UNIVERSIDAD DE LAS CIENCIAS INFORMATICAS

Diseño e implementación del subsistema Despacho de Medios de Transporte Internacional aéreos.

Trabajo de diploma para optar por el título de Ingeniero en Ciencias informáticas

Autor: Gretter Lisbet Rodríguez Valdés

Tutor: Manuel Ramón Almaguer Ochoa

Cuidad de la Habana

Junio 2010

Declaración de autoría

Declaro que soy el único autor de este trabajo y autorizo al <nombre área> de 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 _____.

Gretter Lisbet Rodríguez Valdés

_____

Ing. Manuel Ramón Almaguer Ochoa

_____

Agradecimientos

Agradecer antes que todo a papi y a mami, por su amor y su sacrificio, a Enri claro por aguantar mis regaños, quererme tanto y apoyarme siempre. Al resto de mi familia linda por su eterna preocupación, por brindarme todo lo que tienen cuando me ha hecho falta, necesitaría tres páginas más de este documento para poder mencionarlos a todos. A Ricardo por ser mi amor, mi guía y darme el ánimo y el impulso cuando los perdía. A Olgui por ser mi ejemplo y mi mejor amiga, a Doris, Eli, Mayde, Lili, Dane y Ara por su cariño y su ayuda incondicional. A Manuel, no podría olvidar cuanto me ha ayudado. Agradecer al resto de las personas que se vieron involucradas de alguna forma en la realización de este trabajo, decir más nombres significaría olvidar algunos y sería injusto. Me debo a ustedes.

Dedicatoria

A mi abuelita Concha aunque ya no esté, hubiera estado feliz.

A Fidel Castro, porque en este trabajo se ven realizados muchos de sus sueños.

A mi familia por su guía certera.

Resumen

La informatización de los procesos aduanales que se llevan a cabo en las sedes de la Aduana General de la República de Cuba (AGR)[1], constituye una prioridad debido a que el insuficiente control de los eventos que allí ocurren, podría acarrear graves consecuencias sociales, económicas y políticas para el país. Dentro de los escenarios involucrados y de gran trascendencia se encuentra el Control y Despacho de los MTI[2] aéreos; este proceso requiere consultar la información adelantada de pasajeros (API)[3] así como la información adelantada de las cargas (ACI)[4], información que ya se recepciona y se almacena con sistemas informáticos, pero no se cuenta con una herramienta que se integre eficazmente con estos sistemas para recuperar la información precisada. Este último inconveniente, unido a la realización manual del resto de las rutinas que se ejecutan durante un despacho, impide la generación de respuestas oportunas y eficientes ante las necesidades y trae consigo que el ínfimo tiempo que podría llevar realizar un despacho sea ahora mucho mayor. Para dar solución de la problemática antes descrita se planteó como problema a resolver: ¿Cómo materializar los requisitos explícitos en el análisis de los procesos de negocio involucrados en el despacho de Medios de Transporte aéreos*, siguiendo las* pautas de arquitectura establecidas?

El presente trabajo ofrece como solución el diseño y el código fuente de un sistema informático que cumple con las necesidades planteadas por los clientes y regido por las disposiciones del Decreto Ley de Aduanas.

[1]Por sus siglas en Español Aduana General de la República.
[2]Por sus siglas en Español Medios de Transporte Internacional.
[3]Por sus siglas en Inglés Advanced Passengers Information.
[4] Por sus siglas en Inglés Advanced Charge Information

Contenido

Índice de Figuras

Índice de tablas

Introducción

El mantenimiento, por más de 50 años, de políticas de embargo hacia Cuba le ha cerrado muchas puertas cuando se trata, particularmente, de las tecnologías de la información y las comunicaciones, colocando esta última en un franco atraso en comparación con los logros del primer mundo. En consecuencia, se ha propiciado la búsqueda de soluciones internas a problemas de gran impacto, teniendo como principal meta adecuarse a los avances tecnológicos del mundo desarrollado, de ahí es de donde nace nuestra Universidad, conjuntamente con sus proyectos productivos, muchos de estos son solo de alcance nacional, los cuales ofrecen soluciones informáticas a procesos que se realizan manualmente en nuestras entidades, o que se apoyan de herramientas informáticas de uso restrictivo y verdaderamente costoso en la mayoría de los casos, esto último proporciona deficiencias referentes a la calidad, realización en tiempo de las diferentes operaciones y uso adecuado de recursos.

Lograr un control eficiente de los procesos que se realizan en las fronteras cubanas se convirtió desde hace mucho tiempo en una de las prioridades del país, debido al grave impacto económico, político y social que conllevaría un incorrecto manejo y una insuficiente monitorización de las operaciones que allí ocurren. Con vista a dar respuesta a esta prioridad, surge la necesidad de construir el proyecto Sistema Único de Aduanas (SUA), que con la colaboración del Centro de Automatización y Digitalización de la Información (CADI) desarrollan actualmente el sistema para la gestión integral de procesos aduanales (GINA). La modernización de la Aduana General de la República de Cuba con la implantación de un sistema informático integral de producción nacional, contribuirá en gran medida con el desaduanamiento[5] de las mercancías en menor tiempo y con la correcta administración de los despachos, manteniendo el eficaz control de los mismos, disminuyendo grandemente los costos operativos, la evasión de impuestos y el contrabando, mejorando el servicio y la recaudación fiscal. En proporcionalidad con la modernización de la AGR, y en consecuencia, con la eficiencia, la productividad y con la calidad de los servicios, crecerá el comercio interno y externo, permitirá el desarrollo de las inversiones y aumentará el ingreso público.

Son múltiples los procedimientos aduanales que requieren automatización, este trabajo se centrará en las necesidades relacionadas con el perfeccionamiento del despacho de los medios

[5] Acto que por el cual la aduana autoriza a los interesados a disponer de una mercancía que ha sido objeto de un despacho.

de transporte internacional que es uno de los eventos indispensables que se realizan en las fronteras cubanas; para su desarrollo, se necesita consultar la información adelantada de pasajeros y de cargas, además de otros documentos expedidos por las aerolíneas; en la actualidad, varios problemas tecnológicos dificultan el análisis y la utilización de éstos datos que son imprescindibles, debido que gran parte de este trabajo se realiza de forma manual, revisando información que ya se almacena y se recupera con sistemas informáticos de procedencia nacional y que ya pertenecen a GINA, sin sumergirnos en el resto de las rutinas que completan un correcto despacho, lo que provoca que se prolongue considerablemente el mismo.

El despacho de Medios de Transporte aéreos, tiene como misión garantizar que en los arribos y salidas se cumpla con los requisitos y formalidades establecidas por las leyes cubanas, se debe comprobar que se haya abonado el importe de los derechos y otros adeudos que correspondan, el manejo no óptimo de la información relativa al mismo, impone retrasos y respuestas poco confiables en situaciones apremiantes donde está en juego la imposición de sanciones administrativas por violaciones o infracciones de la Normativa Aduanera[6], a nombre y en representación del Estado cubano, al representante de la aeronave (1).

Una de las principales razones que impulsan esta investigación es el costo excesivo de licencia o de soporte técnico de las aplicaciones existentes que ofrecen este tipo de soluciones informáticas, en las cuales se abundará más adelante. Además de esto, ninguno de estos sistemas se ajusta a las Normativas Aduaneras que se establecen en Decreto Ley De Aduanas de nuestra AGR, el mismo tiene por objeto “regular el control aduanero aplicable a la entrada, el tránsito, el cabotaje, el transbordo, el depósito y la salida del territorio nacional de mercancías, viajeros y sus equipajes, bienes y valores sujetos a regulaciones especiales, incluidas la flora y la fauna protegidas, y los medios en que se transporten”. Además de esto último las transformaciones económicas ocurridas en Cuba durante los últimos años y los recientes cambios en las relaciones comerciales con la creación de sociedades mercantiles, la admisión de representaciones e inversiones extranjeras, así como el auge del turismo internacional, requieren de un proceso de perfeccionamiento de los procedimientos que se llevan a cabo en las sedes de la AGR de todo el país para brindar las facilidades necesarias y la adecuada comprensión de las regulaciones establecidas (1). Como parte de este proceso de

[6] Conjunto de disposiciones vigentes en materia aduanera y arancelaria que incluye el Decreto Ley de Aduanas y sus disposiciones complementarias, así como las emitidas por otros organismos competentes aplicables por la Aduana, tomado de DECRETO LEY No. 162

perfeccionamiento surge la prioritaria necesidad de la modernización de la AGR de Cuba, implícita en esta necesidad se encuentra la informatización de los procesos aduanales que allí tienen lugar, incluyéndose la implementación de un sistema para controlar la información relacionada con los MTI aéreos que cruzan nuestras fronteras diariamente.

Para dar respuesta a las necesidades referidas anteriormente, un equipo de trabajo de la UCI del proyecto SUA, con la asistencia y colaboración de los trabajadores del CADI realizaron el análisis y la descripción de los procesos de negocio involucrados en el despacho de los Medios de Transporte aéreos, para darle continuidad a esta tarea y enfrascados en el perfeccionamiento de este proceso se propone como ***problema a resolver:***

¿Cómo materializar los requisitos *explícitos en el análisis de los procesos de negocio involucrados en el despacho de Medios de Transporte aéreos, siguiendo las* pautas de arquitectura establecidas?

El sistema GINA posee un gran alcance en la solución que ofrece, hasta la actualidad, está conformado por varias aplicaciones o subsistemas especializados en cada área de trabajo aduanero, que resolverían de forma parcial los problemas de la Aduana General de la República, puesto que quedan muchos procesos por mejorar, de modo, que el ***objeto de estudio*** de la presente investigación es la informatización de la gestión de procesos aduanales y el ***campo de acción*** la informatización de los procesos relacionados con el control y despacho de los Medios de Transporte Internacional aéreos.

