Universidad de las Ciencias Informáticas

Facultad 3

Título: *Diseño e implementación de los procesos de inicio y diligencias para el módulo Verificación Fiscal del Sistema de Informatización de la Gestión de la Fiscalía fase II.*

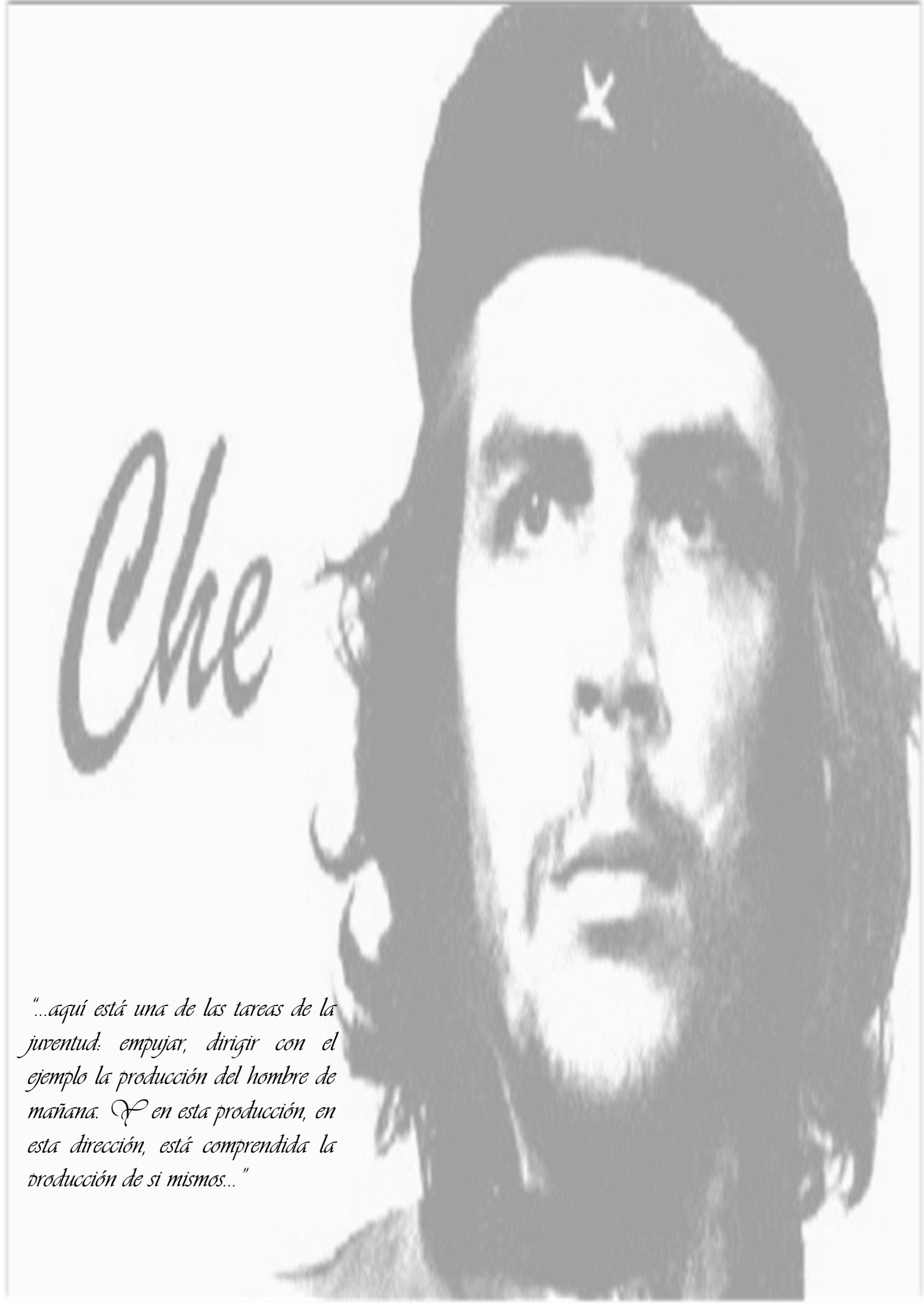
Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor(es): Jessica Carriles Cardo

Yasel Pereira Torres

Tutor: Ing. Giorgy G. Cabrera Lamadrid

La Habana, Cuba. Julio de 2014



"...aquí está una de las tareas de la juventud: empujar, dirigir con el ejemplo la producción del hombre de mañana. Y en esta producción, en esta dirección, está comprendida la producción de si mismos..."

DECLARACIÓN DE AUTORÍA

Declaramos que somos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Jessica Carriles Cardo

_____

Firma del Autor

Yasel Pereira Torres

_____

Firma del Autor

Ing. Giorgy Gilberto Cabrera Lamadrid

_____

Firma del Tutor

DATOS DE CONTACTO

Datos del Tutor:

Nombre: Giorgy Gilberto Cabrera Lamadrid

Ingeniero en Ciencias Informáticas.

Correo electrónico: ggcabrera@uci.cu

Datos de los Autores:

Nombre: Jessica Carriles Cardo

Correo electrónico: jcarriles@estudiantes.uci.cu

Nombre: Yasel Pereira Torres

Correo electrónico: ypereira@estudiantes.uci.cu

AGRADECIMIENTOS

De Jessica

Es muy importante para mi dejar constancia de mi agradecimiento a toda aquellas personas que han hecho posible que este trabajo llegue a verse concluido, pues muchas de ellas han influenciado con su sensibilidad, tiempo, habilidades, amistad, sugerencia, paciencia y amor a su creación.
A mi tutor por sus conocimientos y oportunas orientaciones.
A mi compañero de tesis por ser un amigo más, por estar ahí durante este período de tesis compartiendo conmigo ratos amargos y agradables
A mi mamá por ser madre y mujer ejemplar que me supo criar y educar correctamente, que me apoyo durante toda mi vida y me supo enseñar que en la vida todo es sacrificio y esfuerzo y que el que persevera triunfa. La quiero muchísimo.
A mi papá Julian por ser una persona digna de admiración y respeto, por quererme y criarme como su hija, por apoyarme igualmente cuando lo he necesitado. Por eso y tantas cosas más gracias eternamente.
A mi hermano Orly por ser ejemplo de inteligencia y sabiduría. Por compartir conmigo todos mis años de vida, por ayudarme y defenderme siempre, por preocuparse y estar pendiente. Por ser un gran hermano.
A mi hermana Jocelyn que llegó a mi vida en un momento muy importante de la adolescencia, donde engendró en mí un tipo de amor diferente como se siente cuando tienes un hijo. Por darme amor y hacerme sentir que soy importante en su vida y por ser tan especial.
A mis abuelos Pupi y Ñico porque siempre me apoyaron y me enseñaron que en la vida hay que luchar por lo que uno quiere. Por siempre estar pendiente de mí y apoyarme. Los quiero y los querré siempre.
A mi novio por estar junto a mí estos 5 años y apoyarme siempre en todos los momentos. Por quererme y respetarme.
A mis familiares porque siempre creyeron en mí y por darme el apoyo que uno necesita, por ser parte de mi vida.
A mis suegros que se han portado conmigo como mis otros padres, porque me han acogido y apoyado muy bien, por hacerme sentir parte de la familia.
A mis compañeros que con amor contribuyeron a mi formación en especial a:
A mis amigas Sarisleidy y Wanda por estar ahí conmigo desde el primer día que pisé esta escuela. Por apoyarme siempre y por estar ahí en los momentos buenos y malos de mi vida. Supieron comprenderme y a convivir conmigo a pesar de mi carácter y de mí forma de ser. Por compartir junto a mi estos últimos 5 años de estudiante y hacer de ellos los mejores durante todos los años de estudio.
A mi amigo Leonardo Cabrera por ser la persona que me sirvió de ejemplo en esta escuela para salir adelante, fue mi guía, me hizo saber que yo si podía, solo tenía que creérmelo yo misma primero. A él le agradezco grandemente los resultados que obtuve durante mi carrera porque fue mi fuente de inspiración para ello.
A mi amigo Pedro por ser una persona muy gentil y buena, por siempre estar dispuesto cuando uno necesita de él, por ser especial y servir de ejemplo igualmente.

Dedicada a ellos. Trato de buscar una palabra de gratitud y amor perfecto, no se aún si existe, pero al menos coincide con la que hombres y mujeres de todos los tiempos y de toda condición han creído encontrar y de esa manera han escrito innumerables testimonios de sus sentimientos.

De Yasel

En especial a mi mamá por su apoyo, su esfuerzo y sacrificio incondicional en todo momento, sin ella nada de esto hubiera sido posible, es por eso que quisiera regalarle este momento porque sé que lo disfrutará tanto como yo.

A mi tía Olga Lydia que ha sido mi madre estos 5 años de universidad.

A mi tutor por estar siempre disponible para nosotros y ayudarnos en todo lo que le pedimos.

A mi compañera de tesis que compartió conmigo estas horas de sacrificio, y además de apoyarme como compañera de tesis demostró ser una gran persona y amiga.

A mi padrastro, mi hermano Julio Ernesto, a mi primo Elton

A mis abuelos, que aunque algunos ya no estén conmigo sé que estarían orgullosos de mí, este momento también es para ellos.

A mi novia, a todos mis amigos, los que ya tenía antes de entrar a la universidad y a los que conocí aquí y pude compartir con ellos cada momento de estos 5 años.

Al profesor Yenier Figueroa por brindarme su apoyo cuando lo necesitaba, a la profesora Maribel, la profesora Yulia, la profesora Ailenys gracias a todas por sus consejos y recomendaciones.

A Lázaro, Hector, Kiki, Analiet, Claudia, Felipe, todos los compañeros del proyecto SIGEF II.

A todas las personas que de una forma u otra hicieron posible este sueño.

DEDICATORIA

De Jessica

A mi madre porque siempre ha esperado de mi lo mejor.
A mi abuelo porque aunque no pudo ver a su hija y a su nieta graduarse sabe que fue la fuente de inspiración para ellas.
A mi hermana porque quiero ser un ejemplo a seguir para ella y demostrarle que en la vida solo hay que proponérselo para poder alcanzarlo.

De Yasel

En especial a mi madre por su apoyo, su esfuerzo y sacrificio incondicional en todo momento.
A toda mi familia y amigos por estar siempre ahí cuando les necesitaba.
A todas las personas que de una forma u otro hicieron posible este sueño....
GRACIAS

RESUMEN

La Fiscalía General de la República (FGR), es el órgano a través del cual se garantiza el control, legalidad y protección de la sociedad cubana. Integrada por varias secciones que juntas componen todo el escenario de la fiscalía general. Los procesos que se llevan a cabo en el área de Verificación Fiscal (VF) se realizan de forma manual lo que provoca tiempos elevados en la tramitación de documentos. El objetivo de la presente investigación es el desarrollo de una solución informática para contribuir a la gestión de la información y ayudar en la celeridad y el control de los procesos de inicio y diligencias del módulo VF.

El trabajo consta de tres partes fundamentales para llegar a la completa comprensión del proceso de desarrollo de software. Una primera parte necesitó una investigación sobre el tema de las VF(es) y la elección de las herramientas necesarias para poner en marcha la elaboración del sistema. En un segundo momento se obtiene la base de la implementación, representada por los artefactos generados en los flujos de trabajo. Se finaliza con la implementación y pruebas de software para el aseguramiento del correcto funcionamiento del sistema.

Así, una vez alcanzado el resultado esperado se obtuvo la plena satisfacción de los usuarios y un sistema capaz de cumplir las necesidades de las fiscalías: mejoras en el manejo y protección de la información a través de la celeridad y control de sus procesos.

PALABRAS CLAVES

Fiscalía General de la República, Verificación Fiscal, legalidad, tramitación, implementación, control, celeridad.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ANEXOS

INTRODUCCIÓN

En la actualidad el desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) propician un cambio significativo en todas las esferas de la economía y la sociedad. Estas traen consigo transformaciones que condicionan y determinan la vida ciudadana. Un ejemplo de esto lo constituye el desarrollo de gobiernos electrónicos en países tales como Argentina, Colombia, México, Chile y Brasil los cuales utilizan aplicaciones de software para gestionar sus procesos internos y manejar su información. Permitiendo crear una sociedad completamente informatizada y mejorando la comunicación entre sus miembros. (1)

En Cuba el uso y el desarrollo de las TIC han dado un gran impulso a la creación de soluciones informáticas que multipliquen los servicios que se brindan a la población. Uno de los centros insignias en el desarrollo de software lo constituye la Universidad de las Ciencias Informáticas (UCI), la cual tiene un papel protagónico en la informatización de la sociedad cubana. La UCI tiene su infraestructura productiva dividida en centros de desarrollo, cada uno con áreas temáticas específicas. El Centro de Gobierno Electrónico (CEGEL) tiene como misión dentro de la UCI, satisfacer necesidades de clientes gubernamentales mediante el desarrollo de productos, servicios y soluciones integrales de alta confiabilidad, calidad, competitividad, fidelidad y eficiencia, a partir de un personal altamente calificado. CEGEL tiene entre sus principales clientes a la Fiscalía General de la República (FGR), para el cual desarrolla un sistema encargado del control de la información y facilitar la gestión de ésta. Dicho proyecto se nombra Sistema de Informatización de la Gestión de las Fiscalías fase II (SIGEF II). El cual contribuye a mejorar la calidad y la rapidez con la que se ejecutan los procesos que se realizan en las diferentes áreas de la FGR. Además de ser una fuente de estadísticas que facilite la toma de decisiones a los diferentes niveles del estado cubano.

La FGR tiene entre sus direcciones la de Verificación Fiscal (VF), la cual tiene como objetivo dar cumplimiento al mandato constitucional, velar por la observancia de la legalidad y combatir las manifestaciones de corrupción en los Organismos del Estado y sus dependencias, las direcciones subordinadas a los órganos locales del Poder Popular, las demás entidades económicas y sociales y por los ciudadanos, formulando en los casos necesarios los pronunciamientos que resulten. (2)

Al ejecutar una verificación o investigación fiscal en una entidad, los procesos de inicio y diligencia que en ella se realizan, generan un cúmulo de información y documentación la cual debe ser consultada en todo momento. Esto trae consigo:

- Una demanda de esfuerzo y tiempo considerable por parte del fiscal, pues tiene que consultar una gran cantidad de documentos para encontrar la información que necesita.
- Que el almacenamiento de la información se torne complejo pues al estar en formato manuscrito o impreso, ocupa un espacio considerable y se deteriora con el paso del tiempo, imposibilitando su conservación por un período prolongado, el cual en la actualidad no puede exceder de los 5 años pues conlleva a su deterioro o pérdida.

Esto unido al déficit de fiscales que tiene el país trae consigo que atiendan más de un caso a la vez, provocando que no estén pendientes de los plazos de vencimientos establecidos, lo que conlleva a que se tenga menor control y supervisión por parte del nivel superior.

Por último el envío de documentos entre las instancias inferiores y superiores se realiza mediante el correo tradicional o los mensajeros de las fiscalías, trayendo como consecuencia que se afecte la celeridad en la gestión de la información y se consuma más tiempo.

De la problemática enunciada se deriva el siguiente **problema a resolver** ¿Cómo mejorar la gestión de la información referente a los procesos de inicio y diligencias en las verificaciones e investigaciones fiscales, de manera que mejore el control y la celeridad en la ejecución de los mismos?
Partiendo del problema anteriormente planteado se ha determinado como **objeto de estudio:** El desarrollo de software de gestión en la informática jurídica, dirigido a cumplir el **objetivo general:** Diseñar e implementar los procesos de inicio y diligencias de manera que permita una mayor celeridad y control en la ejecución de las verificaciones e investigaciones fiscales, enmarcado en el **campo de acción:** Proceso de desarrollo de software de gestión de los procesos fiscales.
Para darle cumplimiento a lo anterior se proponen los siguientes **objetivos específicos:**

- Elaborar el marco teórico de la investigación mediante un análisis del estado del arte alrededor de los procesos de verificaciones e investigaciones fiscales para la realización del diseño teórico.
- Realizar el diseño de los procesos de inicio y diligencias del módulo VF del proyecto SIGEF II.
- Implementar los procesos de inicio y diligencias del módulo VF del proyecto SIGEF II.
- Validar la propuesta de solución mediante pruebas de software.

Tomando en consideración el problema planteado se propone la siguiente **idea a defender:** El diseño y la implementación de los procesos de inicio y diligencias permitirán mejorar la gestión de la información en las verificaciones e investigaciones fiscales, de manera que se logre un mayor control y celeridad en la ejecución de las mismas.

Para dar cumplimiento a los objetivos planteados se definieron las siguientes **tareas de la investigación:**

- Análisis del proceso de desarrollo de software utilizado en el proyecto SIGEF II, específicamente las disciplinas de diseño, implementación y pruebas.
- Análisis de los procesos que se realizan en el inicio y diligencias en la verificación e investigación fiscal.
- Modelado del diagrama de entidades, de vistas, de controladores y gestores, de secuencias y de repositorio de los procesos de inicio y diligencias del módulo VF del proyecto SIGEF II.
- Validación del diseño asociado a los procesos de inicio y diligencias del módulo VF del proyecto SIGEF II.
- Implementación de las funcionalidades de los procesos de inicio y diligencias del módulo VF del proyecto SIGEF II.
- Validación de los resultados obtenidos a través de pruebas al sistema, a fin de comprobar su correcto funcionamiento.
- Validación de la propuesta de solución de la investigación frente al problema planteado con el cliente.

Pretendiendo alcanzar como **posibles resultados** un sistema funcional para agilizar la gestión de los procesos de inicio y diligencias del módulo VF en el proyecto SIGEF II.

Métodos Teóricos

- Histórico-Lógico: Se empleó para el estudio de los antecedentes del proceso de VF que se lleva a cabo en las FGR. Se utilizó además para el análisis de las tendencias actuales empleadas en el proyecto SIGEF II para el desarrollo del software.
- Analítico-Sintético: Se utilizó para el estudio de la bibliografía y permitió el procesamiento de la información, arribando a conclusiones prácticas.
- Modelación: Para el diseño del sistema informático se utilizó UML como lenguaje, por su notación gráfica muy expresiva que permite representar todas las fases de un proyecto informático.

El presente documento está dividido en tres capítulos:

Capítulo 1. Fundamentación Teórica

Se realiza un estudio del estado del arte donde se hace un amplio recorrido en el sector internacional para la obtención de información sobre los procesos de verificaciones fiscales en busca de características similares al resultado que se desea obtener. Además se describe la metodología seleccionada, herramientas, tecnologías y lenguajes que serán utilizados en la presente investigación.

Capítulo 2. Descripción de la solución propuesta

Se obtienen los artefactos generados a través de los flujos de trabajo como diagramas de clases, de repositorios, de secuencia, de vistas, gestores y controladores, haciendo uso de patrones de diseño. Se valida la diagramación mediante el empleo de métricas de diseño para brindar un esquema sencillo de implementación. Finaliza el capítulo con el modelo de implementación, mediante los diagramas de despliegue y de componentes.

Capítulo 3. Validación de la solución propuesta

Se realiza la validación del resultado obtenido para simplificar los posibles errores existentes mediante las pruebas de caja blanca y caja negra para asegurar su correcto funcionamiento. Se comprueba que el resultado final cumple con las necesidades de la FGR y mejora la gestión de la información en cuanto a celeridad y control en la ejecución de sus procesos.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se abordan conceptos fundamentales para lograr un mejor entendimiento del proceso de ejecución de las verificaciones e investigaciones fiscales. Se realiza un estudio del estado del arte sobre las principales soluciones informáticas existentes en el mundo, relacionadas con las VF. Además se fundamenta la selección de la metodología de software, así como las tecnologías y herramientas utilizadas tomando como punto de partida la política de migración hacia software libre que sigue el país y las restricciones por las cuales se regirá el sistema que se quiere desarrollar.

1.1. Conceptos Fundamentales

Fiscalía General de la República de Cuba: es el órgano encargado de proteger el orden político y jurídico del Estado y la sociedad, procurando el restablecimiento de la legalidad quebrantada promoviendo acciones y medidas contra los infractores. (2)

Acción: Se denomina acción al proceso en el cual se realiza una investigación o verificación fiscal. La realización de una acción está marcada por los procesos de inicio y las diligencias que en ella se practican. Siendo estas últimas la base de dichas acciones. (3)

Diligencias: Las diligencias son documentos que se realizan a lo largo de todo el proceso de investigación o verificación fiscal. Son la base de las acciones y la esencia de las verificaciones fiscales. (3)

Inicio: Fase en la que se encuentran los procesos que dan comienzo a una investigación o verificación fiscal. (3)

1.2. Escenario actual del proceso

La FGR tiene una dirección de VF que tiene a su cargo la dirección metodológica y el control de la realización de las verificaciones fiscales e investigaciones, por los órganos de la Fiscalía, en los Organismos del Estado, sus dependencias y las direcciones subordinadas a los órganos locales del Poder Popular y las demás entidades económicas y sociales.

Las investigaciones o verificaciones fiscales reciben la clasificación de acción. Inicialmente se elabora el plan de trabajo por parte de los fiscales actuantes, en el que a partir de los objetivos de la acción y otros objetivos que pueden ser agregados, se trazan una serie de tareas a realizar para darle

cumplimientos a estos. La ejecución de una acción en una entidad se puede realizar de manera sorpresiva o con previo aviso. Para realizar una acción sorpresiva esta debe ser solicitada de manera formal al departamento o a la dirección de VF según corresponda, luego esta solicitud puede ser aprobada o denegada por dicha instancia. La ejecución de una acción comienza con la reunión de inicio que se realiza entre los fiscales actuantes y los principales cuadros de la entidad verificada o investigada. Durante todo el proceso de ejecución de una verificación o investigación fiscal se practican una serie de diligencias, las cuales constituyen uno de los pasos fundamentales durante todo este proceso. Dichas diligencias son:

Despacho: En esta diligencia se realiza una solicitud al fiscal actuante que se encuentra en la entidad con la necesidad de que practique varias diligencias. Se pueden realizar varios despachos durante todo el proceso. (4)

Acta de entrevista: En esta diligencia se toman las declaraciones de las personas entrevistadas así como sus datos personales. (4)

Acta de ocupación: Durante la ejecución de una acción se pueden realizar varias actas de ocupación, en cada una de ellas se plasma lo ocupado, así como los datos personales de la persona a quien se le ocupa. (4)

Acta de entrega: En ella se relaciona todo lo entregado así como los datos personales de la persona a quien se le hace la entrega de los bienes ocupados. (4)

Informe de especialista: Esta diligencia puede ser presentada por un especialista de la fiscalía o adjunto a esta. En esta se plasma toda la información recogida durante la acción de control. (4)
En la Figura 1 se muestra lo planteado anteriormente.

