfecho el cliente final con el módulo.

Conclusiones

En este capítulo se describió todo lo referente a las pruebas realizadas a la propuesta de solución con vista a validar la solución y a garantizar que esta cumpliera con los requisitos especificados. Se puede concluir que la estrategia usada por el proyecto y definida por la jefatura del mismo fue bastante efectiva quedando demostrada con la ganancia de calidad de la solución. Luego de analizar los resultados satisfactorios obtenidos y habiendo corregido cada no conformidad detectada, se puede decir que el producto final obtenido es un producto con elevada calidad y que da cumplimiento a todos los requisitos funcionales y no funcionales asociados al módulo de Transmisiones lo que demuestra la validez de la solución.

Conclusiones

El presente documento contiene la documentación necesaria del proceso de desarrollo del módulo Transmisiones perteneciente al software SIIPOL, módulo encargado de registrar y controlar las Transmisiones generadas en un turno de guardia. Este trabajo incluye desde el estudio de las tendencias tecnológicas actuales, factor fundamental para la propuesta de la solución hasta la validación de la propuesta de solución siguiendo todos los flujos de trabajos de la metodología de desarrollo de software establecida RUP.

Como elementos concluyentes de la presente investigación se aprecia:

Las herramientas y definiciones arquitectónicas puestas en práctica permitieron el desarrollo claro y fluido de un sistema construido sobre bases sólidas y un entorno bien definido.

El uso de RUP como metodología de desarrollo, guió todo el proceso de desarrollo, aseguró la calidad del producto según lo esperado. También facilitó la creación de los artefactos necesarios para el proceso.

La validación del software fue de gran importancia, pues se comprobó el adecuado funcionamiento de las principales funcionalidades del módulo, logrando una elevada satisfacción con el usuario final.

Como legado de esta investigación se obtuvo el módulo Transmisiones integrado al software SIIPOL que da cumplimiento a los requisitos especificados, al problema a resolver planteado al inicio de esta investigación y al objetivo general de la investigación: Desarrollar un subsistema que dé solución a los requisitos funcionales y no funcionales asociados al módulo de Transmisiones del SIIPOL.

Recomendaciones

- Realizar un proceso de refactorización en todos los casos de uso ajustándose también a las demás refactorizaciones ejecutadas por los demás módulos del SIIPOL.
- Ampliar las funcionalidades del módulo según las necesidades del cliente.
- Tener en cuenta este trabajo para proyectos futuros, siempre y cuando se cumpla con las normas de confidencialidad establecidas.
- Integrar el módulo Transmisiones con el módulo Guardia desarrollado en la 5ta etapa del proyecto.

Bibliografía

1. **CICPC.** Portal CICPC. [En línea] 2011. http://www.cicpc.gov.ve/.

2. **Alonso, Arlan Galvez y Pérez, Amed Vázquez.** *Análisis, Diseño e Implementación del submódulo Evidencia del módulo Registro y Control del SIIPOL.* s.l. : UCI, 2009.

3. *Mapas Digitales.*

4. **Ivar Jacobson, Grandy Booch,James Rumbaugh.** *El Proceso Unificado del Desarrollo del Software.* 1999.

5. **Rational Software Corporation.** *Ayuda del RUP. .* 2003. Suite del Rational 2003.

6. **Sánchez, Eduardo Alfonso.** *Análisis, Diseño e Implementación de los Sub Módulos Denuncia y Control de Investigación pertenecientes al Módulo de Investigación Penal del Sistema de Investigación e Información Policial (SIIPOL).* UCI : s.n., 2009.

7. **Network, Oracle Technology.** Oracle Technology Network. [En línea] 2011. http://www.oracle.com/technetwork/java/javaee/overview/index.html. http://www.oracle.com/technetwork/java/javaee/overview/index.html.

8. **Soria, Jorge Amado.** *DESCRIPCION DE LA ARQUITECTURA DE SOFTWARE.* 2011.

9. **Grails, and Roo.** *Spring Persistence with Hibernate.*

10. **Frómeta, Ing. Maykell.** *Documento Requerimiento Suplementarios.* 2008.