Para la solución del problema se propone como ***objetivo general*** diseñar e implementar una aplicación para el Despacho de los Medios de Transporte Aéreos para el Sistema Integral de Gestión de la Aduana (GINA) siguiendo la arquitectura definida.

Objetivos específicos

- ✓ Elaborar el Modelo del Diseño.
- ✓ Establecer la Arquitectura del subsistema.
- ✓ Elaborar el Modelo de Datos.
- ✓ Obtener el Código Fuente del Sistema.

Para una mejor comprensión del documento a continuación se dará una breve descripción de su estructura:

Capítulo I: Fundamentación teórica.

En este capítulo se expondrá el análisis de la bibliografía consultada, donde se mostrará lo más actual referente a las soluciones informáticas que existen en el mundo para la gestión de procesos aduanales, haciendo especial énfasis en las herramientas desarrolladas con el fin de apoyar el proceso de despacho de Medios de Transporte Internacional. Se abordarán temas referentes a las buenas prácticas que se aplican hoy para el desarrollo de aplicaciones. Se tomará partida de lo investigado para darle el mejor enfoque posible a la solución que se propondrá en el presente trabajo.

Capítulo II: Diseño de la solución propuesta.

En este capítulo se mostrará el diseño de la solución, se hará referencia a los artefactos correspondientes que se obtendrán durante el desarrollo de esta actividad para una mejor comprensión del contenido.

Capítulo III: Implementación y prueba.

Durante este capítulo se desarrollarán las actividades de implementación y prueba de la solución propuesta.

Capítulo 1. Fundamentación teórica

Introducción

Es una realidad en la sociedad moderna que la industria del software ha penetrado en casi todos los aspectos y procesos de las distintas organizaciones, por tanto la construcción de sistemas informáticos confiables se ha convertido en uno de los temas más sensibles y por tanto de mayor importancia para los desarrolladores de software durante el ciclo de vida de los procesos de desarrollo de software.

Muchas han sido las investigaciones en pos de buscar mecanismos que guíen a los desarrolladores de software a la adopción de estrategias y principios que disminuyan potencialmente los problemas de seguridad. Se puede decir que se ha avanzado mucho en el tema, se han documentado disímiles investigaciones que pueden tomarse como guías e integrarlas a los procesos de desarrollo de software con el objetivo de convertirlos en procesos más estrictos y centrados en la seguridad. Por otro lado muchas de las metodologías de desarrollo de software modernas están diseñadas para minimizar el número de vulnerabilidades presentes en el diseño, la programación y la documentación, para detectarlas y eliminarlas en etapas tempranas y menos costosas en el ciclo de vida de desarrollo.

Muchos son los autores que abogan no solo por la aplicación de buenas prácticas a la hora de escribir el código de una aplicación, sino también por la necesidad de emplear buenas prácticas durante el diseño, las pruebas y en la documentación. Todos estos aspectos son importantes para entregar un sistema confiable, seguro, es indispensable además adoptar procesos de desarrollo de software disciplinados e incluir estos temas.

Antes de comenzar el desarrollo de una aplicación es necesario asegurar que existe un Ciclo de Vida de Desarrollo Software adecuado, en el cual la seguridad sea inherente (2) (3), en (3) se propone la incorporación de un grupo de mejoras para agregar responsabilidad y estructura en términos de seguridad al proceso de desarrollo de software, sus autores ponen especial atención a la cuestión de proporcionar un buen adiestramiento al equipo de trabajo enfocado en la obtención de una cultura relacionada con la construcción de software seguro, exponiendo esto como una garantía para lograrlo; formulan, que para hacer software seguro es imprescindible elevar el nivel de los conocimientos acerca de seguridad de todo el equipo de trabajo para poder insertar satisfactoriamente el tema en cada una de las partes que conforman el ciclo de vida de desarrollo de un software. Asegurar también que estén implementadas las

políticas y estándares de seguridad adecuadas para el equipo de desarrollo, (aclarar que ninguna política o estándar puede cubrir todas las situaciones con las que se enfrentará un equipo de desarrollo), y que estén desarrolladas las métricas y criterios de medición ya que los criterios de medición proporcionan una visibilidad de los defectos tanto en el proceso como en el producto y las métricas se pueden utilizar en caso de que sea necesario modificar el proceso de desarrollo para poder capturar los datos necesarios.

El caso que nos ocupa requiere el diseño y la implementación de una aplicación web con características muy particulares propias de sistemas de control aduanero, que debe ser protegida mayoritariamente de acceso no autorizado a diferentes funcionalidades, debe ser defendida además del uso de usuarios malintencionados que puedan extraer información de valor y facilitársela a entidades externas completamente ajenas al gobierno cubano y que la puedan utilizar en su contra.

En este capítulo se dedicará una breve sección al tema del desarrollo de sistemas de información para el control aduanero en el mundo, específicamente los que apoyan el despacho de los MTI, se estará abordando además temas relacionados con las buenas prácticas recomendadas en bibliografías reconocidas y actualizadas, relativas al diseño y la implementación de aplicaciones, y se especificará cuáles se adoptarán para el desarrollo del presente trabajo para la entrega de una solución de calidad.

Desarrollo

Sistemas informáticos para las aduanas

Son muchos los sistemas informáticos desarrollados en el mundo para apoyar los procesos aduaneros. Todos estos, o al menos los estudiados, constan de un módulo, o varios, que controlan de alguna forma del arribo y/o salida de los MTI mediante despachos. Podrían mencionarse varios, el SIM[7] en Argentina, aunque en estos momentos este país se encuentra enfrascado en su sustitución por un sistema de última generación llamado Malvina; Isidora de Chile, el cual cuenta con un módulo aéreo para el manejo de los manifiestos y las declaraciones de ingreso y salida de MTI; SIDUNEA de Venezuela, el mismo cuenta con el módulo de transporte que sirve para la preparación y transmisión de detalles del transporte de carga en formato electrónico, se utiliza también para generar el formato electrónico del Manifiesto de Carga y sus Documentos de Transporte, además se utiliza junto a otros módulos

[7] Sistema Informático María por sus siglas en español

del sistema para el control de la carga, incluyendo el retiro de las mercancías y el manejo de los inventarios de carga. Cuba por su lado, tiene la imperiosa necesidad de contar con un sistema que logre manejar los procesos aduaneros en concordancia con las normativas de las leyes cubanas, para este caso en particular, un sistema que apoye las actividades de despacho con los representantes de los MTI que se integre a GINA de forma exitosa, características que no satisfacen ninguno de los mencionados anteriormente. A partir de este breve análisis se dedujo la necesidad de crear un sistema de producción nacional, con costos mínimos para su construcción, que se integre a GINA y creado bajo la directa supervisión de los especialistas informáticos del CADI y de los propios trabajadores de la AGR.

Diseño e implementación de aplicaciones

Desarrollo con Symfony

La arquitectura definida para el sistema GINA establece la utilización del framework[8] Symfony para su desarrollo, este marco de trabajo diseñado para sistemas web complejos y grandes, como es el caso, simplifica el desarrollo mediante la informatización de algunos de los patrones utilizados para resolver tareas comunes, además proporciona una estructura al código fuente, forzando al desarrollador a crear código más legible y más fácil de mantener y nos facilita la programación encapsulando operaciones complejas en instrucciones sencillas.

Symfony proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación (4). Como ejemplos de algunas de estas herramientas y tareas podrían mencionarse:

- La utilización de Propel para lograr un acceso efectivo a una base de datos relacional desde un contexto orientado a objetos permitiendo un alto nivel de abstracción y una fácil portabilidad.
- El uso del Framework de formularios (5) para la gestión segura de los datos, proporcionando al programador todas las herramientas necesarias para mostrar y validar fácilmente datos en un formulario de forma similar a los objetos.
- La utilización de Plugins (6) para agrupar todo el código diseminado por diferentes archivos y reutilizar este código en otros proyectos.

[8] Traducción: Marco de trabajo. Puede definirse como un marco de aplicación o conjunto de bibliotecas orientadas a la reutilización a muy gran escala de componentes software para el desarrollo rápido de aplicaciones.

Todos estos rasgos y herramientas de Symfony permiten realizar un grupo de tareas para reducir el tiempo de desarrollo y para alcanzar una mayor eficiencia en el trabajo de la aplicación. Proporciona una estructura de ficheros y un grupo de archivos de configuración integrado, muy sencillos de usar y lo suficientemente potente para personalizar cualquier aspecto del framework así como la forma en que interactúan las aplicaciones. Por otro lado, partiendo de un modelo de datos y mediante el ORM se obtiene un modelo divido en dos capas capaces de complementarse ofreciendo instrumentos que nos dan la posibilidad de acceder a los datos sin atarnos a las particularidades de una de base de datos específica y ahorrándonos gran cantidad de trabajo, pues si se cambia de sistema gestor de bases de datos, solamente es necesario actualizar la capa de abstracción de la base de datos. El framework de formularios nos facilita todo un mecanismo, verdaderamente flexible y al mismo tiempo poderoso, que permite una gestión segura de los datos.

Teniendo en cuenta las facilidades que nos proporciona el marco de trabajo Symfony las tareas de diseñar e implementar la aplicación en cuestión, se tornan relativamente más abreviadas y centradas en el diseño del modelo que contendrá y manejará la lógica del negocio descrita en el análisis. En las siguientes secciones del documento se estará analizando las mejores prácticas que se utilizan en el mundo para llevar a cabo el diseño y la implementación de aplicaciones y la forma en que se integrarán algunas de ellas con las herramientas de Symfony para lograr un diseño y una implementación de calidad.