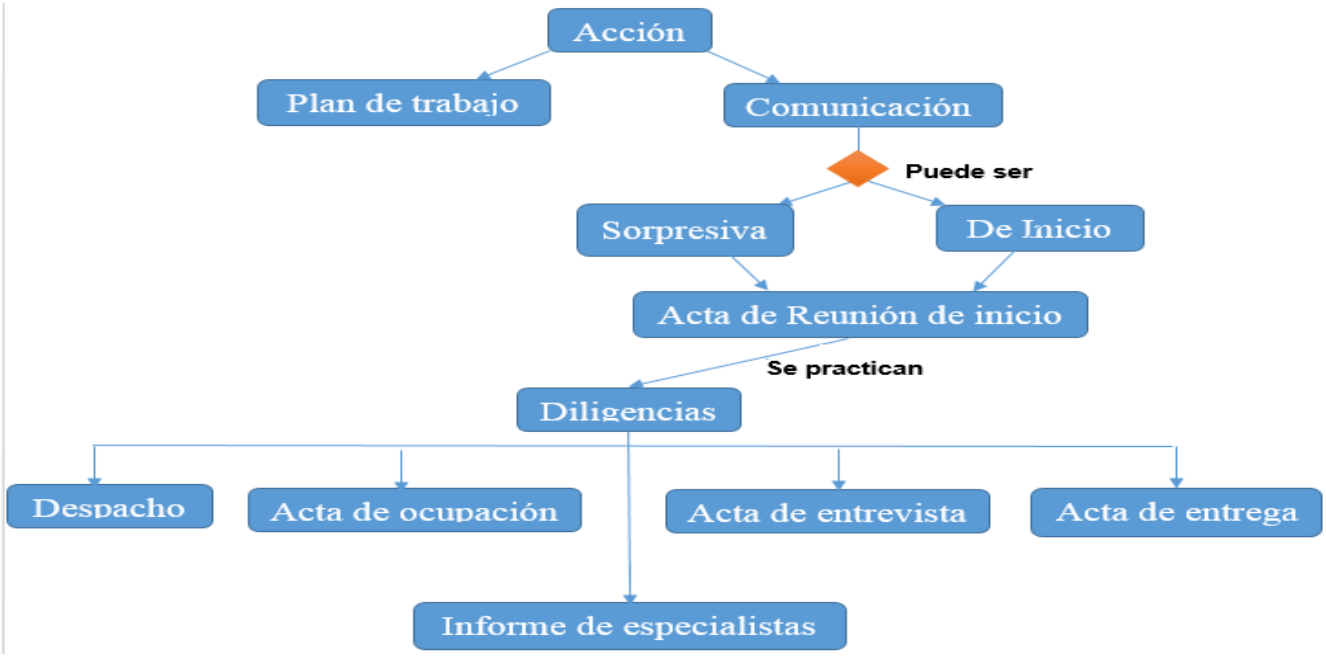


Figura 1: Flujo de los procesos de inicio y diligencias

1.3. Sistemas para gestionar los procesos de verificaciones fiscales.

En la actualidad existen varios países que se encuentran inmersos en la creación de un gobierno electrónico los cuales a través del uso eficiente de las tecnologías de la información, han logrado crear sistemas de gestión fiscal con el objetivo de lograr un mayor control del almacenamiento y procesamiento de la información, así como obtener una mayor transparencia en la política fiscal. Algunos de los sistemas informáticos de gestión fiscal desarrollados por dichos países se dedican específicamente a gestionar las investigaciones y verificaciones fiscales que en ellos se realizan, entre los más significativos están:

SINOR: El Sistema de Información y Producción Normativa es construido y empleado por la Contraloría General de la República de Colombia. Este sistema permite brindarle servicios y datos de interés a la población, entre ellos se encuentran los procesos penales, quejas y denuncias, servicios de atención al ciudadano y las verificaciones fiscales. Las verificaciones están orientadas a los recursos, bienes e intereses patrimoniales del Estado. (5)

SICC: El Sistema de Inspección, Control y Consulta es una herramienta utilizada en la Fiscalía General del Estado de España, permite obtener estadísticas y consultar datos en todas las fiscalías dependientes del Ministerio de Justicia, o radicadas en Comunidades Autónomas que cuentan con un convenio de

utilización de los instrumentos informáticos suministrados por el Ministerio. Además es un sistema que aporta herramientas de auditorías y control. (6)

KIWI: Sistema Informático Central del Ministerio Público Fiscal CABA implementado y puesto en práctica por la Fiscalía General de Buenos Aires. Este software permite gestionar digitalmente los casos y las solicitudes administrativas que se diligencian internamente en la institución. Abarca desde la toma de denuncias por distintas vías, hasta el análisis jurídico, diligenciamiento, investigación y judicialización de los casos jurisdiccionales. (7)

Fortuny: Sistema de Información del Ministerio Fiscal desarrollado por el Ministerio de Justicia y la Fiscalía General del Estado de España, encargado de la gestión procesal que incorpora información relativa a la incoación de causas, trámites procesales, informes, escritos de acusación, señalamientos y sentencias. Su utilización facilita las comunicaciones entre las fiscalías, los órganos jurisdiccionales y los distintos operadores jurídicos y agiliza la tramitación de procesos. Permite a todas las fiscalías trabajar con el mismo registro, información y tramitación, garantizando de esta forma una mayor eficacia y agilidad en la justicia de la región, pues la información puede ser accedida de forma segura y fiable desde todos los territorios en tiempo real. (8)

Los sistemas antes mencionados son medidos a continuación teniendo en cuenta si son privativos, nivel de escalabilidad que presentan en alto y bajo, si gestionan o no información y si generan alertas a los diferentes niveles de las fiscalías.

Sistemas	Software privativos	Escalable	Gestión de información	Generación de alertas
SINOR	Si	Alto	Si	No
SICC	Si	Bajo	Si	Si
KIWI	Si	Bajo	Si	No
Fortuny	Si	Bajo	Si	No

Tabla 1: Comparación de los sistemas existentes

Estos sistemas fueron desarrollados para lograr eficiencia y calidad en el marco al que fueron aplicados, pero no constituyen una solución factible para Cuba, pues no son lo suficientemente flexibles para adaptarlos a otras leyes, además son costosos y trabajan sobre software privativos, lo cual entra en contradicción con las políticas de migración de software libre que se está llevando a cabo por parte del país. A pesar de que permiten la gestión de información no se realizan procesos específicos que necesita el sistema judicial cubano cuando ejecuta una verificación o investigación fiscal. Por lo que se hace necesario crear un sistema que permita gestionar los procesos de inicio y diligencia que se ejecutan en

las VF. El cual responda a las necesidades de la dirección de VF de la FGR y que se integre como un módulo al SIGEF II.

1.4. Proceso de desarrollo de software.

El proceso de desarrollo de software es aquel donde las necesidades del usuario son traducidas en requerimientos de software, estos requerimientos transformados en diseño y el diseño implementado en código, el código es aprobado, documentado y certificado para su uso operativo. Específicamente define quién está haciendo qué, cómo y cuándo alcanzar un determinado objetivo. (9)

Para garantizar la gestión del desarrollo de un sistema están definidas varias metodologías, pues de no ser guiado el proceso lo que se obtiene finalmente son clientes insatisfechos y productos inoperables. Pero no solo se trata de regirse por una metodología sino de seleccionar la más adecuada de acuerdo a las características específicas de cada proyecto.

1.5. Metodología de Desarrollo de Software

Las metodologías de desarrollo de software se definen como *"el conjunto de procedimientos, técnicas, herramientas y soporte documental que ayuda a los desarrolladores a realizar un nuevo software"*. (10)

Proceso Unificado de Desarrollo (RUP)

RUP es la metodología de desarrollo de software establecida en la propuesta de arquitectura de software de CEGEL para proyectos de gestión sobre plataformas web, con grandes equipos de desarrolladores. Su selección está dada por ser un proceso iterativo e incremental, lo que facilita que el proceso de desarrollo sea planificado y gestionado. Además es configurable, porque a pesar de ser una metodología extensa y que genera una gran cantidad de artefactos, ofrece la posibilidad de definir solamente los entregables que los desarrolladores consideren necesarios. Permite adaptarse a los cambios de los requisitos con pocas alteraciones, así como detectar y gestionar los riesgos durante todo el ciclo de vida del proyecto. Otra característica que contribuyó a su selección es que se trata de una de las metodologías más utilizadas en la actualidad para el desarrollo de software. Utiliza el Lenguaje Unificado de Modelado para modelar todos los artefactos necesarios durante el desarrollo del sistema.

El ciclo de vida de RUP es una implementación del desarrollo en espiral, a través de un modelo iterativo e incremental, este organiza las tareas en fases y dentro de ellas se realizan iteraciones según la etapa en que se encuentre el proyecto. En la siguiente figura se muestran las fases por las cuales transita el proyecto para su desarrollo.

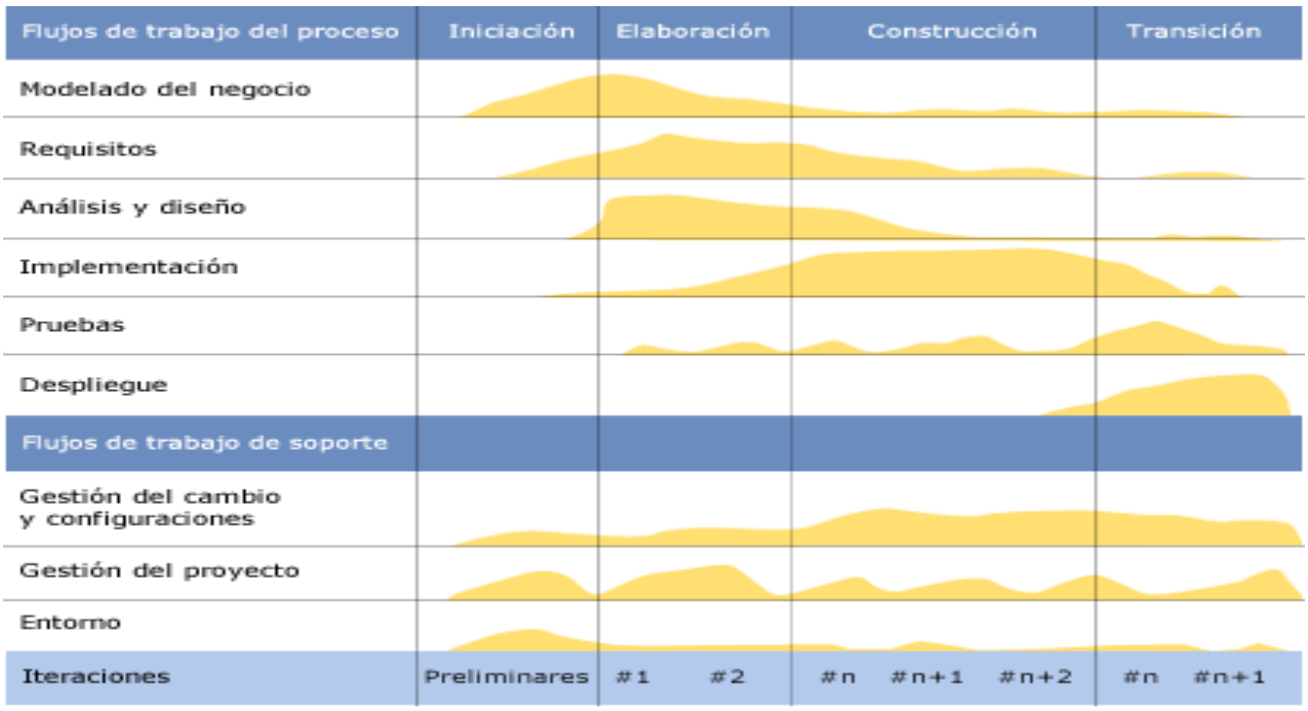


Figura 2: Esfuerzo en actividades según fase del proyecto *(11)*

Las fases por las que pasa el proyecto son: (11)

- **Inicio:** Su objetivo es determinar la visión del proyecto, en esta fase las iteraciones hacen mayor énfasis en actividades de modelado del negocio y de requisitos.
- **Elaboración:** En esta etapa se determina la arquitectura óptima del proyecto, las iteraciones están orientadas a los flujos de trabajo de requisitos, modelado de negocio, análisis, diseño y a una parte de la implementación.
- **Construcción:** El principal objetivo en esta etapa es llegar a completar la funcionalidad del sistema, en esta fase es necesario seleccionar algunos casos de uso, refinar su análisis y diseño y proceder con la implementación y pruebas.
- **Transición:** El propósito de esta fase es asegurar que el software esté disponible para los usuarios finales, ajustar los errores y defectos encontrados en las pruebas de aceptación. Se debe verificar que el producto cumpla con las especificaciones entregadas por las personas involucradas en el proyecto.

Este trabajo se centra en las fases Elaboración y Construcción, donde se generan cada uno de los artefactos definidos por la metodología en estas fases, para ello es necesario hacer uso de herramientas y tecnologías.

1.6. Tecnologías y herramientas

Las herramientas y tecnologías se emplean para informatizar actividades de manera completa o parcial, para incrementar la productividad, la calidad y para reducir el tiempo de desarrollo. A continuación se detallan las características significativas de las herramientas seleccionadas para el desarrollo del sistema que resuelva los problemas planteados en el presente trabajo. Su selección es llevada a cabo a través de un estudio realizado por los principales desarrolladores del proyecto SIGEF II, además de sus convincentes características para apoyar el desarrollo de software.

1.6.1. Gestor de Base de Datos PostgreSQL 9.1

PostgreSQL es un sistema gestor de base de datos de código abierto, disponible libremente para la comunidad de usuarios. Este gestor utiliza el modelo cliente-servidor y emplea multiprocesos para garantizar su estabilidad. Entre las características más generales se destaca la creación de copias de seguridad y que presenta múltiples métodos de autenticación. (12)

PostgreSQL también presenta una alta concurrencia mediante un sistema denominado MVCC[1], éste permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Como nuevas características que presenta la versión 9.1 se incluye la replicación sincrónica, que permite una alta disponibilidad con consistencia sobre múltiples servidores. (13)

Se selecciona PostgreSQL por ser un sistema de gestión de base de datos relacional orientado a objetos, el cual puede trabajar con herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional, (14) además es multiplataforma y compatible con Linux, plataforma sobre la cual se desarrollará el sistema.

PgAdmin III 1.14

PgAdmin es una aplicación gráfica que permite administrar el gestor de base de datos PostgreSQL. Es de código abierto y está diseñado para responder a todas las necesidades del usuario, desde escribir consultas SQL[2] simples hasta desarrollar bases de datos complejas. La interfaz gráfica soporta todas las características de PostgreSQL. (15) Se utiliza PgAdmin debido a que será necesario el trabajo con múltiples tablas de la base de datos. PgAdmin facilita la administración de la base de datos y sus campos.

[1]MVCC: Acceso concurrente multiversión.
[2]SQL: por sus siglas en inglés Structured Query Language, en español Lenguaje de Consulta Estructurado. Es un lenguaje de base de datos normalizado, utilizado por los diferentes motores de bases de datos para realizar determinadas operaciones sobre los datos o sobre la estructura de los mismos.

1.6.2. Herramienta para el modelado Visual Paradigm for UML 8.0

Visual Paradigm es una herramienta CASE[3] para el desarrollo de aplicaciones utilizando el lenguaje de modelado unificado, provee el modelado de procesos de negocios y genera además un mapeo de objetos-relacionales para los lenguajes de programación Java, .NET y PHP. Esta herramienta ofrece navegación intuitiva entre la escritura del código y su visualización, además puede generar informes en formato PDF/HTML. Como características principales, ayuda a los equipos de desarrollo de software a sobresalir de toda la acumulación de trabajo y así desplegar el proceso de desarrollo de software en el tiempo establecido, lo que permite maximizar y acelerar tanto las contribuciones individuales como las de equipo, además proporciona código y compatibilidad hasta con más de 10 lenguajes. (16)

Esta herramienta se selecciona para la diagramación visual de la base de datos y la ingeniería, debido a las ventajas que ofrece en la diagramación de los artefactos obtenidos por los flujos de trabajo a través del lenguaje de modelado unificado. Además proporciona integridad con el IDE de desarrollo y con el sistema gestor de base de datos utilizado.

1.6.3. Lenguaje de Modelado Unificado (UML)

UML es un lenguaje para especificar, construir, visualizar y documentar los elementos que forman un producto de software. Es utilizado para el modelado de procesos de negocios. Permite organizar el proceso de diseño de tal forma que los analistas, clientes, desarrolladores y otras personas involucradas en el desarrollo del sistema lo comprendan y convengan con él. Su principal objetivo es la unificación de los métodos de modelado de objetos, además se encarga de la elección de una representación gráfica cuya sintaxis sea simple, expresiva e intuitiva, también define varios modelos para la representación de los sistemas que pueden verse y manipularse mediante un conjunto de diagramas diferentes. (17)

Se decide utilizar UML como notación para el desarrollo del software porque permite modelar sistemas haciendo uso de técnicas orientadas a objetos, además de que permite especificar todas las decisiones de análisis y diseño, construyéndose así modelos precisos, no ambiguos y completos. (18)

1.6.4. Entorno de Desarrollo Integrado (IDE) NetBeans 7.3

NetBeans IDE 7.3 es una herramienta de código abierto, pensada para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Es un producto libre y gratuito sin restricciones de uso. Incluye entre sus características más notables el soporte para múltiples framework PHP, mejora la

[3]CASE: por sus siglas en inglés Computer Asisted Software Engineering, en español Ingeniería de Software Asistida por Ordenador.

velocidad de escaneos de proyectos, así como el rendimiento en sistemas remotos y tiene completamiento de código para todos los lenguajes soportados. (19)

El software se desarrollará sobre NetBeans 7.3 puesto que contiene las herramientas para llevar a cabo la implementación de la totalidad de los componentes establecidos por la arquitectura y el diseño. Además es multiplataforma, permite una perfecta integración con el marco de trabajo Symfony2 y con los sistemas de control de versiones, como es el caso del Subversión.

1.6.5. Subversion 1.5

Un sistema de control de versiones es un sistema que tiene la capacidad de recordar todos los cambios que se hacen tanto en la estructura de directorios como en el contenido de los ficheros. (20)

Entre las razones que fundamentan su uso están que puede acceder a repositorios a través de la red, lo que le permite ser usado por personas que se encuentran en distintos ordenadores, además genera las trazas sobre los archivos que han sido modificados, agregados y eliminados, para un mejor control de lo que va sucediendo con el software. Proporciona agilidad a los desarrolladores, pues permite que varias personas estén trabajando sobre un mismo archivo.

1.6.6. Marco de trabajo

"Un marco de trabajo simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Proporciona además una estructura al código fuente, forzando al desarrollador a crear código más legible y más fácil de mantener". (21)

1.6.6.1. Symfony2 2.0

Symfony2 es un marco de trabajo diseñado para optimizar el desarrollo de aplicaciones web. Su funcionamiento interno está basado en la arquitectura Modelo Vista Controlador (MVC), por lo que separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web, es rápido y flexible. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. (18)

Symfony2 está desarrollado completamente con PHP5, además es compatible con el gestor de base de datos PostgreSQL y puede ser ejecutado tanto en plataformas Linux como Windows. (18)
Entre sus características se encuentran: (19)

- La capa de presentación utiliza plantillas Twig que pueden ser creadas por diseñadores HTML sin ningún tipo de conocimiento del marco de trabajo.

- Los formularios incluyen validación automatizada y relleno automático, lo que asegura la obtención de datos correctos.
- El sistema de enrutamiento y las URL limpias permiten considerar a las direcciones de las páginas como parte de la interfaz, además de estar optimizadas para los buscadores.

Es seleccionado Symfony2 como marco de trabajo por su facilidad para contribuir con el patrón arquitectónico MVC. Se integra con la mayoría de los mapeadores de objetos relacionales. Symfony automatiza el desarrollo de aplicaciones web. Se desea alcanzar un sistema seguro desde todos los puntos de vistas posibles. Symfony proporciona seguridad a las aplicaciones a través de la creación de roles y usuarios únicos en el sistema.

1.6.6.2. Doctrine ORM 2.3.2

"Un Mapeo de Objetos Relacional (ORM) consiste en una serie de objetos que permiten acceder a los datos y que contienen en su interior cierta lógica de negocio". (18)

Doctrine 2.3.2 está escrito en PHP y es una capa de abstracción que se sitúa justo encima de un Sistema Gestor de Base de Datos. Puede generar clases a partir de una base de datos existente y después se puede especificar relaciones y añadir funcionalidades extras a las clases autogeneradas. Una característica importante de Doctrine es la posibilidad de escribir consultas de base de datos utilizando un dialecto de SQL denominado Doctrine Query Language (DQL) que está inspirado en Hibernate4. (20) Doctrine es una librería muy completa y confiable, entre sus características más importantes se encuentran: (20)

- Permite la generación automática del modelo.
- Simplificación de la herencia.
- Facilidad de búsqueda.
- Relaciones entre entidades.
- Posee un lenguaje propio.

Es seleccionado Doctrine ORM pues evita utilizar una sintaxis específica de un sistema de bases de datos concreto. Además de su integración con el marco de trabajo Symfony2 y permite la generación automática del modelo.

[4] Herramienta que facilita el mapeo de atributos.

1.6.7. Lenguajes de programación

1.6.7.1. PHP 5.3

PHP[5] es un lenguaje del lado del servidor, de código abierto y muy popular especialmente en el desarrollo de páginas web dinámicas. PHP admite la mayoría de servidores web y simplifica los flujos de trabajo, de desarrollo y pruebas. Presenta soporte para un amplio abanico de bases de datos, puede comunicarse con otros servidores a través de protocolos. Es multiplataforma y libre, permite el paradigma de programación orientada a objetos, mejoras de rendimiento, manejo de excepciones, no requiere definición de tipos de variables. El código fuente escrito en PHP es invisible al navegador y al cliente, ya que es el servidor el que se encarga de ejecutar el código y enviar su resultado HTML al navegador, esto hace que la programación en PHP sea segura y confiable. (21)

Se utilizó PHP por todas las ventajas que aporta para el desarrollo de aplicaciones web, además de la integración de este lenguaje con el marco de trabajo seleccionado (18). Se integra perfectamente con muchos sistemas operativos incluyendo Linux que será la plataforma sobre la cual correrá la aplicación.