11. **MAYLIN DIAZ CABRERA, KENNY LÓPEZ ESPINOSA.** *ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE LA AGENDA TRABAJO DEL SISTEMA DE INVESTIGACIÓN E INFORMACIÓN POLICIAL.* UCI : s.n., 2009.

12. **Larman, Craig.** *UML y Patrones.*

13. **Guevara, Humberto Rivero.** *Análisis, diseño e implementación del módulo Aprehensión del SIIPOL.* s.l. : UCI, 2008.

14. **CHRISTIAN BAUER, GAVIN KING.** *Hibernate in Action.* 2006.

15. **MURPHY, PAUL TEPPER FISHER & BRIAN D.** *Spring Persistence with Hibernate.* 2010.

16. **James Rumbaugh, Ivar Jacobson, Grady Booch.** *El lenguaje Unificado de Modelado. Manual de Referencia.* 2007.

17. **David Geary, Cay Horstmann.** *Core JavaServer™ Faces, Second Edition.* 2007. ISBN 978-0-13-173886-7.

18. **Fallows, Jonas Jacobi and John R.** *Pro JSF and Ajax Building Rich Internet Components.* 2006. ISBN-10: 1-59059-580-7.

19. **KING, CHRISTIAN BAUER AND GAVIN.** *Java Persistence with Hibernate.* 2006.

20. **Wiley, John.** *Patterns In Java - Volume 2.* 2005.

21. **FORMAN, IRA R. y FORMAN, NATE.** *Java Reflection in Action.* 2009.

22. **Ediciones Especiales.** Ediciones Especiales. [En línea] http://www.edicionesespeciales.elmercurio.com/destacadas/detalle/index.asp?idnoticia=0110082006021X3020026&idcuerpo=414/.

23. **Hispano, Java.** Java en Castellano. [En línea] 2011. http://www.programacion.com/java.

24. **ACN.** La Evolución del Periodismo. [En línea] http://www.acn.com.ve/actualidad/sucesos/item/13203-venezuela-es-el-pa%C3%ADs-m%C3%A1s-violento-del-mundo.html.

25. **Software, Departamento de Ingeniería de.** *Conferencias de ISW. UC.* 2008-2009.

26. **Rumbaugh, James, Jacobson, Ivar y Booch, Grady.** *El Lenguaje Unificado de Modelado. Manual de Referencia.*

27. **Rodríguez, Yadiel Ramos.** *Sistema de Apoyo a la Investigación Criminalística para el Cuerpo de Investigación Científica Penales y Criminalística de Venezuela.* Habana : s.n., 2007.

28. **reserved, Laura Chinchilla – Seguridad 2010. All rights.** Laura firme y honesta. *Sistema integrado de estadísticas policiales / Vigilancia electrónica.* [En línea] 2010. http://www.lauracr.com/seguridad/2010/sistema-integrado-de-estadisticas-policiales-vigilancia-electronica/.

29. **Presman, Roger S.** *Ingeniaría de Software. Un enfoque práctico.*

30. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* Madrid : Pearson Education.S.A, 2000. 84-7829-036-2.

31. **Sitio Oficial JAVA. [En línea] 2009.** Sitio Oficial JAVA. [En línea] 2009. http://java.com/es.

32. *Visual Paradigm for UML.* [En línea] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p/.

33. **www.springsource.org .** Spring Source Community. [En línea] 2010. [Citado el: 13 de Enero de 2010.] http://www.springsource.org/.

34. **ORACLE CORPORATION.** ORACLE. [En línea] 2009. http://www.oracle.com/index.html.

35. *La flecha,tu diario de ciencia y tecnologia.* [En línea] http://www.laflecha.net/canales/softlibre/noticias/red-hat-presenta-developer-studio-basado-en-eclipse.

36. JUnit.org Recursos para la Prueba de Desarrollo Impulsado. [En línea] 2009. http://www.junit.org/.

37. **Oracle Corporation.** Java EE at a Glance. [En línea] 2010. http://java.sun.com/javaee/index.jsp.

38. **Ambysoft Inc.** *El Proceso Unificado Ágil v1.1.* [En línea] 13 de mayo de 2005-2006. http://cgi.una.ac.cr/AUP/index.html.

39. Documento de Especificación Requerimiento No funcionales del Proyecto Mejoramiento de procesos Analsisis y Diseño del Sistema de Información. [En línea] 7 de Junio de 2006. http://www.minproteccionsocial.gov.co/vbecontent/library/documents/DocNewsNo16758DocumentNo5401.PDF.