Con la guía de RUP

Los equipos de trabajo involucrados en el desarrollo de los subsistemas de GINA trabajan guiados por la metodología de desarrollo de software Rational Unified Process; esta metodología captura varias de las mejores prácticas en el desarrollo moderno de software en una forma que es aplicable para un amplio rango de proyectos y organizaciones. Constituye una guía de cómo utilizar de manera efectiva UML. La metodología RUP permite seleccionar fácilmente el conjunto de componentes de proceso que se ajustan a las necesidades específicas de cada proyecto. Se pueden alcanzar resultados predecibles unificando el equipo con procesos comunes que optimicen la comunicación y creen un entendimiento común para todas las tareas, responsabilidades y artefactos (7).

Diseño en el ciclo de vida de desarrollo de software según RUP y adaptaciones en el proyecto

En el diseño se modela el sistema y se encuentra su forma para que soporte todos los requisitos, incluyendo todos los nos funcionales y otras restricciones que se le suponen. Una entrada esencial en el diseño es el resultado del análisis, (en este último caso en el proyecto la entrada del diseño es la descripción textual de los requisitos y los prototipos de las interfaces de la aplicación) el cual proporciona una comprensión detallada de los requisitos e impone una estructura del sistema, la cual debe ser conservada. De forma más concreta los propósitos del diseño son:

- Adquirir una comprensión en profundidad de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, tecnologías de UI (interfaz de usuario), sistemas operativos, tecnologías de distribución, tecnologías de gestión de transacciones, etc.
- Crear una entrada apropiada y un punto de partida para las actividades de implementación subsiguientes.
- Crear una abstracción sin costuras de la implementación del sistema, en el sentido de que la implementación en un refinamiento directo del diseño que rellena lo existente sin cambiar la estructura. Esto permite la utilización de tecnologías como la generación de código y la ingeniería de ida y vuelta entre el diseño y la implementación (7).

Artefactos que se generan durante el diseño dentro del proyecto Aduana

- Artefacto: Modelo de diseño
- Artefacto: Clase del diseño
- Artefacto: “Diseño por requisitos”
    - Diagramas de clases del diseño por requisitos o escenarios
    - Diagramas de interacción por requisitos o escenarios.
    - Requisitos de la implementación
- Artefacto: Subsistema de diseño (en caso de necesitarlo)
- Artefacto interfaz (en caso de necesitarlo)
- Artefacto descripción de la arquitectura (Vista del modelo de diseño)

Buenas prácticas recomendadas durante la fase de diseño de software

La fase de diseño no es sólo uno de los momentos más rentables para identificar defectos, sino que puede ser uno de los más eficaces para hacer cambios. Un ejemplo podría ser, si se

identifica que el diseño establece que se tomen decisiones de autorización en varios lugares, puede ser apropiado considerar un componente central para la autorización. Si la aplicación está realizando la validación de datos en varios lugares, puede ser apropiado desarrollar un marco central de validación (fijar la validación de entradas en un solo lugar, en vez de en varios lugares, es mucho menos costoso) (3) (2).

Howard y LeBlanc's,(3) plantean que existen cifras que muestran que se necesita diez veces más tiempo, dinero y esfuerzo para corregir un error en la fase de implementación que en la fase de diseño y diez veces más en la fase de prueba que en la fase de implementación, y así sucesivamente. Se puede decir con seguridad que es más fácil implementar algo que no es necesario revisar ya que fue diseñado correctamente. La lección es determinar todos o casi todos los requisitos de seguridad y diseñar correctamente el sistema tan pronto como sea posible.

Teóricamente, el desarrollo es la implementación del diseño. Sin embargo, en la práctica, muchas decisiones de diseño se hacen durante el desarrollo del código. Esas son a menudo decisiones pequeñas que o bien son demasiado detalladas para ser descritas en el diseño o en otros casos, son cuestiones acerca de las cuales no fueron ofrecidas políticas o estándares. Si el diseño y la arquitectura no son adecuados el desarrollador se tendrá que enfrentar a muchas decisiones. Si hubo insuficiencia en las políticas y normas, el desarrollador se enfrentará aún más a la toma de decisiones (2).

A la hora de realizar el diseño se debe tratar la seguridad como un requisito más del producto, y no como un aspecto confuso y difícil de llevar a la práctica en el desarrollo de software, debe ser prioridad y no una tarea pendiente, solamente realizada cuando sea conveniente. Quiénes son sus clientes y cuáles son los requisitos de seguridad que necesitan en el producto final son aspectos fundamentales que deben ser del conocimiento de todo el equipo de desarrollo para que todo el trabajo esté enfocado en materializarlos (3).

Desarrollo con patrones de diseño de clases

Los patrones de diseño constituyen una poderosa arma para enfrentar los problemas del desarrollo de software, sin embargo, se deben manejar con cautela, pues la inclusión patrones innecesarios puede llevar al incremento de la complejidad, por lo que para aprovechar al máximo su uso se debe tener muy en cuenta qué patrones se necesita utilizar, y cuáles están

asociados con el funcionamiento de éstos, para evitar la introducción de patrones incompatibles entre sí (8).

La idea del uso de patrones de diseño es la de poder establecer soluciones a problemas recursivos en el desarrollo de software, de manera que se los pueda utilizar siempre que se presente el mismo problema; para poder cumplir con esto, las soluciones deberán ser bien definidas.

"Nuestro diseño deberá ser específico para el problema que se tiene en frente, pero deberá ser también lo suficientemente generalizado para poder orientarlo a la solución de futuros problemas y requerimientos" (8).

Los patrones de diseño aportan en varios aspectos a mejorar el proceso de desarrollo de software, entre los aportes más importantes se deben destacar (8):

- Permiten mantener una alta cohesión[9]

Los patrones de diseño permiten que se creen clases de cooperación mutua, pero al mismo tiempo las mantienen lo suficientemente independientes del funcionamiento de otras, por lo cual se obtiene un bajo acoplamiento[10] de los componentes del software y una alta cohesión entre los módulos.

- Facilitan la reutilización[11] de software

Este beneficio se deriva de la definición misma de los patrones de diseño, y constituye en una fortaleza para aquellos que los dominan, pues consiste en optimizar el diseño de una solución a un problema de desarrollo determinado, la garantía del uso de los patrones de diseño radica en que se respaldan en la experiencia de la reutilización de los mismo, al enfrentarlos varias veces a problemas similares, pero en entornos diferentes.

[9] Define la forma en que se agrupan unidades de software (módulos, funciones, subrutinas, bibliotecas, etc.) en una unidad mayor. Por ejemplo, la forma en que se agrupan funciones en una biblioteca de funciones o en la forma en que se agrupan métodos de una clase, mientras mayor sea la cohesión, mejor será la calidad de software.

[10] Grado de interdependencia entre los componentes de un sistema informático (módulos, funciones, subrutinas, bibliotecas, etc.); es decir, define el nivel de dependencia entre los elementos del sistema, mientras menor sea al acoplamiento mayor será la calidad del software.

[11] (Reutilización de código): Consiste en el uso de software existente para desarrollar un nuevo software; la idea es que parte, o todo el código, de un programa de computadora escrito una vez sea, o pueda ser usado, en otros programas. La reutilización de códigos programados es una técnica común que intenta ahorrar tiempo y energía, reduciendo el trabajo redundante.

- Facilitan la identificación de objetos

Permiten identificar abstracciones menos obvias pero más efectivas que aquellas que saltan fácilmente a la vista al momento de diseñar una solución.

- Permiten distribuir necesidades

Al ser organizados de manera jerárquica, en clases y protocolos, los patrones de diseño distribuyen efectivamente las responsabilidades del equipo de desarrollo, pues facilita el desarrollo individual de soluciones que se integran posteriormente al funcionamiento general del sistema.

- Garantizan reusabilidad, extensibilidad y mantenimiento[12]

Esto gracias a que cada patrón permite controlar el tamaño y ubicación de los elementos a ser implementados, garantizando un alto nivel de calidad y simplificando el trabajo en grupo por el vocabulario estándar que se maneja en cada componente, facilitando principalmente el mantenimiento adaptativo (integración de nuevos requerimientos o cambios de los existentes).

- Proveen soluciones concretas

Esta podría ser la más importante ventaja del uso de patrones de diseño, ya que estos proveen una solución concreta a un problema haciendo frente a casi todas las metodologías o tecnologías empleadas y además considerando las consecuencias, beneficios y viabilidad de la solución propuesta; esto se debe a que funcionan como una guía para resolver problemas comunes de programación.

- Proveen una solución más equilibrada

Debido a que el empleo de patrones requiere un análisis profundo, se puede encontrar el punto de equilibrio entre las restricciones y objetivos de la solución, obteniendo como resultado un sistema fácil de emplear y que cumple con las expectativas de funcionalidad.

- Facilitan el entendimiento de la solución gracias al encapsulamiento

[12] Consiste de un conjunto de tareas necesarias para asegurar el buen funcionamiento de un aplicativo, el mantenimiento que se clasifica en preventivo, correctivo y complementario.

Dado que se basan en la experiencia, resulta sencillo encapsular el conocimiento detallado sobre un tipo de problema y sus soluciones; de esta manera es fácil entenderlo sin adentrarse a ámbitos como codificación.