1.6.7.2. Motor de plantillas Twig 1.12

Twig es un motor y lenguaje de plantillas para PHP muy rápido y eficiente. La sintaxis de Twig ha sido diseñada para que las plantillas sean concisas y muy fáciles de leer y escribir. (22)
Sus principales características son: (23)

- Rápido: Twig compila las plantillas a código PHP sencillo optimizado. La sobrecarga del código PHP se ha reducido a la máxima expresión.
- Seguro: Twig tiene un modo de recinto para evaluar si el código de la plantilla no es de confianza. Esto te permite utilizar Twig como un lenguaje de plantillas para aplicaciones donde los usuarios pueden modificar el diseño de la plantilla.
- Flexible: Twig es alimentado por un analizador léxico flexible. Esto permite al desarrollador definir sus propias etiquetas y filtros personalizados.

Además permite la herencia entre plantillas, ya que crea un esqueleto de plantilla base que contenga todos los elementos comunes de la aplicación y define los bloques que las plantillas descendientes pueden sustituir. (24) Twig viene integrado como motor de plantilla por defecto al marco de trabajo seleccionado. Por todas las características mencionadas anteriormente es seleccionado el motor de plantillas Twig.

[5] PHP: por sus siglas en ingés Hipertext Processor

1.6.7.3. HTML5

HTML5[6] ha sido diseñado para que sea compatible con todo lo que se ha desarrollado en términos de publicación web. Pretende proporcionar una plataforma con la que se pueda desarrollar aplicaciones web más parecidas a las aplicaciones de escritorio, donde su ejecución dentro de un navegador no implique falta de recursos o facilidades para resolver las necesidades reales de los desarrolladores. Entre las características de HTML se encuentra que, contiene etiquetas para manejar grandes conjuntos de datos, se acopla perfectamente al motor de plantillas Twig, mostrando páginas bien organizadas a través de sus etiquetas. (25)

Es empleado en el proyecto por su seguridad sobre los navegadores más utilizados. HTML mejora la seguridad y la agilidad en el desarrollo de aplicaciones web.

1.6.7.4. Hojas de Estilo en Cascada (CSS3)

CCS3 ofrece nuevas posibilidades para los desarrolladores y sus diseños pues no se opone a las capacidades existentes de la web, en cambio aumentan las capacidades de un navegador para el soporte de mejores características y mejor presentación de contenido basado en HTML. Define nuevos estilos para los elementos de los bordes de un contenedor HTML. (26)

Es seleccionada para el desarrollo del sistema, ya que pueden ser utilizadas para lograr una apariencia uniforme de todo el sitio al activar una sola definición de estilo en cada página, hace que los códigos HTML sean más fáciles de leer ya que los estilos se definen por separado, permite que las páginas se carguen más rápido ya que hay menos cantidad de HTML en cada página y posiciona los elementos de la página de una manera más uniforme. (26)

1.6.7.5. JavaScript 1.2

JavaScript es un lenguaje de programación interpretado, se utiliza principalmente del lado del cliente permitiendo crear efectos atractivos y dinámicos en páginas web. (27) Es utilizado específicamente para validar los datos de los formularios directamente en el navegador del usuario, de esta forma se evita recargar la página cuando el usuario comete algún error al rellenar el formulario.

[6]HTML: por sus siglas en inglés Hypertext Markup Language

1.7. JQuery

JQuery es una biblioteca de JavaScript, que permite simplificar la manera de interactuar con los documentos HTML, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX[7] a páginas web, es un software libre y de código abierto. (28)

Al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio. Entre sus características más importantes se encuentran, la manipulación de la hoja de estilos CSS, presencia de efectos y animaciones personalizadas, modifica la apariencia y el contenido de la página y maneja eventos de los elementos de la página. (28).

1.8. Conclusiones parciales

Luego de finalizar el presente capítulo se puede arribar a las siguientes conclusiones:

- Las soluciones informáticas estudiadas por las características que poseen y las funcionalidades que cumplen, no son adaptables a las necesidades de la FGR, por lo que es necesario desarrollar un sistema para informatizar los procesos de inicio y diligencias que se ejecutan en las VF.
- La metodología de desarrollo seleccionada permitirá la organización, planificación y ejecución de las actividades a cumplir, haciendo énfasis en la generación de artefactos bien documentados.
- Las herramientas y tecnologías seleccionadas son libres y de código abierto, de manera que tribute a la política de migración a software libre que lleva a cabo la UCI y la FGR. Las mismas facilitarán el desarrollo de la solución para la obtención de un producto final con la calidad requerida.

[7]AJAX: por sus siglas en inglés Asynchronous JavaScript And XML. Es una técnica de desarrollo web para crear aplicaciones interactivas.

CAPÍTULO 2: DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

En el presente capítulo se exponen los requisitos funcionales asociados a los procesos de inicio y diligencias del módulo VF que fueron definidos en la etapa de análisis. Se describe el diseño del sistema mediante los diferentes artefactos generados en esta etapa como es el caso de los diagramas de clases, de secuencia, de vista, las controladoras y gestoras, haciendo uso de patrones de diseño. Se valida el diseño empleando las métricas correspondientes. Se describe la arquitectura del sistema, así como el patrón arquitectónico a utilizar. Finaliza el capítulo con la fase de construcción, donde se presenta el diagrama de componentes, el modelo de despliegue y los estándares de codificación seleccionados.

2.1. Requisitos del sistema.

Una buena elección de los requisitos trae consigo identificar de forma correcta las funcionalidades del sistema y restricciones necesarias para su posterior desarrollo. Los requisitos proponen entender lo que desea el cliente analizando sus necesidades y negociando hacia una solución razonable. Los requisitos funcionales que pertenecen a los procesos de inicio y diligencias se evidencian en la tabla que se muestra a continuación.

Procesos inicio y diligencias	
Número	**Requisito Funcional**
Gestionar solicitud sorpresiva de la acción	
RF_VF_VF_1	Listar solicitudes de aprobación de inicio de acción fiscal sorpresiva
RF_VF_VF_2	Adicionar solicitud de aprobación de inicio de acción fiscal sorpresiva
RF_VF_VF_3	Actualizar solicitud de aprobación de inicio de acción fiscal sorpresiva
RF_VF_VF_4	Vista previa de la solicitud de aprobación de inicio de acción fiscal sorpresiva
RF_VF_VF_5	Adicionar constancia de aprobación de solicitud de acción fiscal sorpresiva
Gestionar Disposición sobre Solicitud sorpresiva de la acción	
RF_VF_VF_6	Listar disposiciones sobre solicitud sorpresiva de la acción
RF_VF_VF_7	Adicionar disposición sobre solicitud sorpresiva de la acción
RF_VF_VF_8	Actualizar disposición sobre solicitud sorpresiva de la acción
RF_VF_VF_9	Vista previa de la aceptación de solicitud de inicio de acción sorpresiva
RF_VF_VF_10	Vista previa de la denegación de solicitud de inicio de acción sorpresiva
Gestionar Plan Trabajo	
RF_VF_VF_11	Listar plan de trabajo
RF_VF_VF_12	Adicionar plan de trabajo
RF_VF_VF_13	Actualizar plan de trabajo

RF_VF_VF_14	Vista previa del plan de trabajo
RF_VF_VF_15	Listar objetivos del plan de trabajo
RF_VF_VF_16	Adicionar objetivo del plan de trabajo
RF_VF_VF_17	Eliminar objetivo del plan de trabajo
RF_VF_VF_18	Actualizar objetivo del plan de trabajo
RF_VF_VF_19	Visualizar detalles del objetivo del plan de trabajo
RF_VF_VF_20	Listar Tareas
RF_VF_VF_21	Adicionar Tarea
RF_VF_VF_22	Eliminar Tarea
RF_VF_VF_23	Actualizar Tarea
RF_VF_VF_24	Detalles del Tarea
Gestionar Comunicación de inicio	
RF_VF_VF_25	Listar comunicaciones de inicio
RF_VF_VF_26	Adicionar comunicación de inicio
RF_VF_VF_27	Actualizar comunicación de inicio
RF_VF_VF_28	Vista previa de la comunicación de inicio
RF_VF_VF_29	Adicionar constancia de notificación de comunicación de inicio.
Gestionar Acta de reunión de inicio de la VF o IF	
RF_VF_VF_30	Listar actas de reunión de inicio
RF_VF_VF_31	Adicionar acta de reunión de inicio
RF_VF_VF_32	Actualizar acta de reunión de inicio
RF_VF_VF_33	Mostrar vista previa del acta de reunión de inicio
RF_VF_VF_34	Adicionar constancia de firma de acta de reunión de inicio.
Gestionar Despacho	
RF_VF_VF_35	Listar Despachos
RF_VF_VF_36	Adicionar Despacho
RF_VF_VF_37	Actualizar Despacho
RF_VF_VF_38	Mostrar vista previa del Despacho
RF_VF_VF_39	Adicionar constancia de notificación del despacho.
RF_VF_VF_40	Adicionar constancia de respuesta del despacho.
Gestionar Acta de Entrevista	
RF_VF_VF_41	Listar Actas de Entrevista
RF_VF_VF_42	Adicionar Acta de Entrevista
RF_VF_VF_43	Actualizar Acta de Entrevista
RF_VF_VF_44	Vista previa del Acta de Entrevista
Gestionar Acta de Ocupación	

RF_VF_VF_45	Listar Actas de Ocupación
RF_VF_VF_46	Adicionar Acta de Ocupación
RF_VF_VF_47	Actualizar Acta de Ocupación
RF_VF_VF_48	Vista previa del Acta de Ocupación
Gestionar Acta de Entrega	
RF_VF_VF_49	Listar Actas de Entrega
RF_VF_VF_50	Adicionar Acta de Entrega
RF_VF_VF_51	Actualizar Acta de Entrega
RF_VF_VF_52	Vista previa del Acta de Entrega
Gestionar Informe de Especialistas	
RF_VF_VF_53	Listar Informes de Especialistas
RF_VF_VF_54	Adjuntar Informe de Especialistas
RF_VF_VF_55	Vista previa del Informe de Especialistas

Tabla 2: Requisitos funcionales de los procesos de inicio y diligencias *(22)*

2.2. Arquitectura del sistema

"La arquitectura de un sistema es el diseño o conjunto de relaciones entre las partes que constituyen un sistema". (23)

Según Pressman un patrón de arquitectura "*define la estructura general del software, indica las relaciones entre los subsistemas y los componentes del software y definen las reglas para especificar las relaciones entre los elementos (clases, paquetes, componentes, subsistemas) de la arquitectura".* (24)

La arquitectura del proyecto SIGEF II está basada en el patrón arquitectónico Modelo Vista Controlador, agregándole a la capa del modelo otra encargada de gestionar el negocio del sistema, denominada Negocio.

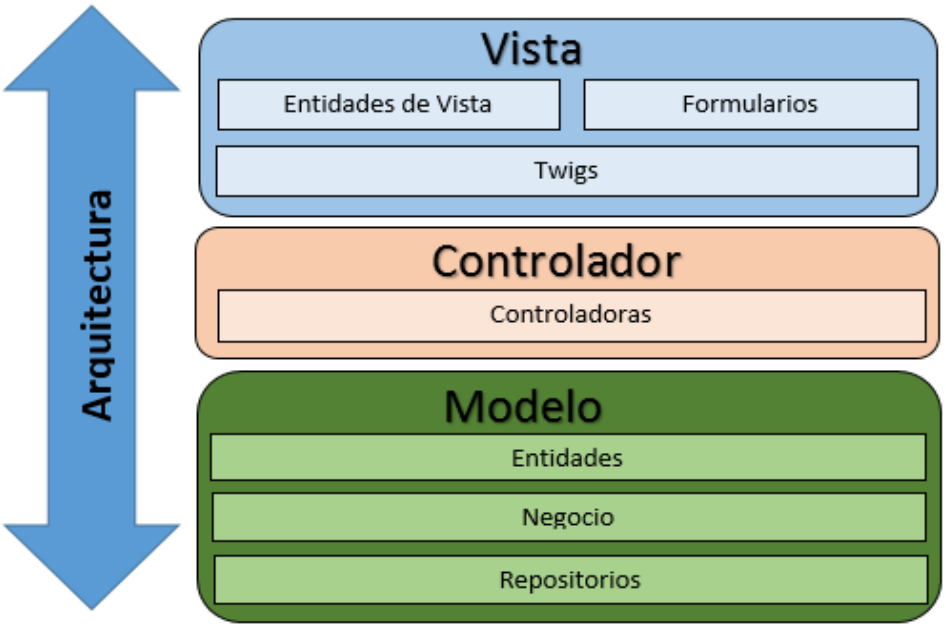


Figura 3: Representación de la Arquitectura

La figura anterior muestra las capas por la que está compuesta la arquitectura del sistema, las cuales se explican a continuación.

Modelo: la capa modelo es la que contiene las clases entidades, repositorios y gestoras, quienes se encargan del manejo de los datos. Las clases gestoras se encuentran en el directorio "SIGEFII/src/VF/Negocio", estas separan la lógica del negocio del resto de la aplicación, facilitando la reutilización del código. Las clases repositorios se encuentran en el directorio "SIGEFII/src/Adm/Repository/VF", donde se encuentran las consultas a la base de datos. La clases entidades se encuentran dentro del directorio "SIGEFII/src/Adm/Entity/VF", las cuales representan las tablas de la base de datos previamente mapeadas por Doctrine.

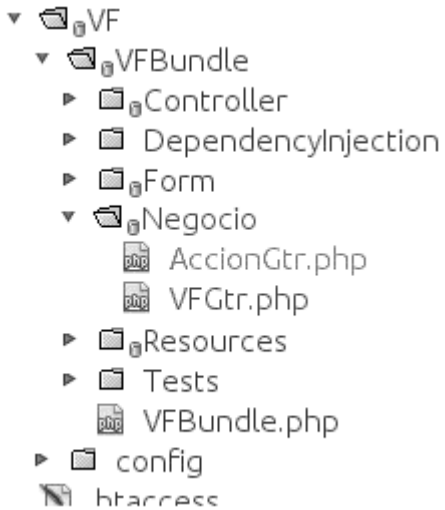


Figura 4: Clase Gestora

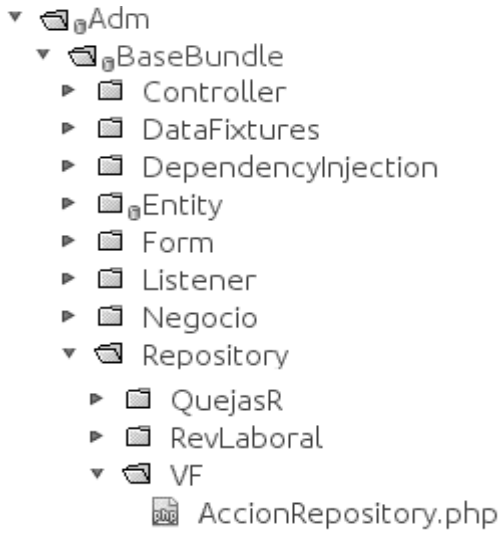


Figura 5: Calse Repository

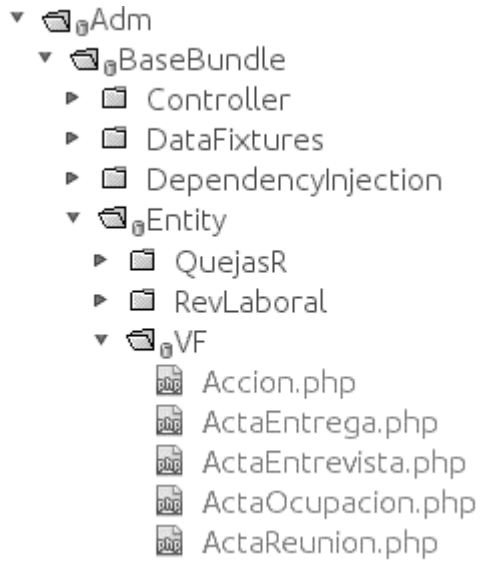


Figura 6: Entidades

Vista: la capa vista es la que interactúa directamente con los usuarios finales a través de clases Twig y formularios. La plantilla layoutBase.html.twig que se encuentra en el directorio "SIGEFII/src/Adm/Resource/views/" y las plantillas que se ubican en el directorio "SIGEFII/src/VF/Resource/views/Accion" son las encargadas de transformar los datos del modelo en vistas y le permiten al usuario interactuar con la aplicación.

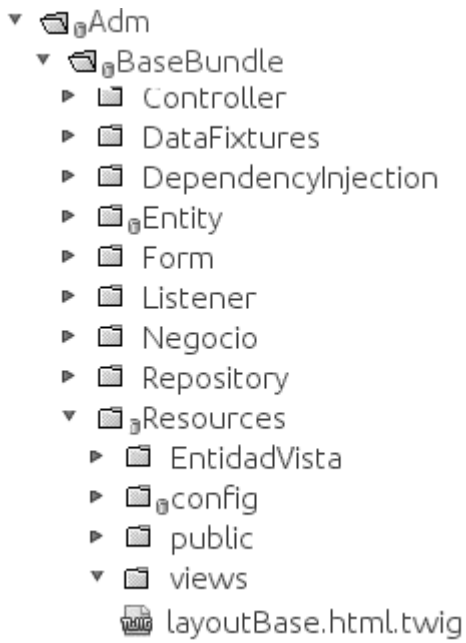


Figura 7: Plantilla base

Figura 8: Plantillas de VF

Controlador: es el intermediario entre la vista y el modelo. La capa controlador se encarga de recibir una petición del usuario, procesar la información, hacer un pedido al modelo y devolver una respuesta al usuario mediante las vistas. Esta clase se encuentra en el directorio “SIGEFII/src/VF/Controller”.

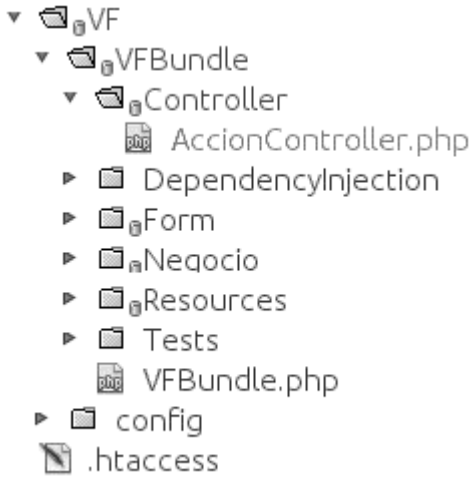


Figura 9: Capa controlador en VF

Como propuesta de arquitectura empleada por el proyecto SIGEF II se incorpora un subsistema denominado arquitectura base, la cual sirve de apoyo en el desarrollo del módulo VF, ya que recoge el diseño e implementación de todas las funcionalidades comunes. Los componentes que se utilizan para la creación de los objetos en los módulos aseguran un adecuado manejo transaccional, un control de excepciones y trazas, mensajerías, seguridad y enrutamiento. De esta manera no solo se reduce la implementación de los requisitos funcionales básicos, sino que el código es altamente reutilizable y se limitan los errores o vulnerabilidades en el mismo. (14)

La arquitectura incorpora como componente de seguridad para la navegación y acceso al sistema el módulo ACAXIA, a través del cual es posible gestionar los distintos departamentos, así como las funcionalidades y el trabajo a través de roles y usuarios.

2.3. Diseño del sistema

El diseño es una de las fases principales en el proceso de desarrollo de software. Este abarca desde la planificación previa hasta el aspecto final que tendrá. Además descompone los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo y facilita la implementación. (25)

Para diseñar los procesos de inicio y diligencias del módulo VF son empleados un conjunto de patrones de diseño que constituyen una solución simple y elegante a un determinado problema, de forma tal que esa solución pueda ser utilizada en un futuro.

2.3.1. Patrones de diseño

El objetivo principal de la utilización de los patrones de diseño es que permite la reutilización de código, diseño de objetos, diseño de sistemas, diseño de análisis y como características fundamentales facilitan la reutilización de las clases y del propio diseño. (26)

Durante el diseño del sistema es necesario asignar correctamente las responsabilidades, para poner en práctica de forma correcta esta tarea es necesario hacer uso de los patrones GRASP[8], estos describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. (27) A continuación se explican cada uno de ellos.

[8] GRASP: Patrones Generales de Software para Asignación de Responsabilidades.

Experto: Este patrón plantea que la responsabilidad debe ser asignada al experto en la información, es decir a la clase que cuenta con la información necesaria para cumplir con una responsabilidad dada. El uso de este patrón se observa en todas las clases entidades, un ejemplo lo constituye la clase Despacho (Figura 10), pues tiene toda la información necesaria de cada uno de los objetos que se manejan en la aplicación. Además se pone en práctica en la clase gestora AccionGtr, pues es la que gestiona la información necesaria acerca del funcionamiento de la lógica de negocio.

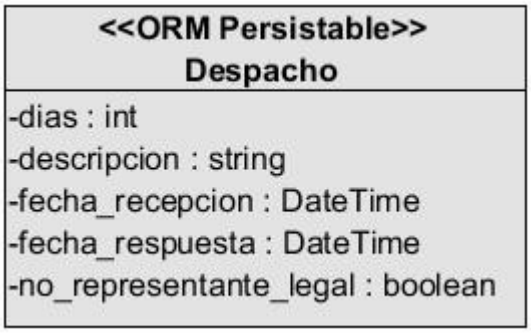


Figura 10: Uso del patrón Experto

Creador: Este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos. Se encarga de asignarle a una clase A la responsabilidad de instanciar una clase B. Es utilizado en la clase AccionGtr, pues es el encargado de crear los objetos de las clases que representan las entidades del negocio, por lo que se evidencia que la clase gestora es creadora de dichas entidades. Un ejemplo se representa en la acción crearDocumentoDespacho (figura 12), donde en caso de no estar ese tipo de documento se crea una instancia de la clase entidad Despacho que contiene los datos necesarios.