40. Agenda Magna. *Sistema de denuncias policiales.* [En línea] 7 de diciembre de 2009. http://agendamagna.wordpress.com/2009/12/07/sistema-de-denuncias-policiales/.

41. *Red Hat Developer Studio.* [En línea] sitio Web de Red Hat Developer Studio, 2009. http://www.redhat.com..

42. **Larman, Craig.** *UML y Patrones.*

43. HIBERNATE. *Relational Persistence for Java & .NET.* [En línea] http://www.hibernate.org/.

44. **Copyright © 1997-2010 Object Management Group, Inc. All Rights Reserved.** UNIFIED MODELING LANGUAGE. *UML® Resource Page.* [En línea] 2007-2010. http://www.uml.org/.

45. **Rodríguez., Msc. Yadiel Ramos.** *ARTEFACTOS A GENERAR.* 6/05/2009.

46. **Muñoz, David Ruiz.** *Manual de Estadística.*

47. **Tamayo, Lic. Deymis.** *REQUERIMIENTOS SUPLEMENTARIOS.* Habana : s.n., 2007.

48. **Boehm, Barry and Turner, Richard.** *Balancing Agility and Discipline: a guide for the perplexed.* Addison Wesley : s.n., 2005. 0321186125..

GLOSARIO

AJAX: Java Script asíncrono y XML es la unión de varias tecnologías que juntas pueden lograr aplicaciones realmente impresionantes como GoogleMaps, Gmail, el Outlook Web Access o algunas otras aplicaciones muy conocidas: **AJAX, en resumen, es el acrónimo para Asynchronous JavaScript + XML** y el concepto es: Cargar y renderizar una página, luego mantenerse en esa página mientras scripts y rutinas van al servidor buscando, en background, los datos que son usados para actualizar la página solo re-renderizando la página y mostrando u ocultando porciones de la misma. (13)

Ajax4JSF: Ajax4jsf es una librería open source que se integra totalmente en la arquitectura de JSF y extiende la funcionalidad de sus etiquetas dotándolas con tecnología Ajax. (13)

API: Application Programming Interface.

AOP: Programación Orientada a Aspectos.

Bean: En el lenguaje Java es un componente software reutilizable que evita programar los distintos componentes uno a uno. Existen con la finalidad de ahorrar tiempo al programar. (13)

DAO: Data Access Object, traducido al español: Objeto de Acceso a Datos, es un Patrón de diseño de clases en ingeniería de software que permite a quien lo aplique suministrar una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos. (13)

Framework: Un conjunto de clases abstractas y concretas que colaboran entre sí y que se pueden utilizar como plantilla para solucionar una familia de problemas relacionados. Normalmente se extiende por medio de la definición de subclases para el comportamiento específico de una aplicación. (11)

Hibernate: Hibernate ofrece la *Persistencia Relacional para Java*, que para los no iniciados, proporciona unas muy buenas maneras para la persistencia de sus objetos de Java a y desde una base de datos subyacente. Más que ensuciar con SQL tus objetos y convertir consultas a y desde los objetos de primera

magnitud, Hibernate puede preocuparse de todo ese maremágnum por ti. Tú utilizas solamente a los objetos, Hibernate se preocupa del SQL y de que las cosas terminan en la tabla correcta. (13)

IDE: En español Entorno Integrado de Desarrollo, es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse exclusivamente para un lenguaje de programación o bien para varios. (11)

IoC: Inversión de Control. Es una técnica alternativa a las búsquedas clásicas de recursos vías JNI. **JEE:** Es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en lenguaje de programación java con arquitectura de N niveles distribuida.

JDBC: Java Database Connectivity.

JSF: Java Server Faces, framework de interfaz de usuario para aplicaciones web desarrolladas en java.

POJO: Acrónimo de Plain Old Java Object es una sigla empleada por desarrolladores de Java para enfatizar el uso de clases simples y que no dependen de un framework en especial.

Spring: Es un framework open source, creado por Rod Johnson y descrito en su libro *Expert One-on-One: JEE Design and Development*.

UML: Lenguaje de Modelado Universal.