Los sistemas desarrollados con patrones de diseño, además de facilitar la integración de nuevos requerimientos o cambio de los existentes, permiten realizar esto sin afectar la estructura del sistema; la clave para lograrlo es anticiparse a los nuevos requisitos de tal manera que la selección del patrón idóneo asegure una evolución adecuada.

Al usar patrones de diseño en el desarrollo de software, el incremento de eficiencia en el diseño es muy notable, ya que ante un problema no es necesario generar una nueva solución: basta con analizar las existentes y tomar las más adecuada con la tranquilidad de saber que estas soluciones son probadas con anterioridad y el riesgo de fallo es prácticamente nulo.

Clasificación

Para un mejor entendimiento, los patrones de diseño GOF[13] se clasifican en tres grandes grupos (8):

- Patrones de creación:

Los patrones de creación están asociados al proceso de creación de los objetos, entre objetos se delegan los procesos de creación. Entre los principales patrones que se encuentran en este grupo se pueden listar los siguientes:

  - Abstract Factory: su objetivo es trabajar con objetos de distintas familias de manera que no se mezclen entre sí, haciendo transparente el tipo de familia concreta que se esté usando.
  - Builder: su función es facilitar la abstracción del proceso de creación de un objeto complejo, centralizándolo en un único punto.
  - Factory Method: su misión es centralizar en una clase constructora la creación de objetos de un subtipo de un tipo determinado.
  - Prototype; se implementa para crear nuevos objetos clonándolos de una instancia ya existente.
  - Singleton: tiene como tarea garantizar la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a la misma.

[13] Gang of Four por sus siglas en inglés

- Patrones estructurales:

Los patrones estructurales tratan la composición de clases u objetos; para lo cual hacen uso de dos recursos, dependiendo del tipo de composición, es decir que para clases es necesaria la herencia mientras que para los objetos se describen formas de ensamblar objetos. Entre los principales patrones que se encuentran en este grupo se pueden listar los siguientes:

  - Adapter: este patrón adapta una interface para que pueda ser utilizada por otra clase que de otro modo no podría ser utilizada.
  - Bridge: la función del puente es una abstracción de su implementación.
  - Composite: su función es manejar objetos compuestos como si se tratase de uno simple.
  - Decorator: facilita la añadidura de funcionalidad a una clase dinámicamente.
  - Facade: provee de una interface unificada y simple para acceder a una interface o grupo de interfaces de un subsistema.
  - Flyweight: reduce la redundancia[14] en situaciones en las que se presentan grandes cantidades de objetos que poseen idéntica información.
  - Proxy: su función es mantener un representante de un objeto.

- Patrones de comportamiento

Los patrones de comportamiento se caracterizan por el modo en que las clases y objetos interactúan y se distribuyen la responsabilidad. Este tipo de patrones, en las clases hacen uso de la herencia para describir algoritmos y flujos de control; mientras que los de objetos describen cómo cooperan un grupo de objetos para realizar una tarea que ninguno podría llevar a cabo por sí solo. Entre los principales patrones que se encuentran en este grupo se pueden listar los siguientes:

  - Chain of Responsibility: su función es establecer la línea que deber llevar los mensajes para que los objetos realicen la tarea indicada.
  - Command: su función es encapsular una operación en un objeto, permitiendo ejecutarla sin necesidad de conocer el contenido de la misma.

[14] Consiste de la repetición innecesaria de código, generalmente repetición de funciones, rutinas o algoritmos, etc. que podrían ser codificadas una sola vez y utilizadas desde otros elementos.

- Interpreter: tiene el propósito de definir una gramática para un lenguaje dado, así como las herramientas necesarias para interpretarlo.
- Iterator: realiza recorridos sobre objetos compuestos independientemente de la implementación de estos.
- Mediator: define un objeto que coordine la comunicación entre otros de distintas clases, pero que funcionan como un conjunto.
- Memento: su objetivo es volver a estados anteriores del sistema.
- Observer: su función es definir una dependencia de uno a muchos entre objetos, de forma que cuando uno cambie de estado se notifique y actualicen automáticamente todos los que dependen de él.
- State: altera la conducta interna de un objeto**.**
- Strategy: define una familia de algoritmos, encapsulados e intercambiables. La estrategia permite al algoritmo variar independientemente de los clientes que lo usan.
- Template Method: se caracteriza por redefinir subclases siguiendo ciertos pasos de un algoritmo sin cambiar la estructura del mismo.
- Visitor: representa un funcionamiento a ser realizado por los elementos de la estructura del objeto, además permite definir un nuevo funcionamiento sin cambiar las clases de los elementos en que opera.

Patrones GRASP[15]

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Algunos ejemplos de los más conocidos serían los siguientes:

- Experto:

  Problema: ¿No existe un principio que se debe seguir para asignar las responsabilidades en el diseño orientado a objetos?

  Solución: Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.

- Creador

Problema: ¿Quién debería ser responsable de crear una nueva instancia de alguna clase?

[15] General Responsibility Asignment Software Patterns por sus siglas en inglés.

Solución: Se asigna la responsabilidad de que una clase B cree un Objeto de la clase A solamente cuando:

- B agrega los objetos A.
- B contiene los objetos A.
- B registra las instancias de los objetos A.
- B utiliza específicamente los objetos A.
- B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado (así que B es un Experto respecto a la creación de A).

- Controlador

Problema: ¿Quien debería encargarse de atender un evento del sistema?

Solución: Asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas

- Bajo acoplamiento

Problema: ¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización? Cuando una clase recurre a muchas otras:

- Los cambios de las clases afines ocasionan cambios locales.
- Es más difícil de entender.
- Es más difícil de reutilizar porque se requiere la presencia de otras clases de las que depende.

Solución: Asignar una responsabilidad para mantener pocas dependencias entre las clases.

- Alta cohesión

Problema: ¿Cómo mantener la complejidad dentro de límites manejables?

Solución: Cada elemento de nuestro diseño debe realizar una labor única dentro del sistema. Clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.

- No hables con extraños

Problema: ¿A quién debe invocar un método?

Solución: Un Objeto, solamente invocará a métodos de:

- Del Objeto mismo (this)
- De su área de parámetros
- Un elemento de una colección que sea atributo del mismo
- Un objeto creado en su propio ámbito

En la implementación se parte de los artefactos del diseño y se implementa el sistema en términos de componentes: ficheros de código fuente, scripts, ejecutables y similares. La mayor parte de la arquitectura del sistema es definida durante el diseño, siendo el propósito principal de la implementación desarrollar la arquitectura y el sistema como un todo. De forma más específica los propósitos de la implementación son:

- Planificar las integraciones del sistema necesarias en cada iteración. Se sigue para ello un enfoque incremental, lo que da lugar a un sistema que se implementa en una sucesión de pasos pequeños y manejables.
- Distribuir el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue.
- Implementar las clases y subsistemas encontrados durante el diseño. En particular, las clases se implementan como componentes de fichero que contienen código fuente.
- Probar los componentes individualmente, antes de ser enviados para ser integrados y llevar a cabo comprobaciones del sistema.

Los artefactos que son resultados de esta fase según la guía de RUP (7):

- Modelo de implementación
- Componentes
- Subsistemas de implementación (en caso de necesitarlo)
- Interfaces (en caso de necesitarlo)

Buenas prácticas recomendadas durante la fase implementación de software.

La implementación incluye la escritura y la depuración del código de la aplicación, y las buenas prácticas de esta etapa están enfocadas en hacer que los desarrolladores escriban código de calidad lo que significa código seguro y fácil de mantener.

A continuación en el documento se abordan algunas de las prácticas que proponen Howard y LeBlanc's en (3) que se deben seguir en la fase de desarrollo para la codificación segura de aplicaciones.

- Definir las directrices de codificación segura.

Es preciso definir un conjunto mínimo de directrices de codificación para el equipo, donde debe quedar reflejado cómo manejar los buffers, cómo se deben tratar los datos que no son de

confianza, cómo se deben cifrar los datos y así sucesivamente. Estos solo serán lineamientos mínimos, el equipo debe tratar de superar las directrices.

- Revisar defectos antiguos.

La premisa es que usted debe aprender de los errores del pasado para que no los repita. Pídale a alguien en su propio equipo el proceso de determinar por qué se producen errores y qué se puede hacer para evitar que ocurran de nuevo.

- Examen externo de seguridad.

Es provechoso contar con una entidad externa, como una empresa de consultoría de seguridad, que revise sus códigos y planes.

- Sea consciente de su cuenta de errores.

Los errores de seguridad se encontrarán si se enfoca la búsqueda hacia ellos, pero hay que asegurarse de que su total no se convierte en inimaginable. Una regla usada para algunos grupos es permitir a los desarrolladores no tener más de cinco errores activos al mismo tiempo. Además el total de números de errores para el producto no debe exceder en tres veces la cantidad de desarrolladores en el grupo. Si la regla se rompe, los desarrolladores deben abandonar la búsqueda de otros errores y dedicarse a resolver los encontrados. Vale aclarar que esta es una regla usada para algunos grupos, cada proyecto puede definir su regla. Lleve un registro de mediciones de fallas. Cuando una falla de seguridad es encontrada en el código, esta debe ser registrada en su base de datos de seguimiento de errores, como normalmente se debe hacer, pero además debe añadírsele un campo adicional a la base de datos donde se pueda definir qué tipo de amenaza a la seguridad provoca esta falla. Esto permite clasificar las amenazas así como al final del proceso analizar las causas que propiciaron esas fallas encontradas

Principios de seguridad en la programación

- Minimizar la zona de ataque.

Cuando se instala más código y se escucha a través de más protocolos basados en red, se debe comprender que los atacantes tienen más puntos potenciales de entrada. Es importante que se mantengan estos puntos de entrada en un mínimo y que se permitan a los usuarios habilitar la funcionalidad según la necesite.