Figura 11: Método en el Gestor

```
public function crearDocumentoDespacho($id_doc, $proceso) {
    if (is_null($id_doc)) {
        $despacho = new Despacho();
        $tipo_doc = $this->getDocumentoGtr()->cargarTipoDocumento(ConfigUtilVF::doc_despacho);
        $despacho->setTipoDocumento($tipo_doc);
        $despacho->setProceso($proceso);
    } else
        $despacho = $this->getDocumentoGtr()->cargarDocumento($id_doc);
    $despacho->setProceso($proceso);
    return $despacho;
}
```

Figura 12: Uso del patrón Creador

Bajo Acoplamiento: Este patrón se encarga de relacionar las clases entre sí lo menos posible para eliminar la dependencia entre clases, así cuando alguna sufra algún cambio no repercuta en las demás. Su uso se muestra en la clase AccionController, pues al heredar de la clase ProcesoController, ésta última puede necesitar modificaciones y la clase AccionController no será afectada.

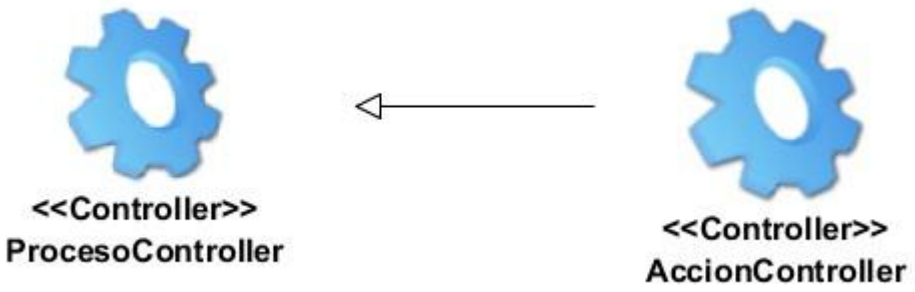


Figura 13: Uso del patrón Bajo Acoplamiento

Alta Cohesión: Se evidencia en todas las clase del diseño propuesto, las cuales están formadas por funcionalidades relacionadas entre sí proporcionando de esta forma la flexibilidad del sistema ante cambios inesperados. Por ejemplo en la clase controladora AccionController no existe demasiada carga pues cuenta con las funcionalidades necesarias para realizar una acción.

Controlador: El patrón se observa principalmente en la clase controladora AccionController que es la intermediaria entre la gestora y las interfaces. La clase controladora recibe los datos y los distribuye a las demás clases en correspondencia a la función a realizar. Su objetivo es aumentar la reutilización de código separando la lógica de negocio de la capa de presentación.

Otro conjunto de patrones que son utilizados, son los llamados patrones GoF[9], estos patrones según su propósito se clasifican en creacionales, estructurales y de comportamiento. Los creacionales se basan en la creación de instancias de objetos. Los estructurales describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Los patrones de comportamiento definen la comunicación entre los objetos del sistema. (28) Dentro de este grupo se utilizan específicamente Fábrica Abstracta, Solitario y Decorador.

Fábrica Abstracta: Patrón creacional que proporciona una interfaz para la creación de familias de objetos, pero delega la responsabilidad de instanciarlo. Se utilizó en la clase AccionController para la creación de formularios, mediante el método createform().

Solitario: Patrón creacional, este garantiza que solamente se cree una instancia de la clase y proporciona un punto de acceso global a este objeto. Un ejemplo de este se pone de manifiesto en la clase Controller la cual en cualquier lugar de la aplicación brinda acceso a la información del usuario que está autenticado en el sistema.

Decorador: Patrón estructural que añade funcionalidad a una clase dinámicamente, se aplica en la generación de las vistas, añadiendo elementos comunes y reutilizables para varias plantillas como el menú o el pie de página. Un ejemplo de esto es el archivo layoutVF.html.twig que contiene el código común para todas las demás páginas, las páginas que se crean heredan de ésta y así se decoran todas las páginas del módulo.

2.4. Artefactos generados

Haciendo uso de la herramienta Visual Paradigm y el lenguaje de modelado UML se generaron los diferentes artefactos durante la fase del diseño. A continuación se muestra un ejemplo de cada uno de estos diagramas diseñados para la funcionalidad Gestionar Despacho, el cual constituye una de las funcionalidades más significativas dentro del sistema.

2.4.1. Diagrama de Clases Persistentes

La Figura 14 permite reflejar las clases con las cuales trabaja la funcionalidad Gestionar Despacho. Muestra además las relaciones entre las clases, las cuales pueden ser de herencia y asociativas, esta

[9] GoF: Grupo de los Cuatro, por sus siglas en inglés Gang of Four

última especificando la cardinalidad entre ellas, así como sus atributos con sus respectivos nombres y tipo de dato. Las clases representadas con el color verde son las que pertenecen al subsistema base de la aplicación, permitiendo reutilización y facilidades a la hora de implementación. La clase principal de este diagrama es Despacho. Para ver el diagrama de clases correspondientes a los procesos de inicio y diligencias del módulo VF, ver Anexo 1.

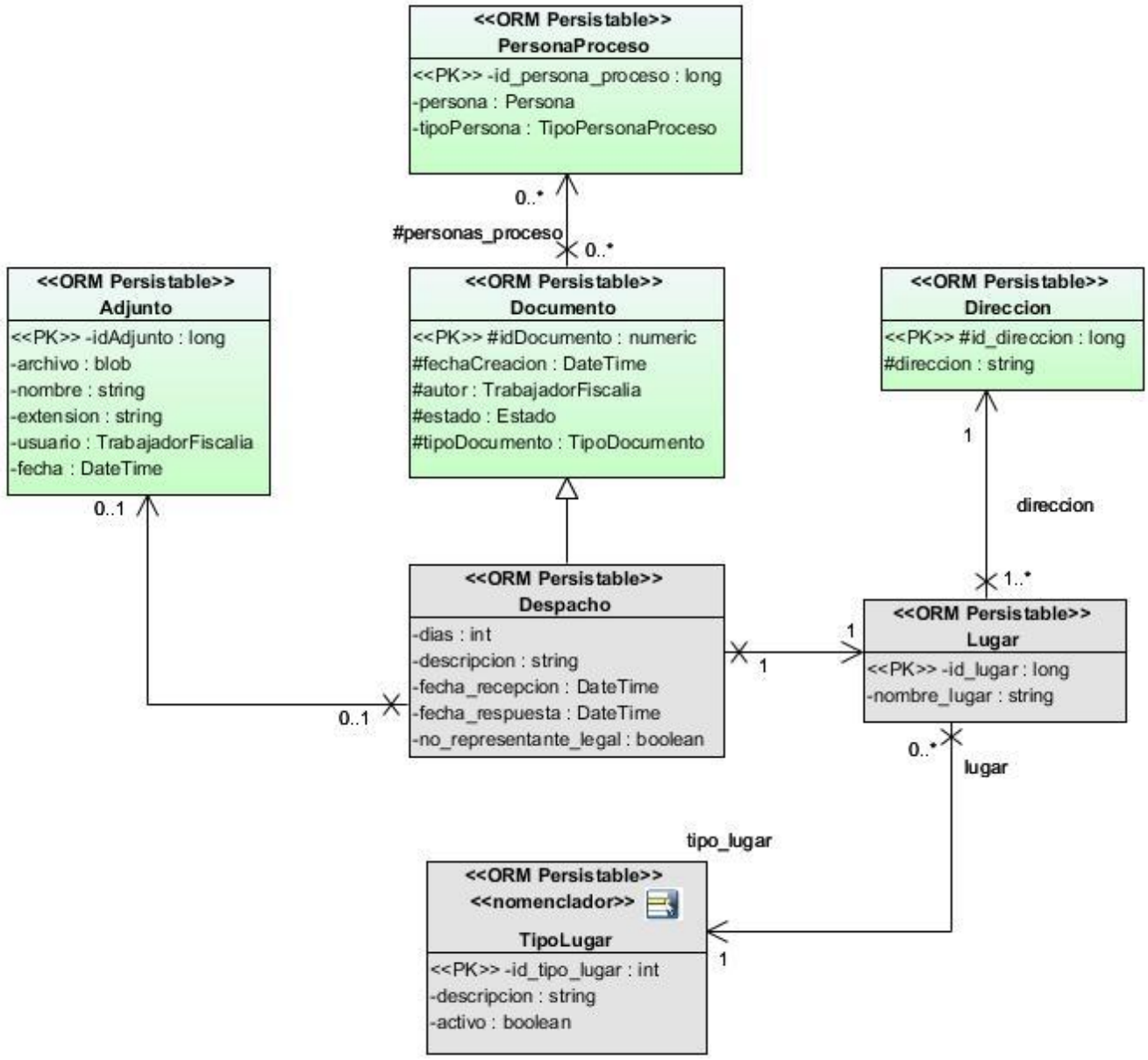


Figura 14: Diagrama de clases persistentes de la funcionalidad Gestionar Despacho

2.4.2. Diagrama de clases Controladoras y Gestoras

En la Figura 15 se muestra el diagrama correspondiente a las clases controladoras y gestoras, donde se puede observar las relaciones existentes entre ellas. La clase ProcesoController y ProcesoGtr contienen todas las funcionalidades que son comunes para todos los módulos evitando la duplicidad de métodos en las clases. Las clases controladoras gestionan el flujo de la aplicación, encapsulan el comportamiento de una funcionalidad y realizan las operaciones más complejas. Estas interactúan con las clases repositorios a través de las clases gestoras. Las clases controladoras y gestoras son incluidas en el mismo diagrama por la estrecha relación que existe entre ellas a la hora de ejecutar alguna operación dentro del sistema. El diagrama de controladoras y gestoras correspondiente a los procesos de inicio y diligencias del módulo VF, se encuentra en el Anexo 2 en este documento.

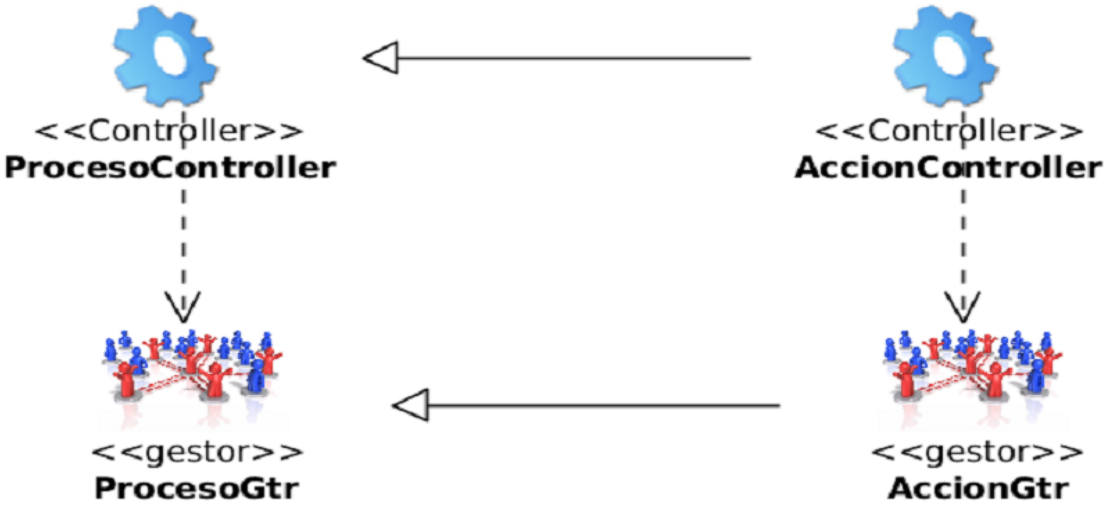


Figura 15: Diagrama de clases Controladoras y Gestoras para la funcionalidad Gestionar Despacho

2.4.3. Diagrama de clases de repositorio

El diagrama de clases de repositorio define las clases que van a interactuar con la base de datos, las consultas pueden ser realizadas mediantes los métodos básicos find(), findAll(), findBy() y findOneBy(), los cuales son necesarios para obtener los datos que requiere la aplicación para su ejecución. En la Figura 16 se muestra la relación de herencia entre la clase del módulo VF AccionRepository y la clase común

EntityRepository. Se evidencia además la operación a realizar por la clase del módulo que permite obtener todos los despachos adicionados al sistema.

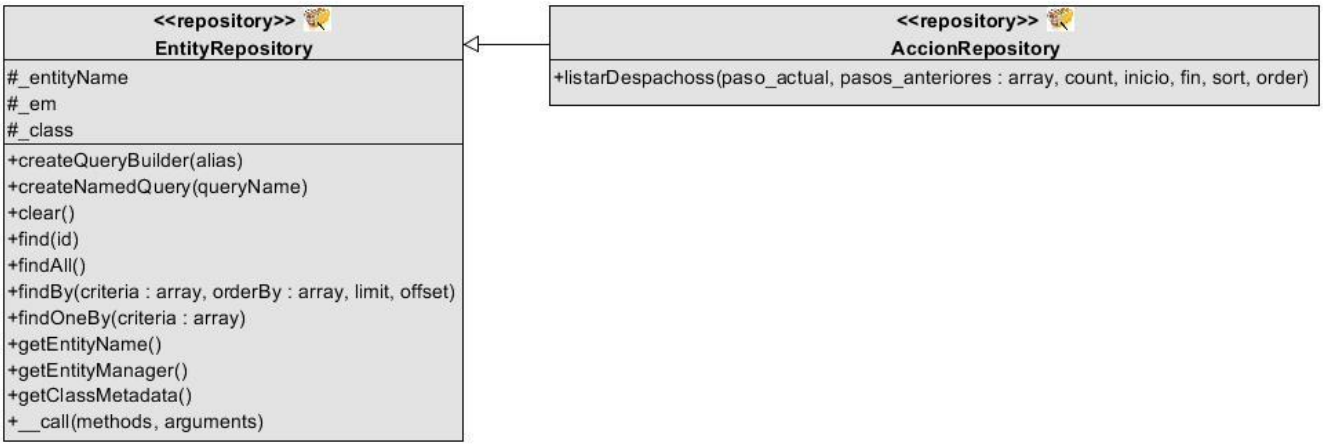


Figura 16: Diagrama de clases de Repositorio para la funcionalidad Gestionar Despacho

2.4.4. Diagrama de clases de Secuencia

Los diagramas de secuencia son fundamentales en la implementación de un software ya que representan la interacción entre los componentes del sistema. Estos diagramas muestran los componentes como líneas de vida a lo largo de la página, y las interacciones en el tiempo mediante los mensajes, son representadas con flechas desde la línea de vida origen hasta la línea de vida destino. (29) A continuación se muestra el diagrama de secuencia de la funcionalidad Adicionar y Actualizar Despacho, los restantes diagramas de secuencia de la funcionalidad Gestionar Despacho se encuentran en el Anexo 4 en este documento.

En la Figura 17 el usuario selecciona la opción Adicionar en la página inicial donde se muestra el listado de todos los despachos adicionados previamente al proceso de verificación o investigación fiscal. La acción activa la operación correspondiente en la clase controladora, donde verifica si ese proceso se encuentra en edición, en caso de ser negativo entonces se puede adicionar o actualizar un despacho. Si se selecciona la funcionalidad Adicionar se muestra un formulario para que el usuario introduzca los datos del nuevo despacho. En caso de seleccionar alguno existente para actualizarlo, se obtiene el documento correspondiente y se muestra un formulario con los datos del despacho existente. Cuando se selecciona la opción Guardar, el sistema almacena los datos ingresados o modificados y se retorna a la página de inicio. En caso de seleccionar la opción Finalizar, se ejecuta la opción de guardar, se cambia el estado del documento a finalizado y se retorna a la página de inicio.

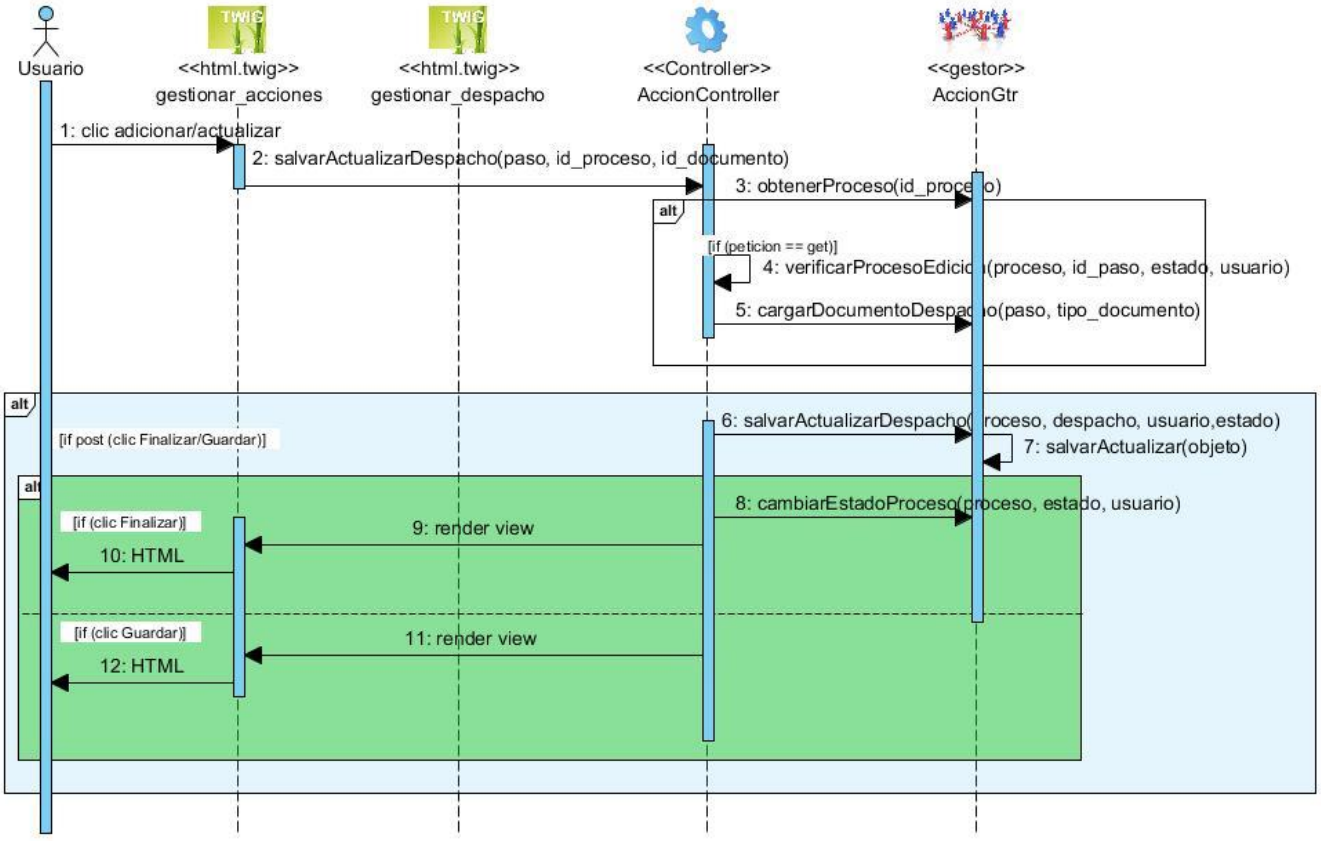


Figura 17: Diagrama de secuencia de la funcionalidad Adicionar y Actualizar Despacho

2.4.5. Diagrama de clases de vistas

Los diagramas de vistas le permiten a los desarrolladores obtener una mejor comprensión del sistema a la hora de su implementación. Estos diagramas muestran la navegación básica por la que debe atravesar el sistema. La Figura 18 representa el diagrama de vistas para la funcionalidad Adicionar y Actualizar Despacho, donde se puede observar la relación entre las clases Twig, los formularios y las clases JavaScript. Este diagrama está compuesto por 11 clases Twig, 3 JavaScript y 4 formularios. El diagrama de vista de los proceso de inicio y diligencias del módulo VF se encuentra en el Anexo 3.

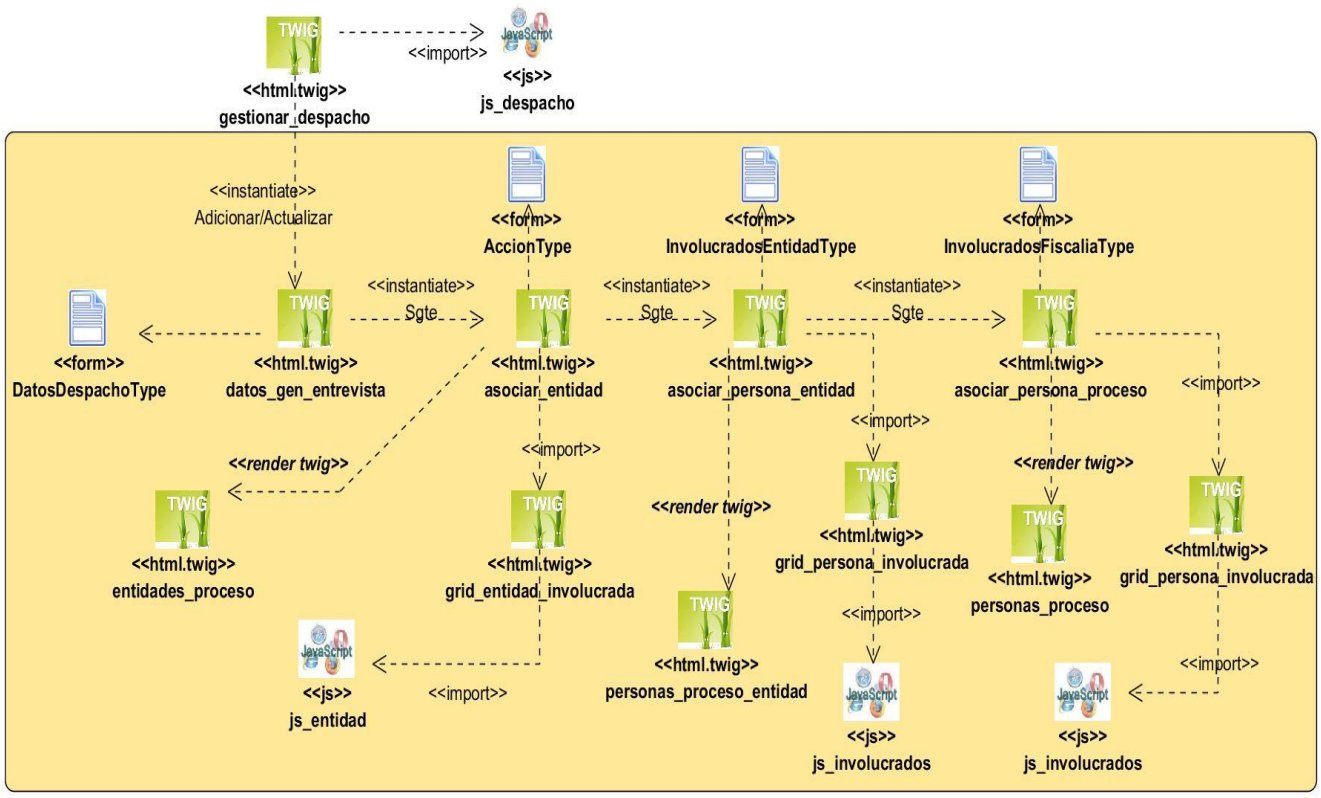


Figura 18: Diagrama de clases de vista para la funcionalidad Adicionar y Actualizar Despacho

2.5. Métricas para la medición del diseño

Cuando se construye un sistema informático, se trata de que éste cumpla con todas las características previstas en la fase de análisis para lograr la completa satisfacción del cliente y calidad del software. Una métrica se puede definir como la aplicación continua de mediciones basadas en técnicas para el proceso de desarrollo del software y sus productos para suministrar información relevante a tiempo, así el administrador junto con el empleo de estas técnicas mejorará el proceso y sus productos. (30)

La medición persigue tres objetivos fundamentales:

1. Entender ¿Qué ocurre durante el desarrollo y el mantenimiento?
2. Controlar ¿Qué es lo que ocurre en los proyectos?
3. Mejorar los procesos y los productos.