- Aprender de errores

Recurrir a las experiencias pasadas constituye una invalorable herramienta para no cometer errores ya rectificados en alguna ocasión, en el libro (3) se plasman algunas citas válidas de retomar aquí:

“History is a vast early warning system”.

—Norman Cousins (1915–1990), American editor, writer, and author

“Those who cannot remember the past are condemned to repeat it”.

—George Santayana (1863–1952), Spanish-born American philosopher and writer

“There is only one thing more painful than learning from experience and that is not learning from experience”.

---Archibald McLeish (1892–1982), American poet

Cada error detectado en el sistema en desarrollo o en el de algún competidor debe tomarse como una fuente de conocimiento para aprender y acumular experiencia, lo que evitaría cometer los mismos costosos errores y ganar en tiempo y calidad.

- Emplear valores predeterminados seguros.

Minimizar la zona de ataque, implica además la definición de una instalación por defecto de forma segura para su producto. El empleo de valores predeterminados seguro es uno de los objetivos más difíciles pero importantes para un desarrollador de aplicaciones. Deben elegirse las características apropiadas para sus usuarios. Las características menos frecuentemente utilizadas deben ser desactivadas por defecto para reducir la exposición potencial de seguridad. Si una función no se está ejecutando, no puede ser vulnerable a ataques.

- Uso de la defensa en profundidad.

Este es un principio sencillo, imagine que su aplicación es el último componente en pie y todos los mecanismos de protección defensiva han sido destruidos. Entonces debe protegerse. Por ejemplo, si espera de un firewall para la protección de la aplicación, constrúyala, como si el mismo se hubiera visto comprometido.

- Uso mínimo de privilegios

Todas las solicitudes se deben ejecutar con el menor privilegio para hacer el trabajo y no más. A menudo existen productos que deben ejecutarse en el contexto de seguridad con una cuenta administrativa, mientras los desarrolladores del producto podían haberlo diseñado para no exigir tales cuentas privilegiadas.

- Compatibilidad con versiones anteriores

La compatibilidad con versiones anteriores es otra razón para entregar productos seguros con valores predeterminados seguros. Imagine que su aplicación está en uso por grandes corporaciones, empresas con miles de equipos cliente. Un protocolo que usted diseñó es inseguro de alguna manera. Cinco años y nueve versiones posteriores, usted hace una actualización a la aplicación con un protocolo más seguro. Sin embargo este no es compatible con la antigua versión del protocolo y cualquier equipo que fuera actualizado con el protocolo actual, ya no podrá comunicarse con cualquier otra versión de la aplicación. Un buen enfoque a este problema es hacer las aplicaciones lo más configurables posibles.

- Asuma que los sistemas externos son inseguros

Asumir que los sistemas externos son inseguros está relacionado con la defensa en profundidad. Considere que cualquier información que reciba desde un sistema del cual no tiene control total es insegura y puede ser una fuente de ataque.

- Fallo en modo seguro

Las fallas en modo seguro garantizan que la aplicación no revele ningún dato en caso de fallo que no sea revelado ordinariamente y que sus datos no puedan ser manipulados; en el caso contrario “fallo en modo inseguro”, la aplicación revela más de lo que debería, sus datos pueden ser manipulados o peor.

- Nunca implementar seguridad con “oscuridad”

Siempre asuma que un atacante sabe todo lo que usted sabe, asuma que el atacante tiene acceso a todo el código fuente y todos los diseños. Incluso si esto no es cierto, es trivialmente fácil para un atacante determinar la información oculta. La oscuridad es una defensa útil, siempre y cuando no es su única defensa. En otras palabras, es muy válido usar la oscuridad como una pequeña parte de una estrategia global de defensa en profundidad.

- No mezcle código y datos

Una vez que se añada código a los datos, "los datos" se convierten en peligrosos. Si su aplicación soporta mezcla de código y datos, usted debe prohibir por defecto la ejecución del código y permitir al usuario la decisión de su ejecución. Esta es la política por defecto en Microsoft Office XP. Las macros no funcionan por defecto, el usuario decide si va a permitir que el código de las macros se ejecute.

Howard y LeBlanc's en su libro (3) destacan la necesidad de corregir correctamente los errores que son hallados en el código, de forma que se busquen errores similares en otros lugares de la aplicación que de seguro se hallarán. Plantean además que es necesario dotar los sistemas de las características correctas y precisas referentes a la seguridad, la implantación de rasgos que no son lo que verdaderamente necesita la aplicación en una pérdida de tiempo total.

- No darle ninguna información al atacante

Los mensajes de error enmascarados son la perdición de los usuarios normales y pueden conllevar a realización de consultas para soporte verdaderamente costosas. Sin embargo se debe balancear la información que se ofrece a los posibles usuarios que usen su aplicación con malas intenciones. Un ejemplo muy claro podría ser el caso en el que un atacante estuviera intentando acceder a algún fichero, no se debería devolver un mensaje de error como este: "Imposible localizar el archivo *archivo1.txt* en *C:\archivosSecretos\documentos*", revelando de esa forma cierta información que puede ser útil para el atacante. Se deben devolver mensajes sencillos: "*Petición fallida*" y archivar el error de forma que solo el administrador pueda observar verdaderamente lo que sucede.

- Añadir comentarios de seguridad en el código

Es útil durante las revisiones de código concernientes a la seguridad, contar con información en forma de comentarios en las porciones de código donde se hayan tomado decisiones relativas a la seguridad.

```
// SECURITY!
//  The following assumes that the user input, in szParam,
//  has already been parsed and verified by the calling function.
HFILE hFile = CreateFile(szParam,
                         GENERIC_READ,                          FILE_SHARE_REA
D,
                         NULL,
                         OPEN_EXISTING,
                         FILE_ATTRIBUTE_NORMAL,
                         NULL);
```

```
if (hFile != INVALID_HANDLE_VALUE) {
    // Work on file.
}
```
Estos comentarios apoyarán a los revisores a la hora de comprender por qué fueron tomadas ciertas decisiones de seguridad en el momento en que fue escrito el código.

- La toma de decisiones importantes referentes a la seguridad no debe depender de los usuarios

Se debe entender, en primer lugar, que la mayoría de los usuarios finales de las distintas aplicaciones no tienen conocimientos acerca de seguridad informática, muchos no quieren ni siquiera saber, además de que no deben tener la necesidad de aprender del tema para trabajar con una aplicación segura. Los usuarios quieren que su información sea protegida sin tener ellos que tomar complejas decisiones de seguridad. La mayoría de los usuarios toman el camino más sencillo y eligen las opciones establecidas por defecto. Esta es una situación difícil de resolver en los casos en que se requiera que el usuario tome la decisión final, para estos casos se recomienda una redacción sencilla y fácil de entender para usuarios parcial o completamente ajenos a estos temas.

Tareas ya resueltas

Son varias las tareas referentes a la seguridad y a la aplicación de buenas prácticas que resuelven Symfony y RUP de manera sencilla y poderosa a la vez, por tanto, una gran parte del trabajo está resulta si durante la construcción de la propuesta de solución que se pretende ofrecer con este trabajo se utilizan de forma óptima las herramientas que brindan el marco de trabajo y la metodología de desarrollo que se utilizarán. A continuación se describe, aunque de forma resumida, algunas de estos elementos que conferirán fortaleza y robustez a este trabajo.

Gestión de usuarios y permisos

Son muchas las recomendaciones que se proponen para la implementación adecuada del manejo de usuarios y sus permisos a la hora de acceder a las diferentes funcionalidades que ofrece el sistema. El marco tiene sus propias herramientas para el manejo de estas situaciones en las que se necesita que cada usuario solamente acceda a las acciones (Las acciones son métodos con el nombre executeNombreAccion de la clase controladora nombreModuloActions que hereda de la clase sfActions (9)) para las que tiene permisos, puesto que la posibilidad de ejecutar una acción puede ser restringida a usuarios con privilegios específicos. Añadir esta seguridad a una aplicación requiere dos pasos: declarar los requerimientos de seguridad para

cada acción y autenticar a los usuarios con privilegios para que puedan acceder a estas acciones seguras. Para restringir el acceso a una acción se crea y se edita un archivo de configuración YAML[16] llamado *security.yml* en el directorio *config/* de la aplicación. En este archivo, se pueden especificar los requerimientos de seguridad que los usuarios deberán satisfacer para cada acción o para todas las acciones. Las acciones no incluyen restricciones de seguridad por defecto, así que cuando no existe el archivo *security.yml* o no se indica ninguna acción en ese archivo, todas las acciones son accesibles por todos los usuarios. Si existe un archivo *security.yml*, Symfony busca por el nombre de la acción y si existe, verifica que se satisfagan los requerimientos de seguridad (10).

Trabajando bajo estos principios se garantiza que los usuarios solamente tengan acceso a los lugares de la aplicación que sean definidos para el sistema, además de que los usuarios navegarán con un número mínimo de privilegios. Es indispensable comentar que el uso de estos recursos debe ir mezclado con las sesiones y los usuarios, para lo cual Symfony también proporciona clases que facilitan su manejo.

Módulo de administración

El sistema GINA consta de un módulo de administración que se encuentra en el plugin *suAdminPlugin*, el cual fue desarrollado bajo los principios de seguridad antes descritos; tiene como objetivo administrar el acceso de los usuarios a las diferentes funcionalidades de los sistemas que lo utilicen, y gestionar los dominios y roles que intervienen en cada una de dichas funcionalidades. Se encarga además del control de acceso a las aplicaciones y módulos del proyecto. Todas estas operaciones están apoyadas en las facilidades que brinda Symfony y en clases que se han implementado con el objetivo de adecuar el funcionamiento del plugin a los requerimientos actuales de la Aduana General de la República. Mediante este módulo se genera un menú dinámico que se crea a partir de los roles que desempeña cada usuario en un dominio específico.