Las métricas de diseño permiten medir de forma cuantitativa la calidad de los atributos internos del software. Las empleadas en la evaluación del diseño del sistema son: Tamaño Operacional de Clases y Acoplamiento entre clases.

2.5.1. Tamaño Operacional de Clases (TOC)

Plantea que el tamaño general de una clase se puede determinar a partir de la suma del número de atributos (tanto heredados como privados de la instancia) que están encapsulados en la clase, más el número total de operaciones (tanto heredado como privado de la instancia) también encapsuladas. Estos valores son sumados para obtener un **umbral**, se le haya el promedio a este valor y se evalúan los atributos de calidad que se muestran en la Tabla 4. (31)

Un aumento del TOC implica un aumento en la responsabilidad de las clases, así como en la complejidad de mantenimiento y una disminución del grado de reutilización de las clases, lo que puede complicar la implementación del sistema. (31)

Clasificación de las clases	Valor de los umbrales
Pequeño	Umbral <=20
Medio	20<Umbral<30
Grande	Umbral >30

Tabla 3: Clasificación de las clases según su tamaño (31)

	Categoría	Criterio
Responsabilidad	Baja	Umbral< =Prom
	Madia	Prom <=Umbral<= 2* Pom
	Alta	Umbral> 2* Prom
Complejidad Implementación	Baja	Umbral < =Prom
	Media	Prom <=Umbral<= 2* Pom
	Alta	Umbral> 2* Prom
Reutilización	Baja	Umbral > 2* Prom
	Media	Prom <=Umbral<= 2* Pom
	Alta	Umbral < =Prom

Tabla 4: Rango de valores para la evaluación de los atributos de calidad (31)

A continuación se muestran los resultados obtenidos tras la aplicación de la métrica TOC, donde:

CP: cantidad de procedimientos (umbral) CI: complejidad de implementación

Res: responsabilidad de las clases Reu: reutilización de la clase

N	Clases	CP	Tamaño	Res	Cl	Reu
1	Documento	53	Grande	Alta	Alta	Bajo
2	Acción	71	Grande	Alta	Alta	Bajo
3	Direccion	31	Grande	Medio	Medio	Medio
4	TipoLugar	9	Pequeña	Baja	Baja	Alta
5	Lugar	12	Pequeña	Baja	Baja	Alta
6	Adjunto	34	Grande	Media	Media	Media
7	Aprobacion	12	Pequeña	Baja	Baja	Alta
8	Disposicion	11	Pequeña	Baja	Baja	Alta
9	Proceso	75	Grande	Alta	Alta	Baja
10	PersonaProceso	20	Pequeña	Baja	Baja	Alta
11	PlanTrabajo	9	Pequeña	Baja	Baja	Alta
12	Objetivo	8	Pequeña	Baja	Baja	Alta
13	Tarea	13	Pequeña	Baja	Baja	Alta
14	SolicitudSorpresivaAccion	6	Pequeña	Baja	Baja	Alta
15	ComunicacionInicio	12	Pequeña	Baja	Baja	Alta
16	ActaEntrevista	13	Pequeña	Baja	Baja	Alta
17	ActaEntrega	13	Pequeña	Baja	Baja	Alta
18	ActaOcupacion	13	Pequeña	Baja	Baja	Alta
19	Despacho	28	Medio	Media	Media	Media
20	InformeEspecialista	37	Grande	Media	Media	Media
21	TipoAccionControl	9	Pequeña	Baja	Baja	Alta
Prom		22.3				

Tabla 5: Representación del tamaño de la clase

Se trabajó con un total de 21 clases, las cuales arrojaron un promedio de 22.3 de cantidad de procedimientos.

A continuación se representa mediante gráficas los atributos de calidad Responsabilidad, Complejidad de mantenimiento y Reutilización.

Figura 19: Porcentaje de clases agrupadas en las clasificaciones según su Responsabilidad

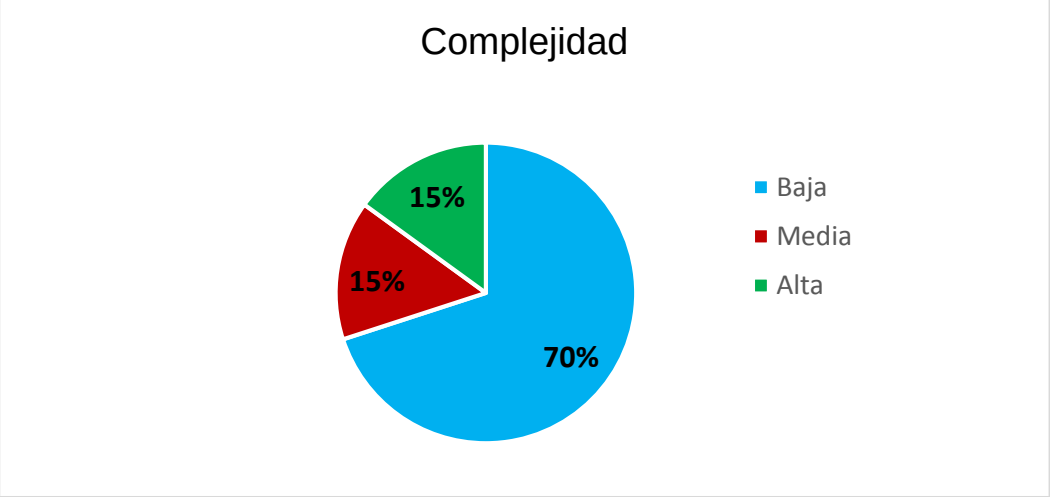


Figura 20: Porcentaje de clases agrupadas en las clasificaciones según su Complejidad

Figura 21: Porcentaje de clases agrupadas en las clasificaciones según su Reutilización

Haciendo un análisis de los resultados obtenidos para los atributos de la métrica TOC, se puede concluir que las clases del diseño de los procesos de inicio y diligencias del módulo VF son pequeñas en su mayoría. No presentan una alta responsabilidad, ni un elevado nivel de complejidad en el mantenimiento y si cuentan con una amplia reutilización de clases, por lo que se concluye que los resultados son positivos.

2.5.2. Acoplamiento entre Clases (AC)

El AC es el número de asociaciones de uso de una clase con otra. Se da dependencia entre dos clases cuando una clase usa métodos o variables de la otra clase. Las clases relacionadas por herencia no se tienen en cuenta. Cuanto más alto sea el acoplamiento, más difícil será reutilizarla y más rigurosas han de ser las pruebas requeridas sobre las clases involucradas. (32) A continuación se muestran los atributos de calidad que serán evaluados en el sistema.

	Categoría	Criterio
Acoplamiento	Ninguno	Asociaciones de uso< =Promedio
	Bajo	Promedio<= Asociaciones de uso <= 2* Promedio
	Medio	Asociaciones de uso > 2* Promedio
Complejidad mantenimiento	Baja	Asociaciones de uso < =Promedio

	Media	Promedio<= Asociaciones de uso <= 2* Promedio
	Alta	Asociaciones de uso > 2* Promedio
Reutilización	Baja	Asociaciones de uso > 2* Promedio
	Media	Promedio<= Asociaciones de uso <= 2* Promedio
	Alta	Asociaciones de uso < =Promedio

Tabla 6: Rango de valores para la evaluación de los atributos de calidad (32)

La evaluación de los atributos por cada clase se muestra en la tabla 7. Mientras que los criterios y categorías de la aplicación de la métrica se pueden observar en la tabla 8.

No	Clases	Cantidad de relaciones de uso	Acoplamiento	Complejidad Mantenimiento	Reutilización
1	Documento	1	Bajo	Baja	Alta
2	Acción	1	Bajo	Baja	Alta
3	Direccion	0	Ninguno	Baja	Alta
4	TipoLugar	0	Ninguno	Baja	Alta
5	Lugar	2	Medio	Media	Media
6	Adjunto	0	Ninguna	Baja	Alta
7	Aprobacion	1	Bajo	Baja	Alta
8	Disposicion	1	Bajo	Baja	Alta
9	Proceso	2	Medio	Media	Media
10	PersonaProceso	1	Bajo	Baja	Alta
11	PlanTrabajo	3	Alto	Alta	Baja
12	Objetivo	1	Bajo	Baja	Alta
13	Tarea	1	Bajo	Baja	Alta
14	SolicitudSorpresivaAccion	3	Alto	Alta	Baja
15	ComunicacionInicio	3	Alto	Alta	Baja
16	ActaEntrevista	2	Medio	Media	Media
17	ActaEntrega	2	Medio	Media	Media
18	ActaOcupacion	2	Medio	Media	Media
19	Despacho	4	Alto	Alta	Baja

20	InformeEspecialista	2	Medio	Media	Media
21	TipoAccionControl	0	Ninguno	Baja	Alta
Prom		1.52			

Tabla 7: Resultados evaluados de la métrica CBO

Criterio	Categoría	Cant de clases	Promedio
0 dependencias	Muy bueno	0	0
1 dependencias	Bueno	6	0.27
2 dependencias	Regular	7	0.31
3 dependencias	Malo	5	0.22
>3 dependencias	Muy Malo	3	0.18
Total		21	

Tabla 8: Criterios y categorías obtenidos en la aplicación de la métrica

Para un total de 21 clases y un promedio de asociaciones de uso de 1.52 se obtienen los resultados que se muestran en las siguientes figuras:

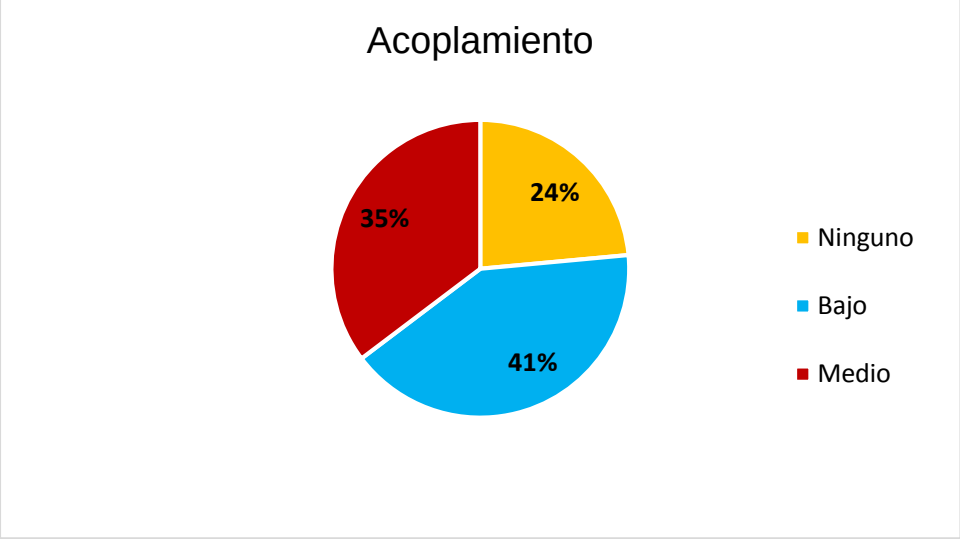


Figura 22: Porcentaje de clases agrupadas en las clasificaciones según su Acoplamiento

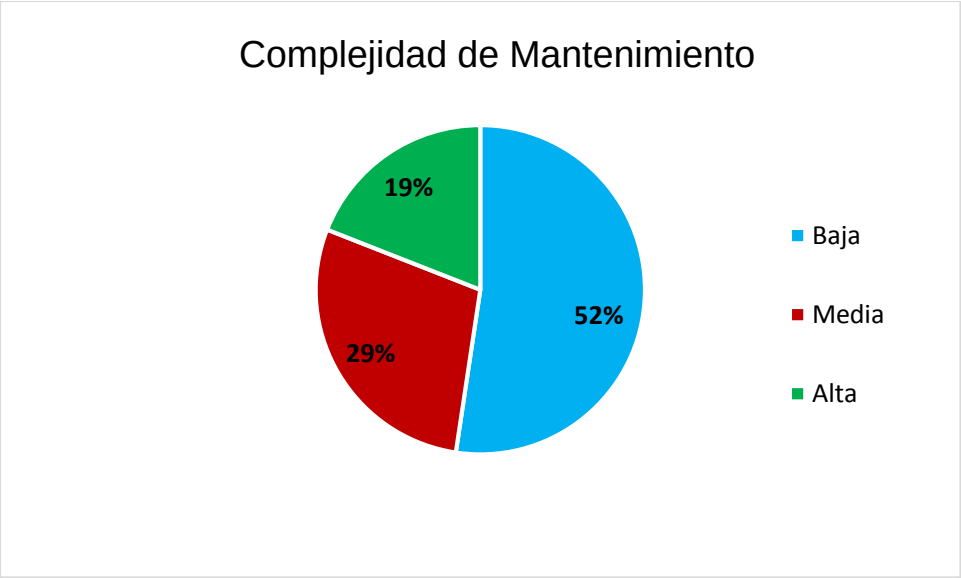


Figura 23: Porcentaje de clases agrupadas en las clasificaciones según su Complejidad

Figura 24: Porcentaje de clases agrupadas en las clasificaciones según su Reutilización

La aplicación de dicha métrica arrojó que la mayoría de las clases tienen entre 1 y 2 dependencias, lo que permite interpretar que el diseño obtenido se encuentra entre los límites aceptables de calidad, ya que el acoplamiento en su mayoría es bajo, haciendo más fácil reutilizar las clases. Además se puede percibir que la complejidad de mantenimiento al sistema será mínimo.

2.6. Implementación

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

2.6.1. Modelo de implementación

Luego de concluida la fase de diseño y la validación de este, le sigue la de implementación. En esta última se construye el sistema en términos de componentes. A continuación se realiza una descripción de cada uno de los artefactos generados, estos son el diagrama de componentes, el modelo de despliegue y los archivos de código fuente, además de los estándares de codificación utilizados en el desarrollo. El modelo de implementación está conformado de acuerdo a la arquitectura que propone el proyecto. El correcto funcionamiento del sistema depende en gran medida de la relación que exista entre los diferentes componentes descritos en los diagramas.

2.6.1.1. Diagrama de componentes

El diagrama de componentes del módulo VF muestra la forma en que están estructurados los componentes de acuerdo a la arquitectura, así como la dependencia entre ellos. En un diagrama de componentes se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software, la reutilización y las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo. (28)

La Figura 25 muestra el diagrama de componente del módulo VF, donde se puede visualizar la estructura general del sistema y el comportamiento de los servicios que proporciona. SIGEF II incluye los componentes:

- Routing: Gestiona las rutas de acceso al sistema.
- Config: Posee las principales configuraciones.
- Security: Se encarga de la seguridad del sistema.
- Controlador Frontal: Recibe cada petición realizada al sistema.

El paquete Arquitectura Base está compuesto por:

- Servicios.
- Componentes.
- Seguridad.

En el subsistema VF se encuentran los componentes:

- Mensajería: Se encarga de gestionar los mensajes que deben ser mostrados.
- Excepciones: Se emplea para la captura y tratamiento de las excepciones generadas.

Dentro de este subsistema se encuentra el módulo VF, el cual está compuesto por tres paquetes.

- Controlador: Posee el componente controladoras, en ésta es donde se encuentra la clase AccionController encargada de gestionar los procesos de inicio y diligencias del módulo.
- Negocio: Se encarga de gestionar la lógica de negocio, posee una amplia relación con el paquete Controlador. Ambos componentes se relacionan a su vez con el componente Entidad.
- Vista: Está compuesto por los componentes Twig para las interfaces con los usuarios, formularios para introducir los datos y extensiones como JavaScript y CSS.

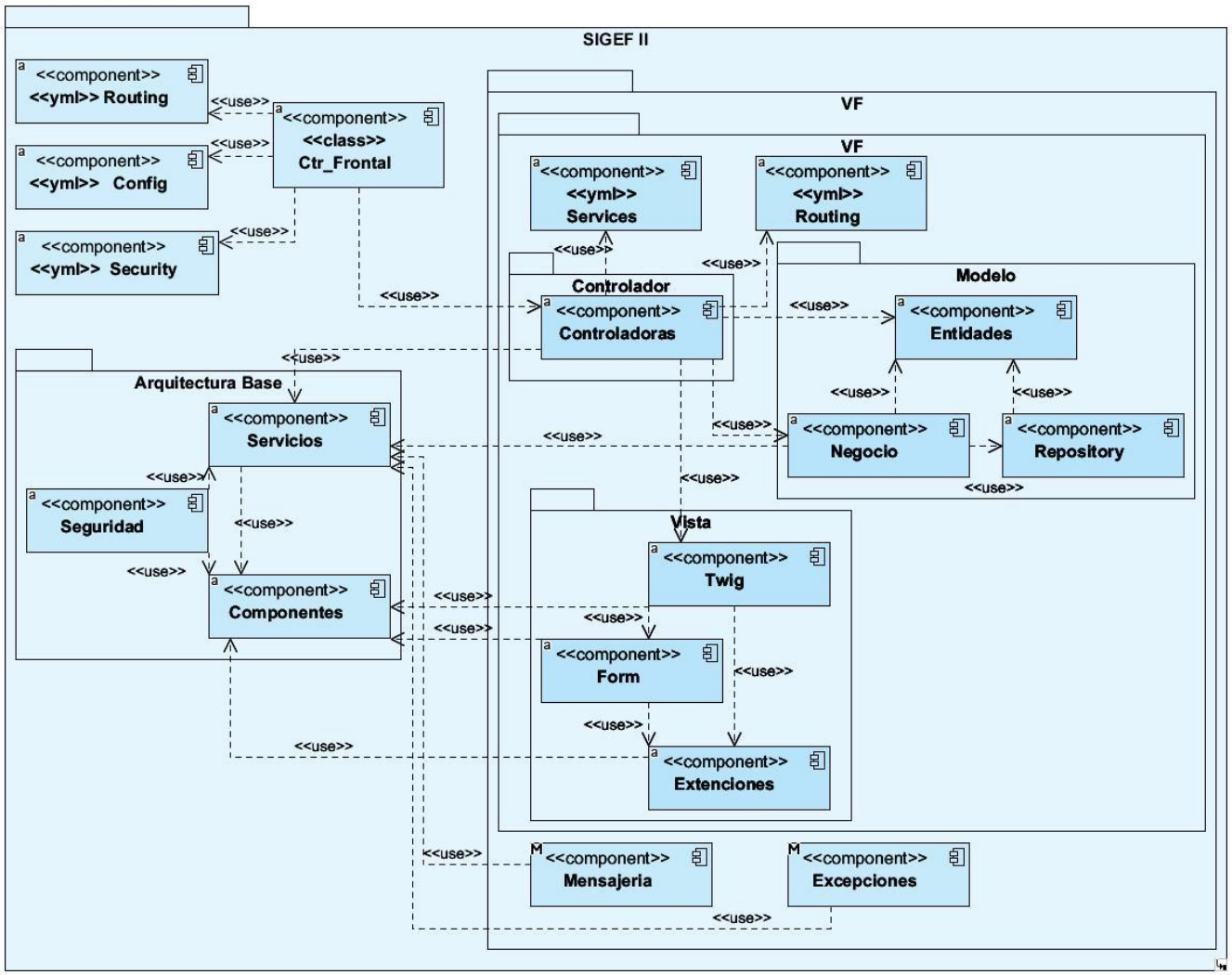


Figura 25: Diagrama de componentes

2.6.1.2. Modelo de despliegue

El modelo de despliegue está compuesto por elementos que muestran la configuración de los nodos de procesamiento en tiempo de ejecución y la comunicación entre ellos. Los elementos que lo componen son: (28)

- Procesador: nodo con elementos de procesamiento, al menos un procesador, memoria, y otros dispositivos.
- Dispositivos: nodos estereotipados sin capacidad de procesamiento en el nivel de abstracción que se modela.
- Conectores: expresa el tipo de conector o protocolo utilizado entre el resto de los elementos del modelo.

El modelo de despliegue describe la distribución física del sistema. (14) Muestra la estructura y relación que debe tener el hardware utilizado en la implementación del sistema con sus componentes. En la Figura 26 se muestra la distribución en que se encuentra la FGR.

En todas las instancias de las fiscalías se utilizará un servidor web, un servidor de base de datos, computadoras con clientes ligeros, impresoras y escáner. La conexión entre los servidores de base de datos y los servidores web, será mediante el protocolo TCP-IP, mientras que la conexión entre el servidor web y las computadoras clientes serán por el protocolo HTTPS. En el caso de la FGR existirá además un servidor de copias de seguridad para cuando exista algún fallo.