Gestión segura de datos de entada

Durante el desarrollo de cualquier sistema se debe prestar especial atención a los formularios, dentro de estos a los valores por defecto, al formato, a la validación y a la recarga de los datos. Symfony ofrece, como ya se había mencionado al inicio del capítulo, una herramienta muy poderosa y cada vez más sencilla de usar, que permite una gestión segura de datos y que

[16] Acrónimo recursivo que significa "YAML Ain't Another Markup Language por sus siglas en inglés

facilita en gran medida esta tarea que es una de las más engorrosas para cualquier desarrollador de software.

Por defecto, los formularios sólo son validos si los campos rellenados por los usuarios disponen de un validador, de esta forma Symfony asegura que todos los campos tengan establecidas reglas de validación y que no pueda ser introducida información en campos que no estén incluidos en el formulario original. Para redefinir los campos que formarán parte de nuestros formularios y que podrán ser accedidos por los usuarios, los formularios cuentan con los métodos *setWidgets()* y setValidators(), mediante los cuales se pueden dejar fuera los campos que puedan representar una brecha en la seguridad si llegan a ser manipulados o modificados por un usuario malicioso o aquellos para los que no cumple objetivo que formen parte del mismo. Los campos que no tengan un validador definido y que se incluyan en el formulario forzosamente por un usuario o por error de un programador, causarán un error global y se mostrará el mensaje de error *"Extra field nombre_campo_sin_validador_asociado"*

Si no se utiliza ninguna protección, el código de la aplicación es vulnerable ya que el usuario podría enviar el formulario con todos los campos rellenos. Modificar el código HTML de los formularios para enviar cualquier información es muy sencillo gracias a herramientas como *Firebug*[17]. Además, para los campos que no tienen asociado ningún validador, su valor siempre es válido. Por lo tanto, se actualizarán todos los campos del formulario que fueron enviados por el usuario sea cual sea su valor (10).

Patrones de diseño que implementa Symfony

Algunos patrones GRASP implementados por Symfony

Creador: en las acciones se crean los objetos de las clases que representan las entidades, evidenciando de este modo que la clase *Actions* es "creador" de dichas entidades.

Experto: Symfony, en su capa de abstracción a las bases de datos en el modelo, encapsula toda la lógica de los datos y son generadas las clases con todas las funcionalidades comunes de las entidades.

Alta Cohesión: Symfony permite asignar responsabilidades con una alta cohesión, por ejemplo la clase *Actions* tiene la responsabilidad de definir las acciones para las plantillas y colabora

[17] Extensión de Firefox creada y diseñada especialmente para desarrolladores y programadores web

con otras para realizar diferentes operaciones, instanciar objetos y acceder a las propiedades, es decir, está formada por diferentes funcionalidades que se encuentran estrechamente relacionadas proporcionando que el software sea flexible frente a grandes cambios.

Front Controller: el controlador principal que recibe todas las peticiones (ubicado generalmente en el archivo index.php o como sea que se llame el punto de entrada de tu aplicación). La principal función del *Front Controller* es la de ser el único punto de entrada de peticiones, desde el cual se deriva luego al controlador específico que corresponda, según la URL[18] accedida. Se decide a que controlador derivar la petición según la URL recibida. En el caso de Symfony, las rutas definen qué URL's apuntan a qué controlador. Este es el típico punto de entrada de un patrón MVC.

Algunos patrones GOF implementados por Symfony

En la categoría Creacionales:

Singleton (Instancia única): Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. En el controlador frontal hay una llamada a *sfContext::getInstance()*. En una acción, el método *getContext()*, devuelve un objeto muy útil que guarda una referencia a todos los objetos del núcleo de Symfony y es único para la aplicación.

Abstract Factory (Fábrica abstracta): Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando. Cuando el framework necesita por ejemplo crear un nuevo objeto para una petición, busca en la definición de la factoría el nombre de la clase que se debe utilizar para esta tarea.

En la categoría Estructurales:

Decorator (Envoltorio): Añade funcionalidad a una clase, dinámicamente. El archivo *layont.php*, que también se denomina plantilla global, almacena el código HTML que es común a todas las páginas de la aplicación, para no tener que repetirlo en cada página. El contenido de la plantilla se integra en el layout, o si se mira desde el otro punto de vista, el layout decora la plantilla.

[18] Por sus siglas en inglés *Uniform Resource Locator,* es una secuencia de caracteres, de acuerdo a un formato modélico y estándar, que se usa para nombrar recursos en Internet para su localización o identificación.

Otro ejemplo en Symfony puede ser el manejo de sesiones, las cuales pueden mantener su estado en disco (la opción nativa), o en una base de datos. Guardar datos de la sesión en el cliente es otra alternativa. Estas implementaciones también son "definidas" como patrones de diseño en (11).

También, "desparramados" por todo el framework y el ORM se implementan patrones más genéricos, que son los que se utilizan en todo tipo de situaciones, siendo buenos ejemplos el Table Data Gateway, el Adapter y el Facade (los 3 muy parecidos, pero con leves diferencias), Query Object para “realizar consultas SQL sin hacerlas” entre otros no menos importantes.

Conclusiones parciales

Durante el desarrollo del capítulo se hizo un estudio del estado del arte referente a las buenas prácticas que se aplican hoy en día para el desarrollo de aplicaciones con la calidad requerida. Se está de acuerdo en que el tratamiento de la seguridad constituye uno de los aspectos más sensibles que debe ser considerado desde etapas tempranas en el ciclo de vida del desarrollo de un sistema informático, tal sería el caso de una aplicación web que será publicada en Internet y que podría revelar información importante a usuarios indebidos si no se garantiza su seguridad. En el caso que nos ocupa, la utilización del marco arquitectónico Symfony solucionará gran parte de los problemas relativos al diseño de una arquitectura segura para la aplicación, cubriendo gran parte de las buenas prácticas que se recomiendan para el diseño y la implementación de aplicaciones en general. A pesar de lo anterior es trabajo del equipo de desarrollo dominar su uso para su óptimo aprovechamiento, para que su utilización sume rasgos significativos a la aplicación en lugar de constituir una carga innecesaria. El resto es responsabilidad del desarrollador que debe ser capaz de escribir un código limpio, seguro y fácil de mantener para dotar el sistema de una calidad indiscutible.

Capítulo 2 Diseño de la solución propuesta

Introducción

En este capítulo será descrita la solución ofrecida para el problema en cuestión, haciendo un resumen de las características principales de la Arquitectura a la cual deberá ajustarse la solución informática propuesta en este trabajo, así como los rasgos particulares referentes a este tema que la caracterizarán. Serán representados además los elementos principales del diseño basados en diagramas de clases del diseño con estereotipos web con el objetivo de satisfacer los requerimientos funcionales y no funcionales establecidos para el sistema.

Desarrollo

Arquitectura del sistema Gina

La Arquitectura de Software muestra la estructura y el comportamiento de un sistema que se compone de elementos de software, expone las propiedades de esos elementos y las relaciones entre ellos. Es el más alto nivel de desglose de un sistema en sus partes (12). La solución propuesta en este trabajo está enmarcada dentro de la arquitectura actual definida para el sistema de Gestión Integral de la Aduana (GINA).

Se puede resumir la solución arquitectónica para GINA como: la utilización del marco arquitectónico Symfony para lograr obtener como resultado un estilo Modelo-Vista-Controlador, con Propel para manejar el modelo de la arquitectura mediante un Objetc-Relational-Model y PDO para manejar la abstracción de la Base de Datos. Para manejar la concepción de los paradigmas del WEB 2.0[19] y la seguridad de las aplicaciones se utiliza el marco de trabajo para Javascript ExtJs permitiendo la creación de efectos visuales y validaciones del lado del cliente reflejado esto en la Figura 1 (4). Para la comunicación entre las capas en que se divide la arquitectura se utiliza entre el cliente y el Controlador Frontal, y viceversa, tecnología JSON[20], con esta se serializan los datos pasados por el usuario en Objetos PHP. Siempre que esto sea posible la comunicación será utilizada en combinación con AJAX, para permitir mayor velocidad y dinamismo en la interacción de los sistemas por vía Web. La utilización de AJAX con

[19] La Web 2.0 es la representación de la evolución de las aplicaciones tradicionales hacia aplicaciones web enfocadas al usuario final. El Web 2.0 es una actitud y no precisamente una tecnología (4).

[20] Acrónimo de *JavaScript Object Notation* por sus siglas en inglés, es un formato ligero para el intercambio de datos.

comunicación basada en XML, se utilizará para los casos en los que sea necesario por no ser factible utilizar JSON.

Figura 1 Integración de los Componentes y Tecnologías en la Solución Arquitectónica.

La comunicación entre las capas del Controlador y el Modelo, gracias a la implementación del ORM, está dada por el envío de objetos que contienen la información necesaria para manejar las peticiones del usuario o las acciones a realizar. En caso de que los arreglos de datos a utilizar contengan múltiples registros serán manejados mediante arreglos que contienen los Objetos referentes a las clases del Modelo. Para manejar la comunicación con la Base de Datos se utiliza PDO (actualizado en (13)), librería que utiliza Propel para la abstracción de la BD. La comunicación interna entre los distintos componentes de la aplicación se llevará a cabo mediante los componentes del núcleo de Symfony para manejar los flujos de datos, los cuales permiten el acceso a la información desde los distintos puntos del código de la aplicación, y tienen distintas formas de acceso desde cada uno de ellos para evitar violaciones del acceso al código.