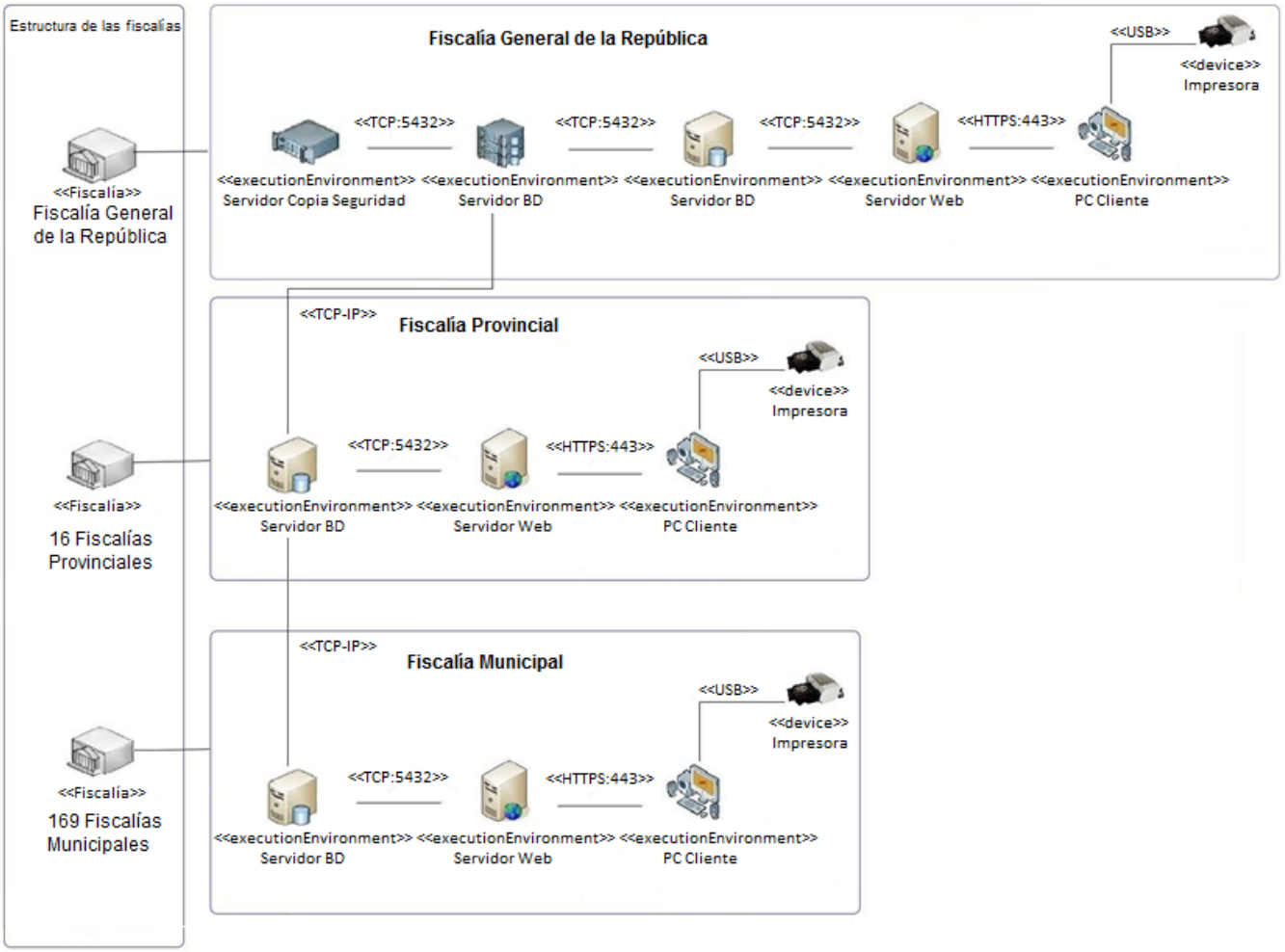


Figura 26: Modelo de despliegue

2.7. Estándares de codificación

Los estándares de codificación son creados con el objetivo de que el sistema siga un estilo de programación homogéneo, para permitir que todos los desarrolladores lo puedan entender en menos tiempo, que el código tenga una alta calidad, menos errores y pueda ser mantenido fácilmente. A continuación se muestran algunos ejemplos de estos estándares de codificación empleados en la implementación de los procesos de inicio y diligencias del módulo VF.

Cabecera del archivo

Cada archivo especificará el paquete que lo contiene y el uso de otras clases.

```
namespace VF\VFBundle\Controller;

use VF\VFBundle\Form\Accion\DespachoType;
use VF\VFBundle\Form\Accion\DespachoPersonaType;
```

Figura 27: Cabecera de inicio del archivo .php

Nombres de variables

Los nombres deben ser descriptivos y concisos. No se deben usar ni grandes frases ni pequeñas abreviaciones para las variables. Siempre es mejor saber qué hace una variable con sólo conocer su nombre.

```
$peticion = $this->getRequest();
$proceso = $this->getAccionGtr()->obtenerProceso($id_proceso);
```

Figura 28: Ejemplo de nombres de variables

Nombres de las funciones

Las funciones deben comenzar con minúscula y luego continuar con mayúscula.

```
public function listarDespachoAction($paso, $id_proceso) {
```

Figura 29: Ejemplo de nombre de funciones

Comentarios

Los comentarios pueden ser incluidos delante de cada función en forma de bloque (/* */).

```
    /* Metodo para listar los despachos en el grid*/
public function listarDespachoAction($paso, $id_proceso) {
    $paso_anterior = $this->listaPasosAnteriores($paso);
    $columns = array('getIdDocumento', 'getIdDocumento', 'getFecha');
    return new Response($this->Mostrar($columns, 'AccionGtr', 'listarDespacho',
                    false, true, array($paso, $id_proceso, $this->getUser())));
}
```

Figura 30: Ejemplo de comentario

Llaves

Siempre utilizar llaves para ganar en legibilidad.

```
if (!is_null($resultado)) {
    return $resultado;
}
```

Figura 31: Ejemplo del correcto uso de llaves

2.8. Conclusiones parciales

Luego de abordadas las temáticas del presente capítulo se concluye que:

- El uso de patrones de diseño, facilitó la solución de problemas con contextos similares y posibilitó la reutilización de código fuente.
- El uso del patrón arquitectónico permitió definir las relaciones entre los diferentes elementos del sistema así como los componentes de este. Sentando las bases para la correcta implementación del mismo.
- Todos los artefactos generados durante el diseño fueron validados mediante métricas, lo cual posibilitó asegurar la calidad del diseño antes de comenzar la fase de implementación.
- El diagrama de componentes posibilitó tener una vista de cómo están estructurados estos según la arquitectura seleccionada.
- El modelo de despliegue permite identificar la distribución de los nodos necesarios para el despliegue del sistema a nivel nacional. Las tecnologías representadas favorecen el despliegue de la solución.
- Los estándares de codificación garantizaron un código fuente legible y con la calidad necesaria.

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

El presente capítulo está orientado a la validación y verificación del sistema. Se verifica el correcto funcionamiento de los componentes implementados mediante los tipos de pruebas realizadas al sistema. Finaliza el capítulo comprobando que el resultado final cumple con las necesidades de la FGR, poniendo en práctica las mejoras en gestión de la información en cuanto a celeridad y control en la ejecución de sus procesos. Sommerville (33) plantea que “la verificación implica comprobar que el software satisface los requerimientos funcionales y no funcionales, mientras que la validación asegura que el sistema satisface las expectativas del cliente”.

3.1. Verificación del sistema

Para la verificación del sistema se realizaron las pruebas de software, con el objetivo de comprobar el correcto funcionamiento del sistema, a nivel de interfaz y de codificación mediante las pruebas de caja blanca y de caja negra respectivamente.

3.2. Pruebas de software

Uno de los factores de éxito más relevantes en el desarrollo de los proyectos de software es que éste sea capaz de satisfacer las necesidades y expectativas de sus usuarios. Lograr un producto final con el nivel de calidad requerido depende en gran medida de tener organizado de forma integral el proceso de pruebas a lo largo del ciclo de vida del proyecto. (34) En este epígrafe se recogen las validaciones del sistema, realizadas mediante las pruebas de caja negra y caja blanca.

3.2.1. Pruebas de caja negra

Las pruebas de caja negra se aplican sobre el sistema sin necesidad de conocer cómo está construido por dentro, se realizan a través de los requisitos funcionales. Estas pruebas son más efectivas si se realizan paralelas con la implementación. Se basan en corregir las siguientes categorías; funciones incorrectas o ausentes, errores de interfaz, errores en la estructura de datos o en accesos a bases de datos externas, errores de rendimiento, errores de inicialización y de terminación. (35)

La técnica que se aplica en esta investigación se denomina Partición de Equivalencia. Esta técnica evalúa las clases de equivalencia, las cuales son representadas por un conjunto de estados válidos, inválidos o que no aplican, para condiciones de entradas. Las condiciones de entrada son valores numéricos específicos, un rango de valores, un conjunto de valores relacionados o una condición lógica.

A continuación se muestra un análisis de las pruebas realizadas al requisito Adicionar Despacho. La ejecución de la técnica se encuentra en el Anexo 5 de este documento. Los restantes casos de prueba se encuentran en el documento Diseño de Caso de Prueba basado en Requisitos de los procesos de inicio y diligencias del módulo VF.

Como muestra la Figura 32 en una primera iteración para un total de 18 pruebas realizadas, se detectaron 6 no conformidades, pues el sistema no ofrecía la respuesta que se esperaba según la descripción de los casos de prueba. Estos errores fueron corregidos inmediatamente y luego se aplicó una segunda iteración.

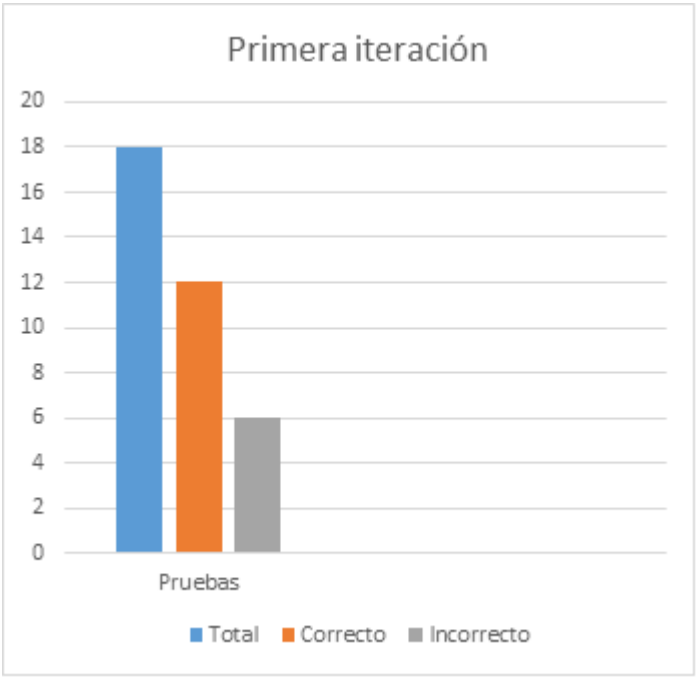


Figura 32: Primera iteración del caso de prueba

La Figura 33 muestra la segunda iteración con las no conformidades anteriores corregidas. Para la misma cantidad de pruebas en esta iteración se detectaron 2 no conformidades las cuales se corrigieron y se continuó con la comprobación del software.

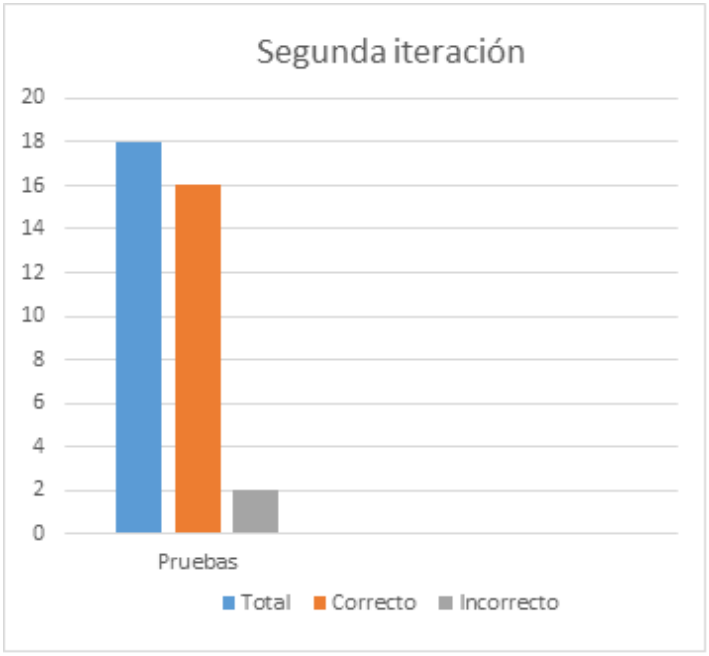


Figura 33: Segunda iteración del caso de prueba

Para una tercera iteración, el sistema devolvió la respuesta esperada para cada descripción del caso de prueba. Se eliminaron todas las no conformidades identificadas en iteraciones previas como se muestra en la Figura 34.

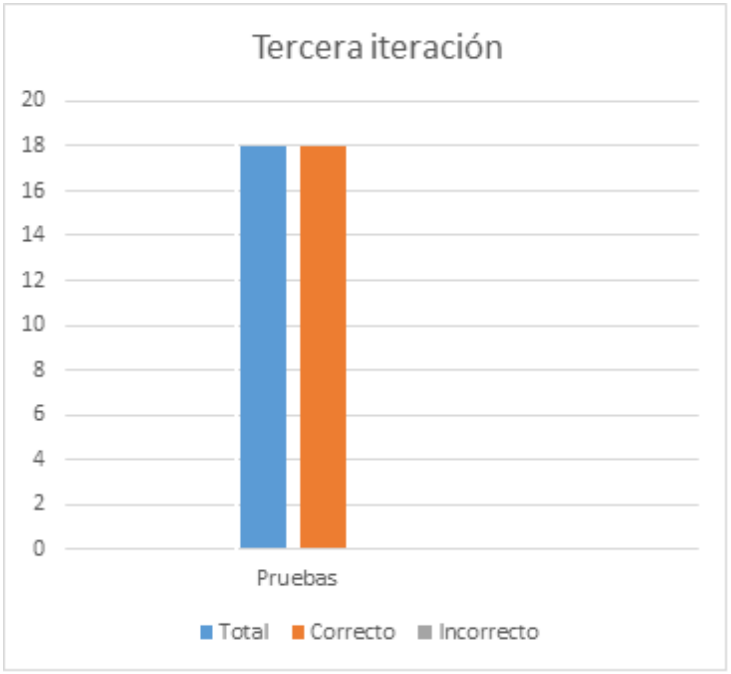


Figura 34: Tercera iteración del caso de prueba

3.2.2. Pruebas de caja blanca

Las pruebas de caja blanca necesitan conocer el código para seguir su estructura lógica y así diseñar pruebas destinadas a comprobar que el código hace correctamente lo que el diseño de bajo nivel indica. (35) Se utiliza como marco de trabajo base para la implementación y realización de las pruebas de caja blanca PHPUnit, el cual se logra integrar perfectamente con el IDE de desarrollo seleccionado.

Para aplicar dicha prueba se empleó la técnica del Camino Básico. Esta técnica comprueba todos los caminos lógicos del software a través de casos de prueba, de forma que se pueda asegurar que el estado real coincide con el esperado.

3.2.2.1. Técnica del camino básico

Al aplicar la técnica del camino básico se construye la gráfica de flujo asociado, se calcula la complejidad ciclomática y se obtienen un conjunto de caminos independientes. Para aplicar esta técnica se tuvo en cuenta las definiciones siguientes:

La gráfica de flujo describe un flujo de control lógico, donde cada círculo llamado nodo representa una o más instrucciones procedimentales. Las flechas llamadas aristas o enlaces representan el flujo de control y son análogos a las flechas de los diagramas de flujo. (28)

Se denomina **regiones** a las áreas delimitadas por aristas y nodos.

La complejidad ciclomática se basa en la teoría gráfica y da un límite superior para el número de pruebas que se deben realizar, asegura que se ejecuta cada sentencia al menos una vez y se puede calcular de las siguientes maneras: (28)

1: El número de regiones corresponde a la complejidad ciclomática.

2: La complejidad ciclomática V (G), de una gráfica de flujo G, se define como V (G) = E-N+2 donde E es el número de arista y N la cantidad de nodos de la gráfica.

3: La complejidad ciclomática V (G), de una gráfica de flujo G, también se define como V (G)=P+1 donde P es el número de nodos predicados incluidos en la gráfica de flujos.

Un camino independiente es aquel donde se pueda recorrer la gráfica sin repetir ninguna arista. (28)

A continuación se muestra el código perteneciente al método salvarActualizarDespachoAction de la funcionalidad "Gestionar Despacho" al que se le realizará la prueba del camino básico.

```
public function salvarActualizarDespachoAction($paso, $id_proceso, $id_documento) {
    $flag_gestionar = false;
    $peticion = $this->getRequest();
1   $proceso = $this->getAccionGtr()->obtenerProceso($id_proceso);
    $despacho = $this->getAccionGtr()->crearDespacho($id_documento, $proceso);
    $frm = $this->createForm(new DespachoType(), $despacho, array());
2   if ($peticion->getMethod() == 'POST') {
3       $frm->bind($peticion);
4       if ($frm->isValid()) {
            $usuario = $this->getUser();
5           $tipo_boton = $peticion->get(ConfigUtilVF::btn_finalizar);
6           if (isset($tipo_boton)) {
                $this->getAccionGtr()->salvarActualizarDespacho($proceso, $despacho, $usuario, ConfigUtilVF::estado_finalizado);
                $this->getAccionGtr()->cambiarEstadoProceso($proceso, ConfigUtilVF::estado_finalizado, $usuario);
7               $errores[] = 'me_adm:me_finalizar';
                $this->adicionarFlashError($errores);
                $flag_gestionar = true;
            } else {
8               $tipo_boton_g = $peticion->get(ConfigUtilVF::btn_guardar);
9               if (isset($tipo_boton_g)) {
                    $this->getAccionGtr()->salvarActualizarDespacho($proceso, $despacho, $usuario, ConfigUtilVF::estado_pendiente)
10                  $flag_gestionar = true;
                }
                $this->getAccionGtr()->salvarActualizarDespacho($proceso, $despacho, $usuario, ConfigUtilVF::estado_pendiente);
11              $flag_gestionar = true;
            }
        }
    }
12  if ($flag_gestionar) {
13      return $this->redirect($this->generateUrl('vf_gestionar_despacho_1', array('paso' => $paso, 'id_proceso' => $id_proceso)))
    } else {
14      return $this->render('VFBundle:Accion:adicionar_despacho_form.html.twig', array(
                'frm' => $frm->createView(),
                'paso' => $paso,
```

Figura 35: Método salvarActualizarDespachoAction de la funcionalidad Gestionar Despacho

Gráfica de flujo a partir del código mostrado

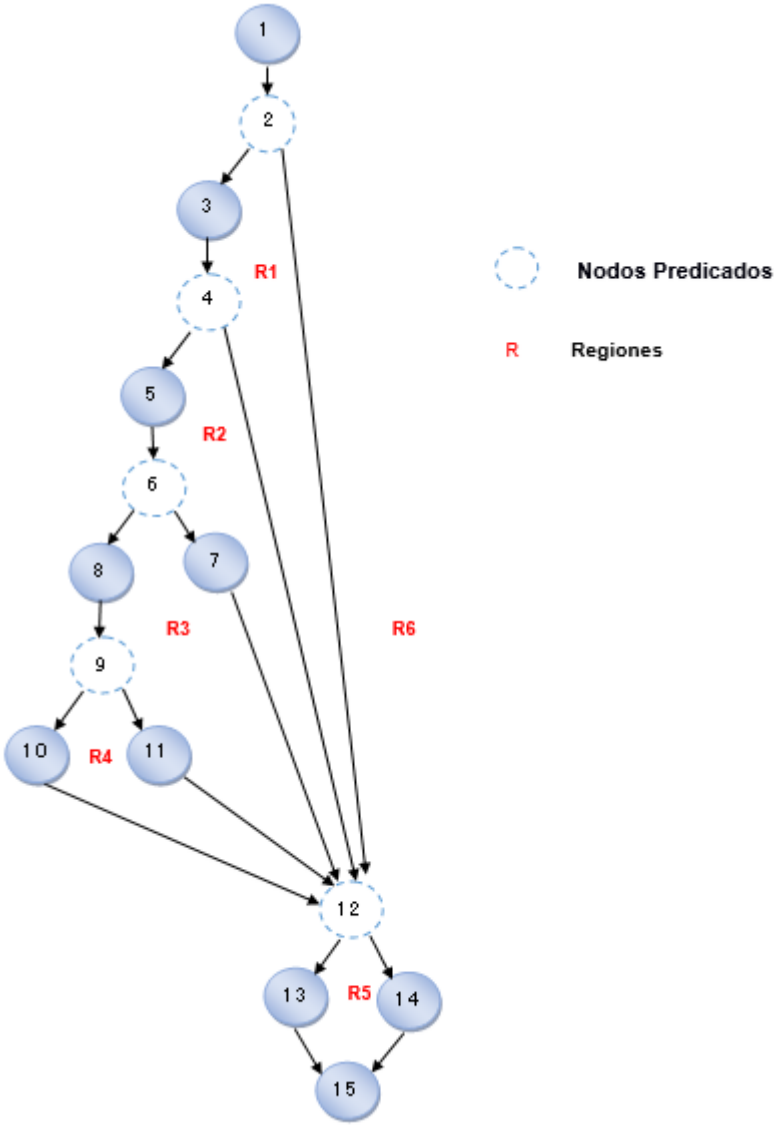


Figura 36: Gráfica de flujo del método salvarActualizarDespachoAction

Luego se calcula la complejidad ciclomática.

1: V (G)=6

2: V (G) = A–N+2. 19–15+2 = 6

3: V (G) = P+1. 5+1 = 6

El valor de V (G) fue igual al número de caminos independientes, por lo que se definieron los siguientes 6 caminos.