Tecnologías a utilizar para el desarrollo de la aplicación

La dirección proyecto estableció, para los equipos de trabajo inmersos en el desarrollo de los subsistemas de GINA, que deberán desarrollar las aplicaciones haciendo uso de las siguientes tecnologías: marco de trabajo arquitectónico Symfony v1.2.X lo que implica el uso del lenguaje de programación embebido PHP[21] v5.2 o mayor para la programación de las páginas del lado del servidor; paradigma POO o Programación Orientada a Objetos por sus siglas en español;

[21] PHP: Hypertext Preprocessor por sus siglas en inglés.

metodología de desarrollo de software RUP; marco de trabajo para Java Script ExtJs v3.0 para la implementación de las interfaces de interacción con el usuario; para la gestión de las bases de datos Oracle v11g con cierta ambición de migrar en algún momento a una tecnología libre para esta gestión; como herramienta CASE [22] Visual Paradigm v6.4 y como entorno de desarrollo integrado para escribir el código de la aplicación Eclipse PDT v3.4 o superior.

Rasgos particulares de la arquitectura del subsistema para el despacho de MTI aéreos (Vista del diseño).

Como se ha mencionado en otras secciones del documento, el subsistema para el despacho de medios de transporte internacional aéreos formará parte del sistema para la gestión integral de procesos aduanales, GINA, integrándose y nutriéndose de muchas de las aplicaciones y plugins que contiene, para ofrecer una solución completa y eficiente a las necesidades de la AGR. El sistema coexistirá como un módulo nombrado *aeronaves*, que se integrará a una aplicación denominada MTI, creada para informatizar todos los procesos relacionados con los medios de trasporte internacional que entran y salen de Cuba. La aplicación de MTI constituye un plugin más de GINA, llamado *sfMtiPlugin*, con el objetivo de ofrecer servicios a otras aplicaciones que así lo requieran, tal y como se ilustra en la siguiente imagen.

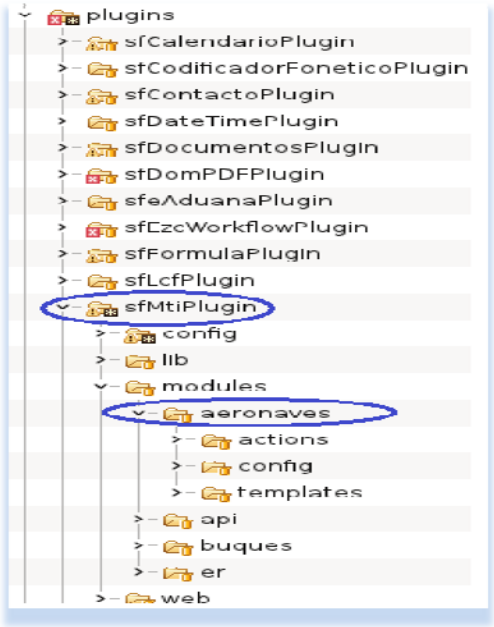

Figura 2 Integración del módulo *aeronaves* al plugin sfMtiPlugin

[22] **C**omputer **A**ided **S**oftware **E**ngineering por sus siglas en inglés o Ingeniería de software asistida por computadora.

El sistema compartirá el modelo con los sistemas para el despacho de MTI marítimos, como es el caso de las Embarcaciones de Recreo y Buques, más adelante en el capítulo se abordará en estas especificaciones.

Para llevar a cabo las actividades relativas al despacho con los representantes de las aeronaves, se necesita contar con la información adelantada que se recibe de los pasajeros y de las cargas de las mismas, esta información se almacena y se recupera con sistemas informáticos de producción nacional. Para el caso de la información adelantada de pasajeros, se encuentra en explotación una herramienta que pertenece también a *sfMtiPlugin* nombrada API, que procesa ficheros con el mismo nombre, que son recibidos desde las diferentes aerolíneas con la información requerida, para satisfacer las necesidades reales de los trabajadores de la AGR, el módulo para aeronaves se debe comunicar eficazmente con este y otros sistemas de los cuales depende parcial o totalmente para su ejecución.

En la siguiente figura se muestra un diagrama de paquetes[23] que contiene los componentes en forma de paquetes o subsistemas con los cuales interactúa la aplicación, los mismos se describen posteriormente.

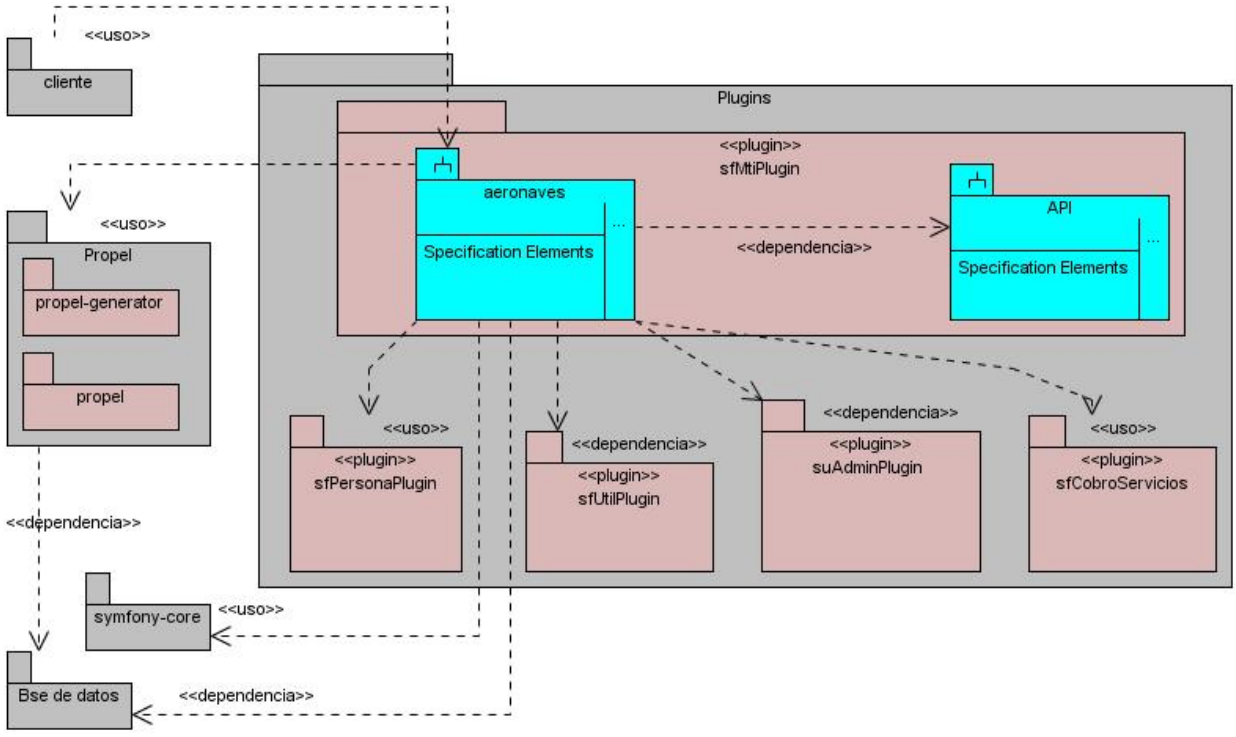

Figura 3 Diagrama de interacción entre paquetes.

[23] Un *diagrama de paquetes* muestra cómo un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones.

Paquetes

- *sfPersonaPlugin*: utlizado para unificar la información referente a las personas que están involucradas de una forma u otra con los procesos aduanales. Se utilizará para almacenar los datos del capitán de la aeronave.
- *sfUtilPlugin*: contiene un módulo para la centralización de nomencladores definidos por la aduana. Se utilizará para nomenclar datos.
- *suAdminPlugin*: se utilizará para la gestión de usuarios y permisos. Descrito en la introducción del documento. Se utilizará además para enviar datos de los eventos ocurridos en el sistema, el mismo los almacenará en forma de trazas.
- *sfCobroServicios*: plugin para poner al cobro los servicios prestados, para este caso el despacho de entrada.
- *API*: sistema para la recepción y el procesamiento de la información adelantada de pasajeros, el mismo será consultado para la identificación de arribos de aeronaves en un rango determinado.
- *Propel*: marco de trabajo utilizado para la generación de las clases del modelo y para la realización de consultas sin la necesidad de crear ni una única sentencia SQL.
- *Symfony*: marco de trabajo arquitectónico.
- *Base de datos*: representa al gestor y a la propia base de datos, para este caso Oracle, imprescindible para el funcionamiento de la aplicación.

La forma de comunicación entre estos componentes se llevará a cabo utilizando servicios que ofrecerán los mismos, de manera que se garantice que no existan violaciones en la seguridad de ninguno de los relacionados en la figura.

Partiendo de la implementación que realiza Symfony del patrón arquitectónico MVC para los sistemas que se desarrollan bajo el mismo, la siguiente figura muestra un diagrama que contiene los paquetes fundamentales que serán propios del módulo aeronaves.