Caminos independientes:

Camino 1: 1-2-12-14-15

Camino 2: 1-2-3-4-12-14-15

Camino 3: 1-2-3-4-5-6-7-12-13-15

Camino 4: 1-2-3-4-5-6-8-9-10-12-13-15

Camino 5: 1-2-3-4-5-6-7-8-9-11-12-13-15

Camino 6: 1-2-12-13-15

Una vez identificada la gráfica de flujo y los caminos a recorrer, se realizaron los casos de pruebas por cada camino, con el objetivo de verificar que las condiciones de los nodos predicados están establecidas correctamente. A continuación se muestran los casos de prueba para cada uno de los caminos seleccionados anteriormente.

Caso de prueba para el camino 1

Entrada	Se obtiene el proceso, se crea el despacho y se crea el formulario. La petición debe ser de tipo GET.
Resultados esperados	Teniendo en cuenta la condición de ejecución se espera que devuelva la página html adicionar_despacho_form
Condiciones	$peticion = GET, $flag_gestionar = false

Tabla 9: Caso de prueba para el camino 1

Caso de prueba para el camino 2

Entrada	Se obtiene el proceso, se crea el despacho y se crea el formulario. La petición debe ser de tipo POST, el formulario no es válido.
Resultados esperados	Teniendo en cuenta la condición de ejecución se espera que devuelva la página html adicionar_despacho_form
Condiciones	$peticion = POST, $frm->isValid()=false, $flag_gestionar = false

Tabla 10: Caso de prueba para el camino 2

Caso de prueba para el camino 3

Entrada	Se obtiene el proceso, se crea el despacho y se crea el formulario. La petición debe ser de tipo POST, el formulario es válido, el tipo de botón es finalizar.
Resultados esperados	Teniendo en cuenta la condición de ejecución se guardan los datos en la base de datos, se cambia el estado del proceso y se redirecciona a la ruta gestionar_despacho_1.

Condiciones	$peticion = POST, $frm->isValid()=true, isset(tipo_boton_finalizar), $flag_gestionar = true

Tabla 11: Caso de prueba para el camino 3

Caso de prueba para el camino 4

Entrada	Se obtiene el proceso, se crea el despacho y se crea el formulario. La petición debe ser de tipo POST, el formulario es válido, se selecciona el botón guardar.
Resultados esperados	Teniendo en cuenta la condición de ejecución se guardan los datos en la base de datos y se redirecciona a la ruta gestionar_despacho_1.
Condiciones	$peticion = POST, $frm->isValid()=true, isset(tipo_boton_g), $flag_gestionar = true

Tabla 12: Caso de prueba para el camino 4

Caso de prueba para el camino 5

Entrada	Se obtiene el proceso, se crea el despacho y se crea el formulario. La petición debe ser de tipo POST, el formulario debe ser válido, se selecciona el botón finalizar.
Resultados esperados	Teniendo en cuenta la condición de ejecución se guardan los datos en la base de datos, se cambia el estado del proceso y se redirecciona a la ruta gestionar_despacho_1.
Condiciones	$peticion = POST, $frm->isValid()=true, isset($tipo_boton_finalizar), $flag_gestionar = true

Tabla 13: Caso de prueba para el camino 5

Caso de prueba para el camino 6

Entrada	Se obtiene el proceso, se crea el despacho y se crea el formulario. La petición debe ser de tipo GET.
Resultados esperados	Teniendo en cuenta la condición de ejecución se espera que devuelva la ruta gestionar_despacho_1.
Condiciones	$peticion = GET, $flag_gestionar = false

Tabla 14: Caso de prueba para el camino 6

En una primera iteración de estas pruebas se encontraron 3 errores los cuales fueron corregidos en una segunda iteración asegurando el correcto funcionamiento, llegando a obtener resultados satisfactorios. Se pudo comprobar que las condiciones que pertenecen al método salvarActualizarDespachoAction están establecidas correctamente.

Para comprobar el funcionamiento de los casos de prueba se empleó el marco de trabajo PHPUnit en la realización de las pruebas de caja blanca. La Figura 37 muestra que tras la primera iteración al aplicar esta prueba se encontró errores en el código, mostrados en color gris, el color verde significa que las líneas fueron ejecutadas de forma correcta, mientras que las líneas de color naranja no fueron ejecutadas. Los errores fueron solucionados posteriormente. En una segunda iteración esta prueba arrojó resultados satisfactorios, quedando eliminados los errores anteriores y devolviendo la respuesta esperada.

```
public function salvarActualizarDespachoAction($paso, $id_proceso)
    $peticion = $this->getRequest();
    $proceso = $this->getAccionGtr()->obtenerProceso($id_proceso);
    $resultado = $this->verificarProcesoEnEdicion($proceso, $this-> listaPasosAnter
        $this->generateUrl('vf_gestionar_despacho_1', array('paso' => ConfigUtilVF:
    if (!is_null($resultado)) {
      return $resultado;
    }
    $despacho= $this->getAccionGtr()->crearDespacho($id_documento, $proceso);
    $accion=$despacho->getProceso();
    $frm = $this->createForm(new DespachoType(), $despacho, array());
    if ($peticion->getMethod() == 'POST') {
        $frm->bind($peticion);
        if ($frm->isValid()) {
            $usuario = $this->getUser();
            $tipo_boton = $peticion->get(ConfigUtilVF::btn_finalizar);
            if (isset($tipo_boton)) {
                $this->getAccionGtr()->salvarActualizarDespacho($proceso, $despacho,
                $this->getAccionGtr()->cambiarEstadoProceso($proceso,ConfigUtilVF::
                $errores[] = 'me_adm:me_finalizar';
                $this->adicionarFlashError($errores);
                return $this->redirect($this->generateUrl('vf_gestionar_despacho_1',
                                array('paso' => ConfigUtilVF::Despacho,'id_proceso'=>$
            } else {
            $tipo_boton_g = $peticion->get(ConfigUtilVF::btn_guardar);
            if (isset($tipo_boton_g)) {
                $this->getAccionGtr()->salvarActualizarDespacho($proceso, $despacho,
                            $usuario, ConfigUtilVF::estado_pendiente);
                return $this->redirect($this->generateUrl('vf_gestionar_despacho_1',
                    array('paso' => $paso, 'id_proceso' => $id_proceso)));
            }  }
        $this->getAccionGtr()->salvarActualizarDespacho($proceso, $despacho, $usuario
        return $this->redirect($this->generateUrl('vf_gestionar_despacho_1',
                    array('paso' => $paso, )));
```

Figura 37: Resultado de la prueba con PHPUnit

3.3. Validación de la propuesta de solución

Una vez terminado el software se evalúan todas las funcionalidades comparándolas con la perspectiva de los futuros usuarios de la aplicación. Para dar respuesta al problema planteado fue necesario medir las variables planteadas en la problemática. Un encuentro realizado entre los compañeros de la fiscalía y los compañeros del proyecto SIGEFII arrojó el siguiente resultado acerca de los tiempos de duración en la realización de los proceso de inicio y diligencias, comparando indicadores para medir las variables en tiempo real y mediante el sistema.

Variables

Gestión de la información: se mide a través de la disponibilidad e integridad de la información que se recoge en el departamento de VF. Rapidez en el envío de documentos entre las diferentes instancias de la fiscalía. Además de la búsqueda de documentos.

Control: se mide con el cumplimiento de los términos a través de alarmas; con la verificación de los documentos creados en un período; con la visualización de los documentos insertados a diario, a través de los detalles y con la persistencia de la información. También garantizando el correcto acceso a la información por niveles en las fiscalías.

Celeridad: disminución del tiempo de respuesta en la creación de los documentos y el acceso a ellos. Disminuyendo el tiempo empleado en la búsqueda de un documento se mejora el acceso a la información. El tiempo de creación y de traslado de la documentación mejora la gestión de los documentos generados.

Resultados

- El tiempo empleado en realizar un proceso de forma manual es considerablemente alto si se compara con el tiempo que demora realizarlo en el sistema.
- El tiempo en que demora el envío de los documentos entre las diferentes instancias utilizando el sistema, es considerablemente menor, además se elimina el envío vía correo electrónico.
- El tiempo en que demora obtener los resultados de la búsqueda de un documento o un proceso es considerablemente alto comparado con lo que demora realizar la búsqueda en el sistema.
- El tiempo de redacción y revisión de un documento es mayor comparado con el tiempo utilizado por el sistema.

Las tablas que se muestran a continuación reflejan el análisis realizado anteriormente y la comparación de los distintos tiempos obtenidos.

Dimensión de la variable	Antes	Después
Acceso a la información	No se conocen los detalles del proceso si no se tiene la Resolución de la acción.	La información de la acción puede ser consultada desde cualquier instancia si se tienen los permisos necesarios.

Tabla 15: Validación para la variable control

Dimensión de la variable	Antes	Después	Indicadores
Acceso a la información	Aproximadamente 1 hora.	Aproximadamente 10 minutos.	Tiempo empleado en la búsqueda de un documento.
Documentos generados	Aproximadamente 2 horas.	Aproximadamente 5 minutos.	Tiempo de creación de un documento. Tiempo de traslado

Tabla 16: Validación para la variable celeridad

Dimensión de la variable	Antes	Después	Indicadores
Acceso a la información	Aproximadamente 1 hora.	Aproximadamente 10 minutos.	Tiempo para realizar búsquedas. Persistencia de la información.
Documentos generados	Aproximadamente 2 horas.	Aproximadamente 5 minutos.	Tiempo de creación y tramitación. Tiempo de traslado de la documentación

Tabla 17: Validación para la variable gestión de la información

CONCLUSIONES GENERALES

Mediante el desarrollo de este trabajo se llega a las siguientes conclusiones:

- Ninguno de los sistemas analizados satisfacen completamente las necesidades de la FGR, por lo que demostró la necesidad de la implementación de los procesos inicio y diligencias del módulo VF del proyecto SIGEF II.
- Todas las tecnologías, herramientas y lenguajes seleccionados propiciaron la calidad del resultado final, tomando en consideración la política de migración hacia software libre que lleva a cabo el país.
- Todos los artefactos generados sirvieron de guía en el desarrollo de la solución, logrando un mejor entendimiento de los flujos de trabajo y facilitando el trabajo de los desarrolladores en la fase de implementación.
- Las técnicas de validación del diseño y patrones empleados en las diferentes fases en la construcción del software permitieron llegar a resultados fiables, los cuales se comprobaron en todos los artefactos obtenidos a lo largo del desarrollo.
- Todos los estándares de codificación utilizados permitieron la aplicación de buenas prácticas en el desarrollo del sistema, logrando así el propósito de lograr un código legible y organizado para la aplicación.
- Se validó el resultado obtenido a través de las pruebas de caja negra y caja blanca, demostrando que las funcionalidades del sistema son correctas, arrojando finalmente los resultados esperados.
- Con el resultado obtenido se mejora la gestión de la información del módulo VF garantizando el control y la celeridad en la ejecución de los procesos de inicio y diligencias.

RECOMENDACIONES

Se recomienda:

- Realizar un estudio con mayor profundidad que permita incorporar nuevas funcionalidades al software.
- Implementar las fases restantes del módulo VF.
- Realizar el despliegue de los procesos implementados en las fiscalías de manera que mejore su gestión.

BIBLIOGRAFÍA

1. Giron, Sandra Milena Moran. *Incidencia de las TIC, Innovacion y Capital Humano.* CSantiago de Cali : s.n., 2010.

2. República, Fiscalía General de la. *Costitución de la República.*

3. *Fiscalía General de la República. Instrucciones de Verificaciones Fiscales.* Habana : s.n., 1999. s.n.

4. República, Fiscalía General de la. *Explicación del negocio de las interfaces de Ejecución.* 2013.

5. Contraloría General de la República. [En línea] 2013. [Citado el: 4 de Diciembre de 2013.] http://www.contraloriagen.gov.co.

6. La Moncloa. [En línea] 2010. [Citado el: 12 de Noviembre de 2013.] http://www.lamoncloa.gob.es. Mo.

7. Ministerio Público Fiscal. [En línea] 2013. [Citado el: 11 de Diciembre de 2013.] http://www.fiscalias.gob.ar.

8. *InfoFiscalia.* Mulet, Marisa del Valle. Madrid : s.n., 2010.

9. Ivar Jacobson, Grady Booch, James Rumbaugh. *El Proceso Unificado de Desarrollo de Software.* Madrid : s.n., 2000.

10. Ivar Jacobson, Grady Booch and Rumbaugh. *El proceso unificado de desarrollo de software.* s.l. : Adison Wesley, 2010. ISBN 84-7829-036-2.

11. Kruchten. *The Relational Unifeid Process: An Introduction.* s.l. : Adison Wesley, 2000.

12. Comunidad Postgres. [En línea] [Citado el: 12 de Diciembre de 2013.] http://postgresql.uci.cu.

13. Martinez, Rafael. PostgreSQL. [En línea] 2012. [Citado el: 12 de Diciembre de 2013.] http://www.postgresql.org.es.

14. Yenier Figueroa, Héctor Fuentes, Manuel Delgado, Yanisleidy Torres. *Propuesta de arquitectura de software para proyectos de gestión sobre plataformas web.* La Habana : s.n., 2012.

15. PgAdmin III. [En línea] 10 de Marzo de 2012. [Citado el: 15 de Mayo de 2014.] http://www.guia-ubuntu.com/index.php?title=PgAdmin_III.

16. Herramientas CASE. [En línea] 2013. [Citado el: 13 de Diciembre de 2013.] http://www.slideshare.net.

17. Introduccion al UML. [En línea] 2013. [Citado el: 13 de Diciembre de 2013.] http://es.scribd.com.

18. Rumbaugh, James, Ivar. *El Lenguaje Modificado de Modelado.* 2000.

19. NetBeans. [En línea] [Citado el: 10 de Diciembre de 2013.] http://netbeans.org.

20. Serradilla, Juan Luis. *Sección de metodología, Normalización y Calidad del Software.* 2007.

21. Fabien Potencier, François Zaninotto. *Symfony la guía definitiva.*

22. *Requisitos Funcionales del Módulo Verificación Fiscal.*

23. Diccionario de Informática. [En línea] [Citado el: 6 de Mayo de 2014.] http://www.diccionarios.com/Dic/arquitecturadesistemas.php.

24. Pressman, Roger. *Ingeniería de Software. Un enfoque práctico. 5ta edición.* 2002. ISBN: 8448132149.

25. Crece Negocios. [En línea] [Citado el: 1 de Abril de 2014.] http://www.crecenegocios.com.

26. Larman, Craig. *UML y Patrones. Introducción al análisi y diseño orientado a objetos.* 1999. ISBN 970-17-0261-1.

27. Astudillo, Marcello Visconti y Hernán. *Fundamentos de Ingeniería de Software.* Mexico : Prentice Hall, 1999. ISBN 970-17-0261-1.

28. Pressman, Roger S. *Ingeniería de Software. Un enfoque práctico, 6ta Edición.*

29. Gutierrez, Demián. *UML Diagrama de Secuenia.* Venezuela : s.n., 2011.

30. Muñoz, Dra. Coral. *Métricas de la Calidad de Software.* 2008.

31. Gestión de Proyectos y Desarrollo de Software. [En línea] 28 de Mayo de 2010. [Citado el: 6 de Marzo de 2014.] https://jummp.wordpress.com.

32. Soldado, Rosana Montes. *Métricas Aplicables al Diseño Orientado a Objetos.* Granada : s.n., 2011.

33. Sommerville, Ian. *Ingeniería del Software. Septima edición.* Madrid : Pearson educación, 2005. ISBN: 84-7829-074-5.

34. Pruebas de Software. [En línea] 2008. [Citado el: 9 de Abril de 2014.] http://pruebasdesoftware.com/laspruebasdesoftware.html.

35. Ingeniería de Software II. [En línea] 2012. [Citado el: 8 de Diciembre de 2013.] http://sities.google.com/ingenieriadesoftwre.

36. República, Fiscalía General de la. *Metodología para la ejecución de las Verificaciones Fiscales.*

37. Sierra, Francisco Gomez. Control Fiscal y Proceso de Responsabilidad Fiscal. [En línea] 2013. [Citado el: 5 de Diciembre de 2013.] http://edileyer.com. CF.

38. Sistema Integral de Comprobantes Fiscales. [En línea] 2010. [Citado el: 5 de Diciembre de 2013.] http://www.sat.gob.mx.

39. LibrosWeb. [En línea] 2013. [Citado el: 13 de Diciembre de 2013.] http://librosweb.es/symfony2.

40. Symfony. [En línea] 2013. [Citado el: 13 de Diciembre de 2013.] http://symfony.com.

41. PHP. [En línea] 2013. [Citado el: 13 de Enero de 2014.] http://php.net.

42. Desarrollo Web. [En línea] [Citado el: 8 de Diciembre de 2013.] http://www.desarrolloweb.com.

43. CSS3. [En línea] [Citado el: 8 de Diciembre de 2013.] http://www.css3.com.

44. JQuery. [En línea] [Citado el: 8 de Diciembre de 2013.] html://www.jquery.com.

45. Gómez, José Jorge Márquez. *Arquitectura MVC. Visión general.* 2009.

46. Usaloa, Macario Polo. *Patrones GRASP.* 2000.

47. doctrine. [En línea] [Citado el: 7 de Marzo de 2014.] http://www.doctrine-project.org/projects/orm.html.

48. Marco de Desarrollo de Junta de Andalucía. [En línea] 2013. [Citado el: 7 de Marzo de 2014.] http://www.juntadeandalucia.es.

49. Eguiluz, Javier. *Desarrollo Web Ágil con Symfony 2.* 2012.

50. SencioLabs. [En línea] 2012. [Citado el: 7 de Marzo de 2014.] http://twig.sensiolabs.org/.

51. *, Fiscalía General República de Cuba. Ley No 83, Artículo 8 ley de la Fiscalía General República. .* Habana, Cuba : s.n., 1997.

52. Yenier Figueroa Machado, Hector Fuentes Blanco. *Estándares de codificación para el diseño. Informe del proyecto SIGEF II.* La Habana : s.n., 2012.

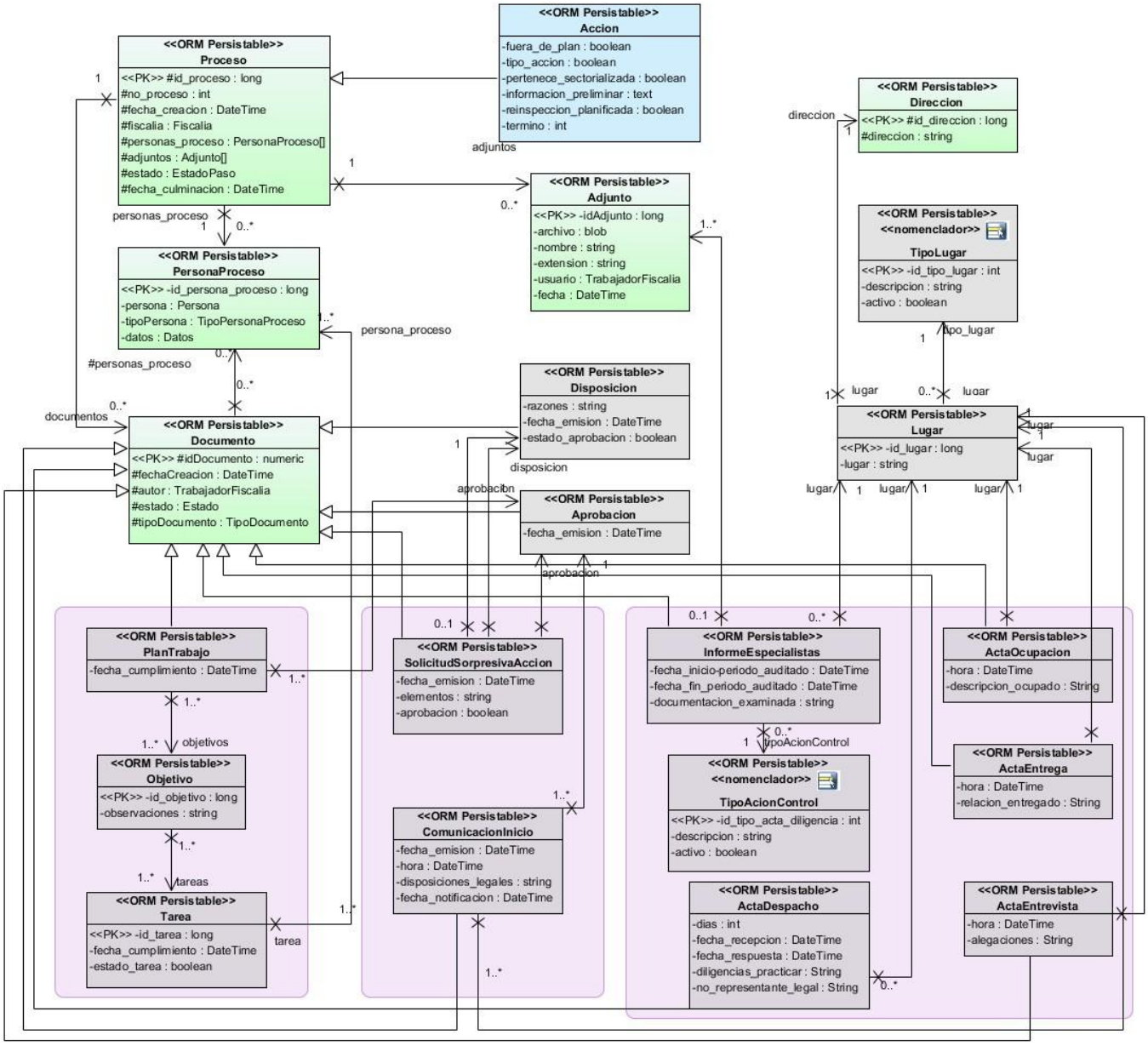
53. Molpeceres, Alberto. *Procesos de desarrollo: RUP, XP y FDD.* 2003.

54. Pacheco, Nacho. *The Twig Book.* 2013.

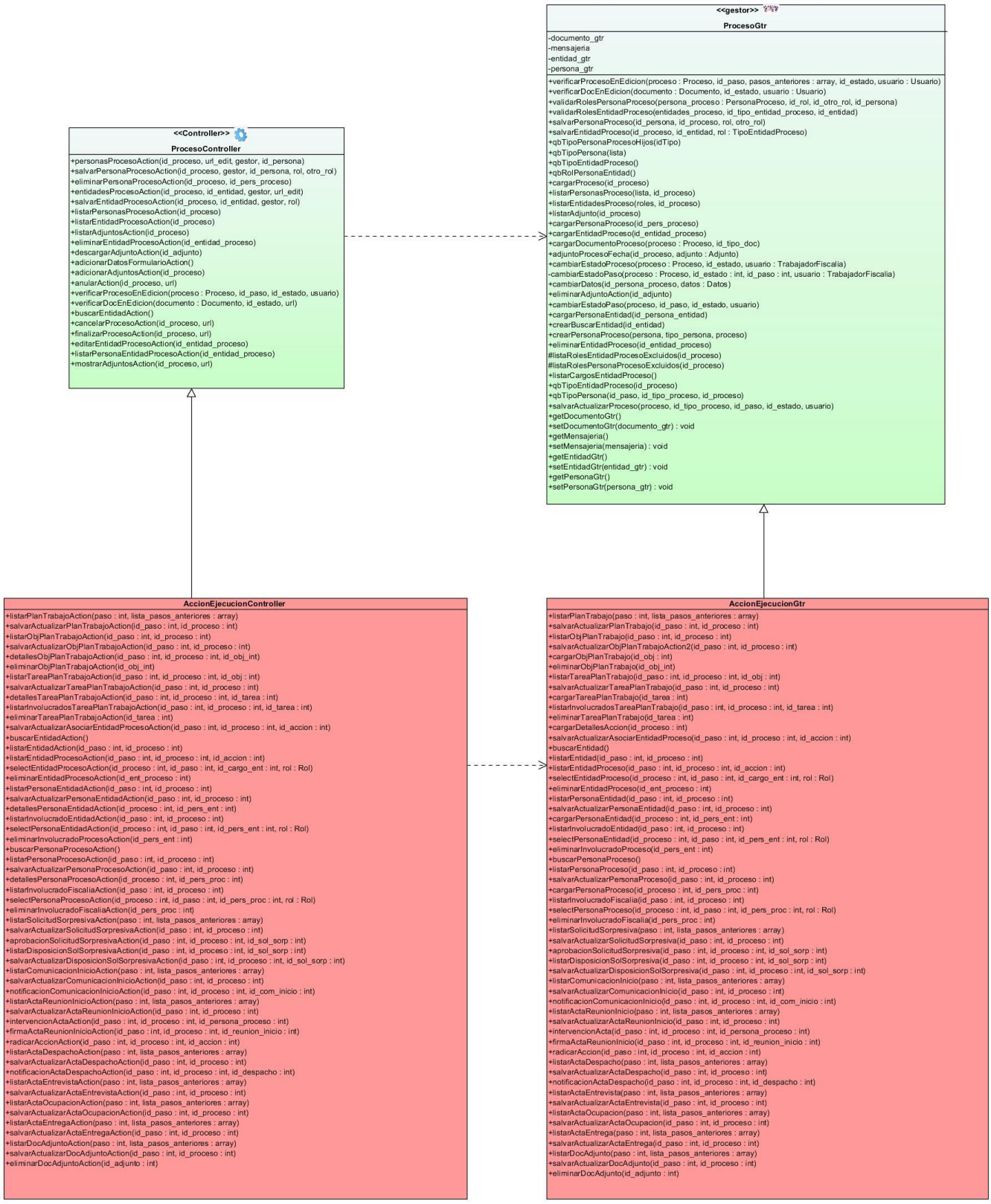
55. LIBROSWEB. [En línea] 2014. [Citado el: 15 de Mayo de 2014.] http://librosweb.es/javascript/capitulo_1.html.

ANEXOS

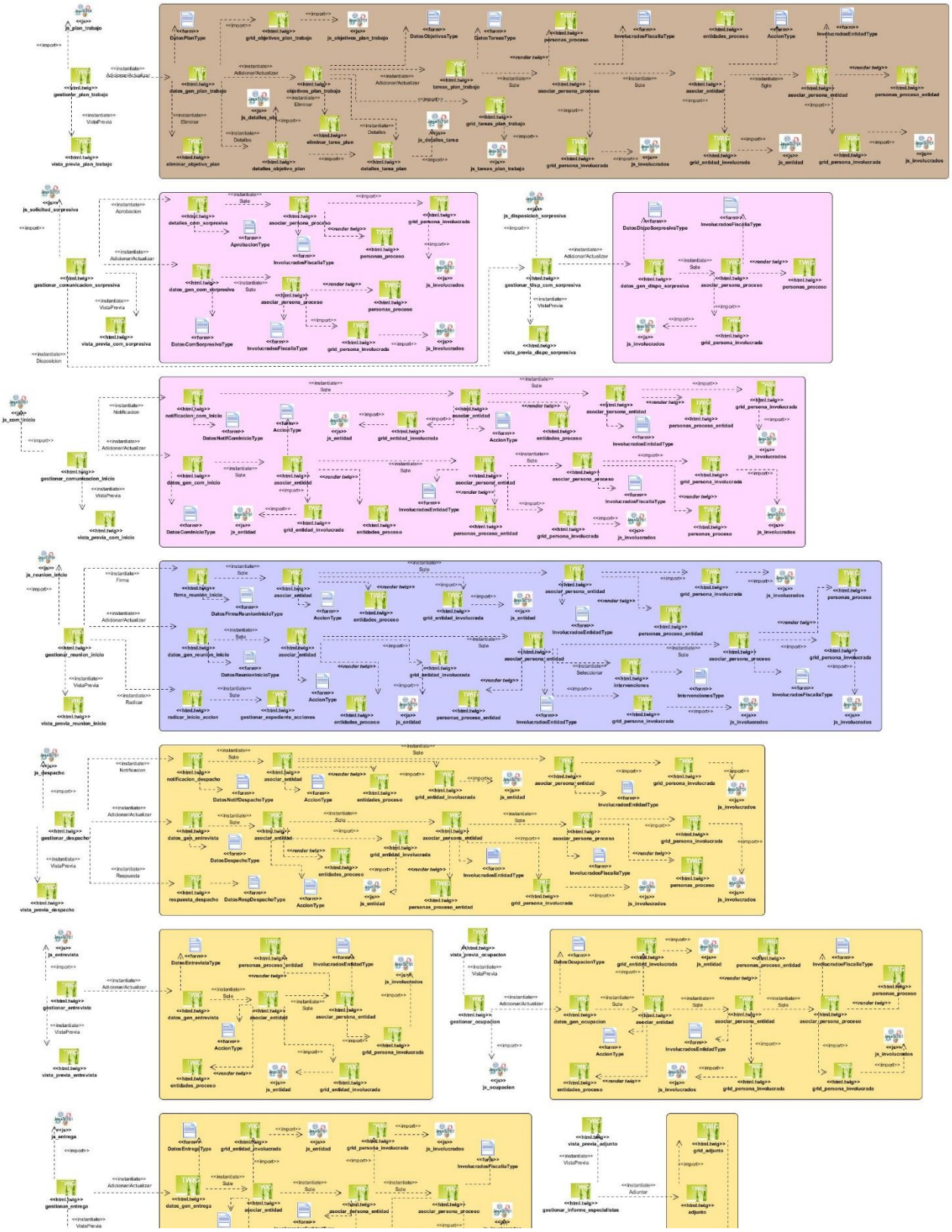
Anexo 1: Diagrama de clases de los procesos de inicio y diligencias del módulo VF



Anexo 2: Diagrama de Controladoras y Gestoras de los procesos de inicio y diligencias del módulo VF

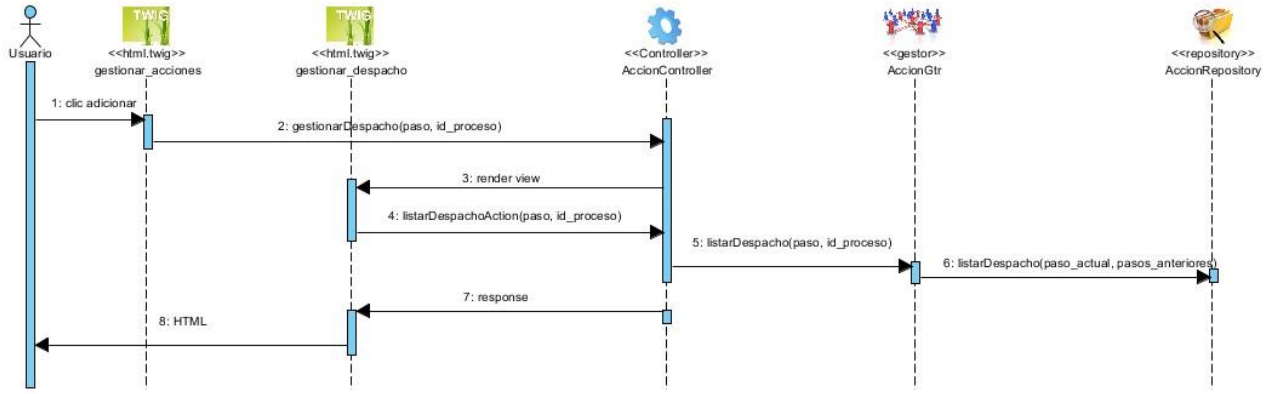


Anexo 3: Diagrama de vista de los procesos de inicio y diligencias del módulo VF

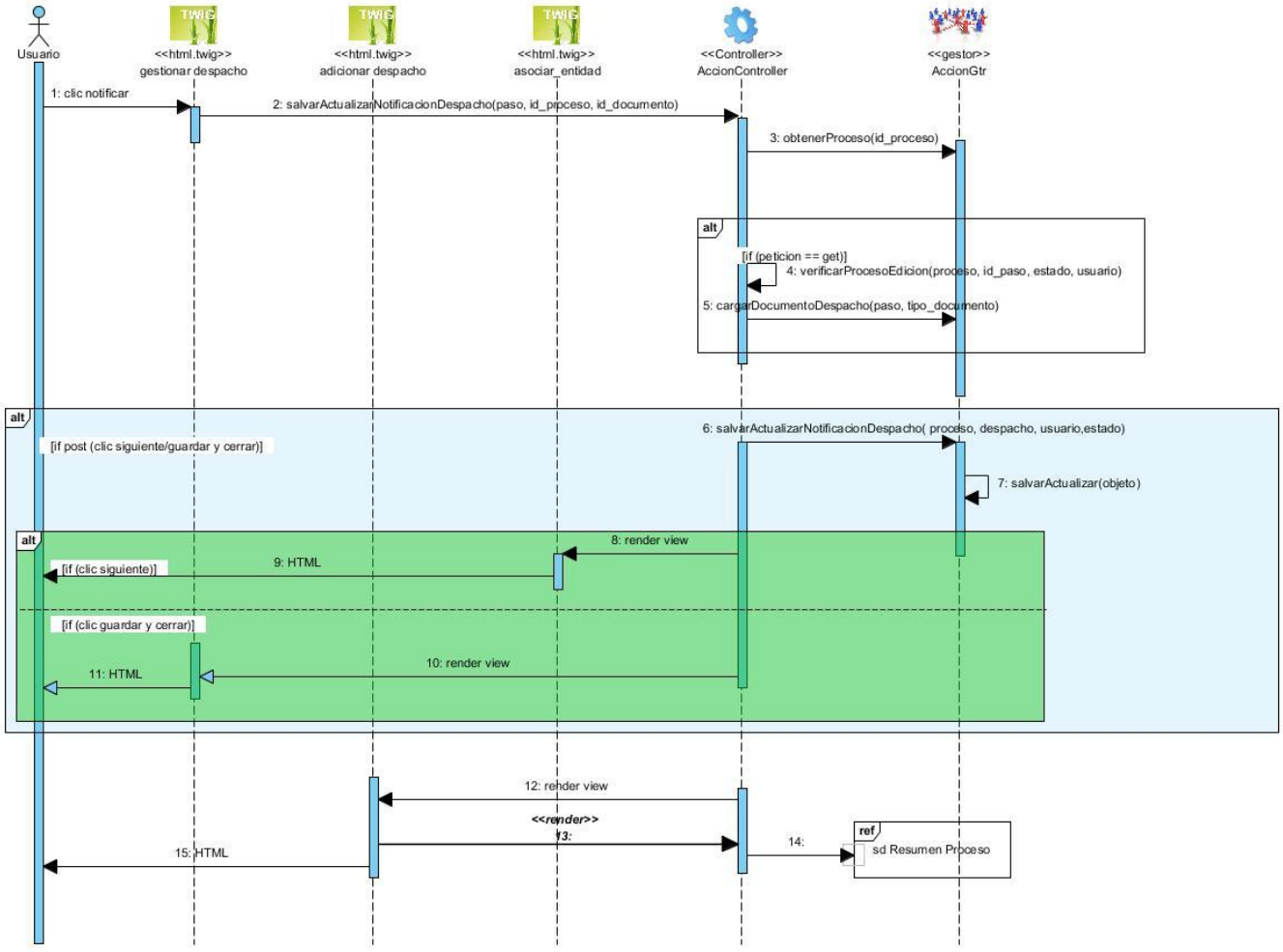


Anexo 4: Diagramas de secuencia de la funcionalidad Gestionar Despacho

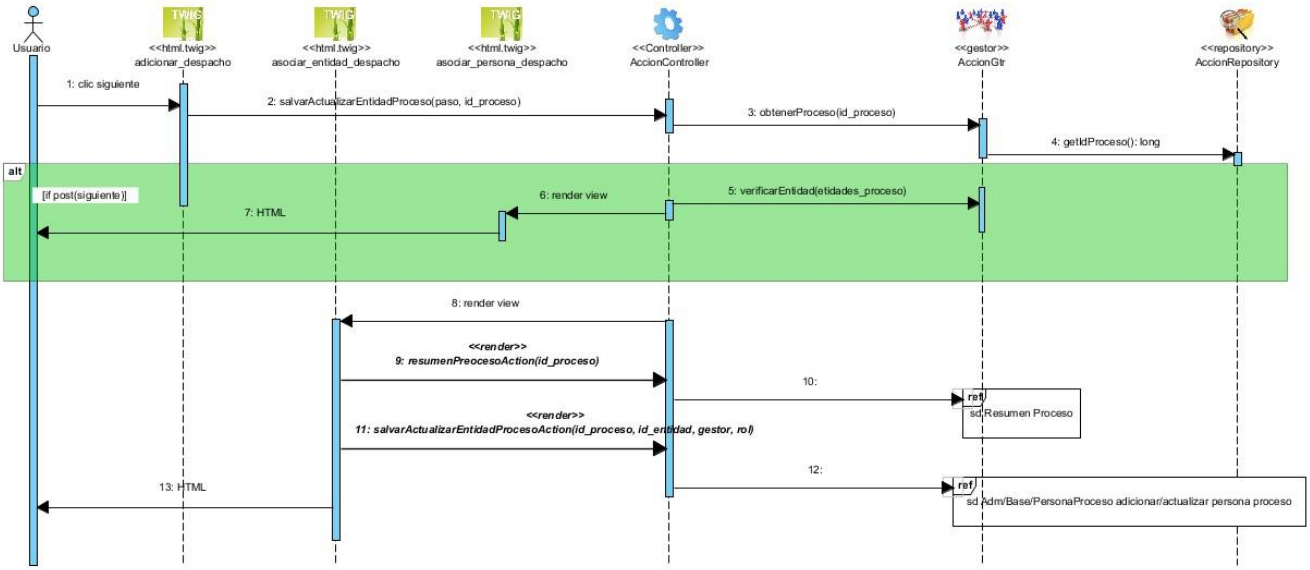
Listar Despachos



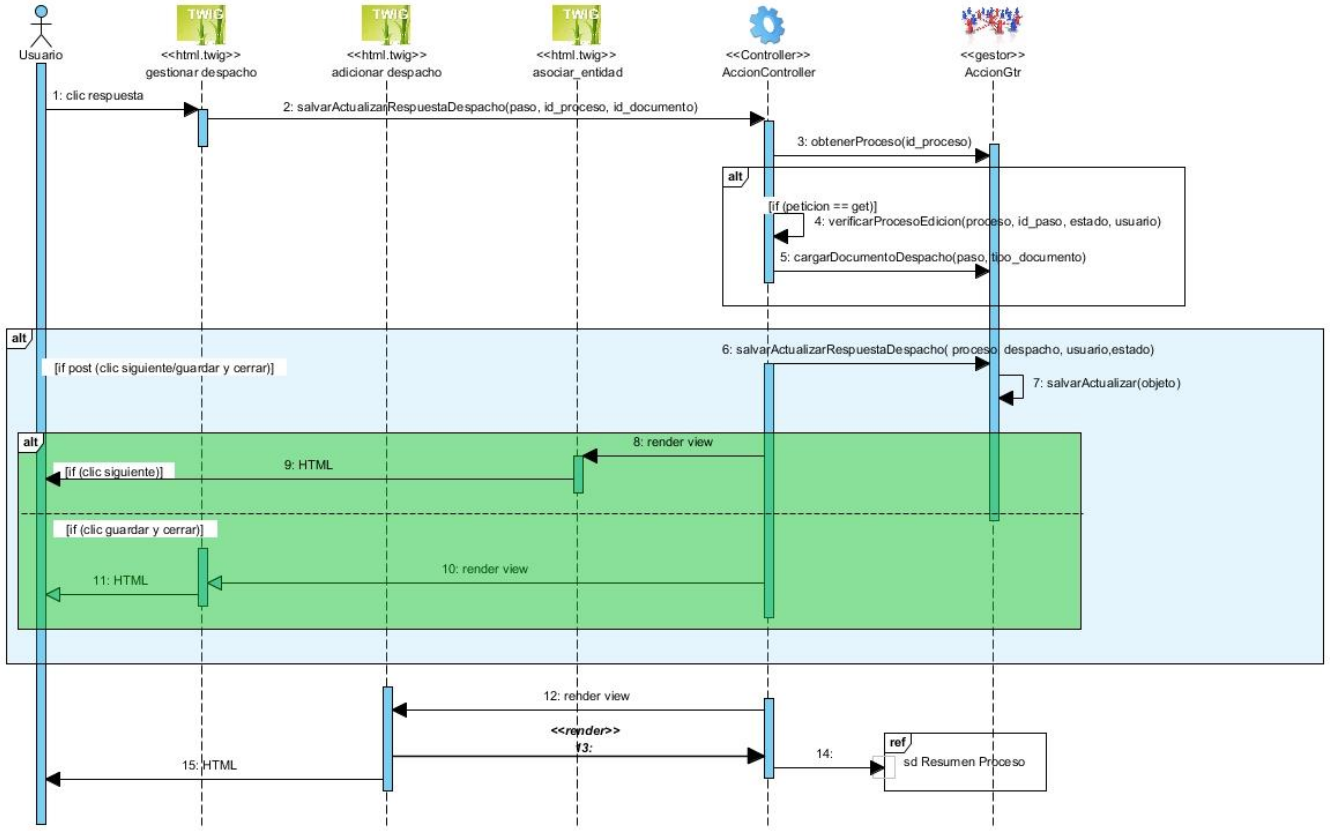
Notificación Despacho



Asociar Entidad



Respuesta Despacho



Anexo 5: Caso de prueba de la funcionalidad Adicionar Despacho

Escenario	Descripción	Fecha de emisión	Días	Diligencias a practicar	Nombre del lugar	Dirección del lugar	Provincia	Municipio	Fiscal actuante	Respuesta del sistema	Flujo central
EC 1.1 Adicionar datos correctos.	Adicionar los datos iniciales correctos del despacho	V 15/01/2014	V 13	V despacho	V Fiscalía provincial	V aranguren	V La Habana	V Guanabo	V Administrador	EL sistema guarda los datos.	Página principal/ VF/ Verificación Fiscal/ Registro/ Accion/ Diligencias/ Despacho/ GestionarDespacho
EC 1.2 Adicionar datos incorrectos.	Adicionar los datos iniciales incorrectos del despacho	I 14/12/2030	V 13	V despacho	V Fiscalía provincial	V aranguren	V La Habana	V Guanabo	V Administrador	El sistema no permite ingresar una fecha mayor a la fecha actual.	
		V 15/01/2014	I 35	V despacho	V Fiscalía provincial	V aranguren	V La Habana	V Guanabo	V Administrador	El sistema muestra un mensaje de error al introducir el día mayor que 30.	
		V 15/01/2014	V 13	V despacho	V Fiscalía provincial	V aranguren	V La Habana	V Guanabo	V Administrador		
		V 15/01/2014	V 13	V despacho	V Fiscalía provincial	V aranguren	V La Habana	V Guanabo	V Administrador		
		V 15/01/2014	V 13	V despacho	V Fiscalía provincial	V aranguren	V La Habana	V Guanabo	V Administrador		
		V 15/01/2014	V 13	V despacho	V Fiscalía provincial	V aranguren	N/A	V Guanabo	V Administrador		
		V 15/01/2014	V 13	V despacho	V Fiscalía provincial	V aranguren	V La Habana	N/A	V Administrador		
		V 15/01/2014	V 13	V despacho	V Fiscalía provincial	V aranguren	V La Habana	V Guanabo	N/A		
EC 1.3 Adicionar datos incompletos.	Adicionar los datos iniciales incompletos del despacho	I Campo vacío	V 13	V despacho	V Fiscalía provincial	V aranguren	V La Habana	V Guanabo	V Administrador	EL sistema muestra un mensaje por dejar campos vacíos.	
		V 15/01/2014	I Campo vacío	V despacho	V Fiscalía provincial	V aranguren	V La Habana	V Guanabo	V Administrador		
		V 15/01/2014	V 13	I Campo vacío	V Fiscalía provincial	V aranguren	V La Habana	V Guanabo	V Administrador		
		V 15/01/2014	V 13	V despacho	I Campo vacío	V aranguren	V La Habana	V Guanabo	V Administrador		
		V 15/01/2014	V 13	V despacho	V Fiscalía provincial	I Campo vacío	V La Habana	V Guanabo	V Administrador		
		V 15/01/2014	V 13	V despacho	V Fiscalía provincial	V aranguren	I Seleccione	V Guanabo	V Administrador		
		V 15/01/2014	V 13	V despacho	V Fiscalía provincial	V aranguren	V La Habana	I Seleccione	V Administrador		
		V 15/01/2014	V 13	V despacho	V Fiscalía provincial	V aranguren	V La Habana	V Guanabo	I Seleccione		
EC 1.4 Cerrar.	Adicionar los datos iniciales correctos del despacho	V 15/01/2014	V 13	V despacho	V Fiscalía provincial	V aranguren	V La Habana	V Guanabo	V Administrador	EL sistema guarda los datos.	

ANEXOS