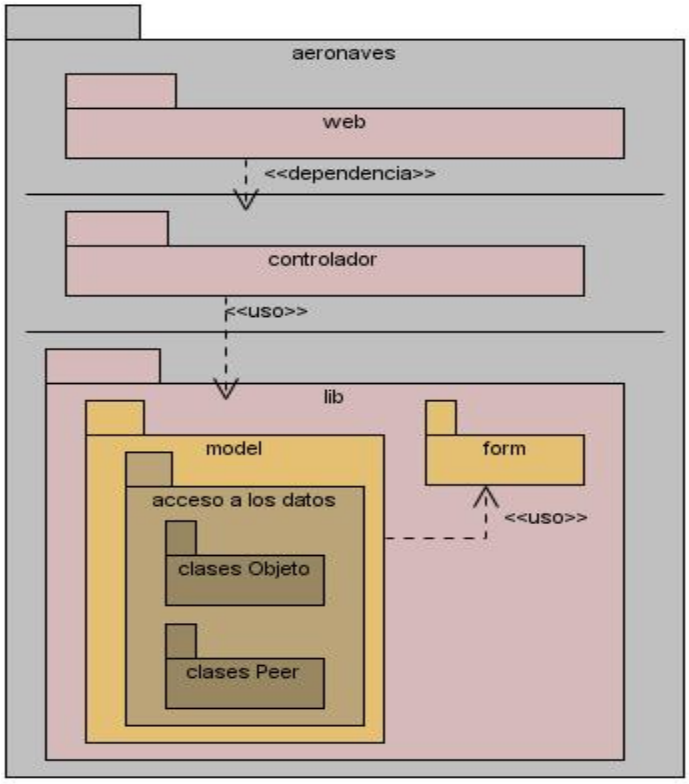

Figura 4 Diagrama de paquetes del módulo aeronaves.

Diagrama de despliegue de la aplicación

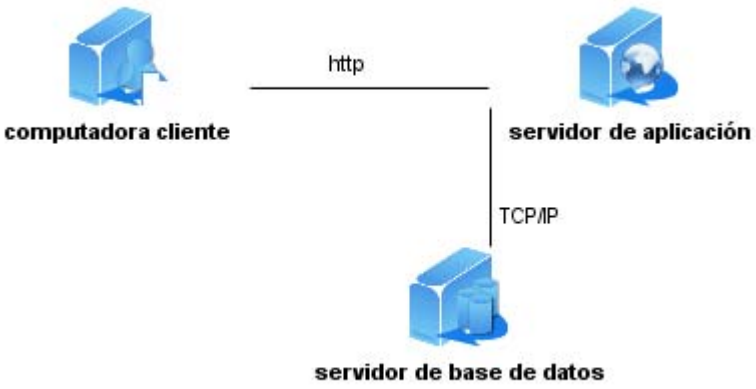

Figura 5 Diagrama de despliegue

Catálogo de requisitos

En esta actividad se especifican los requisitos funcionales y no funcionales que deben ser satisfechos al final del trabajo por el módulo para el despacho de MTI aéreos, para los cuales se definió además sus niveles de prioridad.
A continuación se muestran los niveles de prioridad por medio de flechas de los requisitos del sistema:
↑: Prioridad alta en el sistema.
→: Prioridad media en el sistema.
↓: Prioridad baja en el sistema.

Requerimientos funcionales de la aplicación

↑Confirmar Arribos/Salidas internacionales de aeronaves:

Para el caso de la confirmación del arribo de una aeronave, el sistema debe extraer de la información adelantada de pasajeros y de la información adelantada de cargas los vuelos que están por confirmar dentro de un rango de fechas dado por el usuario y dar la posibilidad de introducir la información de un vuelo del que no se tenga información adelantada; después de confirmado el arribo debe mostrarse en otro listado de vuelos confirmados sin despacho de entrada para darle la posibilidad al usuario de cancelar ese arribo. Para el caso de la confirmación de las salidas de las aeronaves es similar, el sistema debe mostrar un listado de vuelos a los que ya se les ha hecho un despacho de salida previo y debe dar la posibilidad de confirmar su salida, mostrándolos luego en un listado de salidas confirmadas para dar la posibilidad de cancelarlos.

Para facilitar el diseño de este requisito se decidió dividirlo en dos escenarios fundamentales:

- Confirmar Arribos/Salidas de aeronaves.
- Cancelar confirmación de Arribos/Salidas de aeronaves.

↑Gestionar Despacho de aeronaves:

Para el caso de la gestión de los despachos de entrada de aeronaves el sistema debe mostrar un listado de los arribos que han sido confirmados en un rango de fechas dado por el usuario y dar la posibilidad de registrar el despacho correspondiente a un vuelo seleccionado por el usuario. El sistema debe dar también un listado de los despachos de entrada realizados que aun no tienen despacho de salida, y dar la posibilidad de modificarlos o cancelarlos, esto último si el servicio no ha sido cobrado por el sistema de recaudación. En el caso de la gestión de los

despachos de salida, el sistema debe listar los vuelos con despacho de entrada realizados en un rango de fechas dado por el usuario, y dar la posibilidad de registrar un despacho de salida al vuelo seleccionado por el usuario, además debe dar la posibilidad de registrar los datos de un vuelo que haya tenido una entrada nacional y que por tanto no fue despachado a la entrada, y debe permitir realizarle un despacho de salida al mismo.

Para su facilitar el diseño de este requisito se decidió dividirlo en dos escenarios fundamentales:

- Gestionar despacho de aeronaves
    - Registrar despacho de entrada
    - Registrar despacho de salida
    - Modificar despacho de entrada
    - Modificar despacho de salida
- Cancelar despachos de entrada y salida de aeronaves.

↑Otorgar número de manifiesto:

El sistema debe otorgar un número consecutivo a cada uno de los viajes que se registren en el sistema. En el caso de los viajes que envían información adelantada de cargas, desde el momento en que son registrados en el sistema debe asignársele un número de manifiesto, para el resto de los viajes solo se les otorga en el momento en que es confirmado su arribo para el caso de las entradas nacionales con salida internacional, se les otorga en el momento en que se registra su despacho de salida.

→Poner al cobro los servicios.

El sistema debe enviar los datos requeridos de los despachos que registre y enviarlos para el sistema de recaudación, para que sea cobrado el servicio.

→Registrar notificación de infracción.

Requerimientos no funcionales de la aplicación

La distribución geográfica de la entidad a informatizar comprende una extensión nacional y tiene un ancho de banda en sus peores tramos de 64Kb en estos momentos, aunque están pendientes algunas inversiones para mejorar la velocidad de transmisión de datos. Por lo que los objetivos trazados comprenden la rapidez de la transmisión de los datos con la creación de

páginas de poco peso, además de garantizar que la aplicación pueda ser accedida desde cualquier lugar del país. Así como la implementación de medidas de seguridad que permitan que la aplicación sea confiable y que los datos viajen por la red de la forma más segura posible, ya que estos viajarán por medios inseguros a nivel nacional (14).

Diseño de la aplicación.

Extensiones para el diseño Web

Estereotipos	Imagen	Descripción
<<Server page>>		Representa una página Web dinámica que contiene el código ensamblado por el servidor cada vez que se solicita.
<<Client page>>		Representa una instancia de una página cliente. Es una página Web con formato HTML.
<<Form>>		Simboliza un formulario, es el elemento encargado de realizar envíos a las páginas servidoras.

Tabla 1 Extensiones para el diseño web.

Relaciones que se establecen entre las clases que conforman la extensión UML para Web.

Hasta Desde	Server page	Client page	Form
<<server page>>	<<Redirect>>	<<Build>>, <<Redirect>>	--
<<client page>>	<<Link>>, <<Redirect>>	<<Link>> , <<Redirect>>	Contiene
<<Form>>	<<Submit>>	Agregado por.	--

Tabla 2 Relaciones entre clases que conforman la extensión UML para WEB

Requisito: Confirmar Arribos/Salidas internacionales de aeronaves.

Durante el diseño de este requisito, se definió la creación de la clase *MtiViaje*, esta será utilizada igualmente por los subsistemas para el despacho de Buques y de las Embarcaciones

de Recreo, la misma "*agrega*[24]" un "MTI", esta clase, que describe un viaje[25] para el sistema, persistirá en el modelo de datos y almacenará los datos relativos a los viajes, de forma tal que para un único viaje se tendrán dos tuplas[26] en la base de datos, las cuales se podrán identificar como pertenecientes a un único viaje por el numero *número de manifiesto*[27]. Debido a que es una clase que será utilizada por otros subsistemas se decidió además crear una clase especializada *MtiAeronaveViaje* que hereda las funcionalidades y atributos de la primera que no persiste en el modelo de datos de la aplicación. Para guardar un historial de lo ocurrido a un viaje, se creó la clase *MtiSucesos*, la cual esta "*compuesta*[28]" por un *MtiViaje* y que posee los atributos necesarios para realizar ella sola esta función. Para llevar a cabo este requisito el sistema ofrecerá al usuario una única interfaz que contendrá un formulario con los datos precisos para llevar a cabo la confirmación y la cancelación del arribo y la salida de una aeronave. El diagrama de la Figura 6 muestra estas y otras clases imprescindibles para llevar a cabo el requisito por la aplicación, la Figura 7 y Figura 8 muestran el flujo de datos entre objetos que ocurre en la aplicación para ejecutar las diferentes funcionalidades del requisito.

[24] Derivado de "Agregación" relación entre clases que denota que una posee una colección de la otra. Donde la cardinalidad mínima puede ser 0.

[25] Representa lo equivalente a un viaje en el negocio que se maneja, que no es más que la llegada de una aeronave desde un aeropuerto origen a un aeropuerto destino y la salida de la propia aeronave hacia otro destino. Siempre teniendo en cuenta que solo se controlan arribos y salidas internacionales.

[26] En la teoría de bases de datos, una tupla se define como una función finita que mapea (asocia unívocamente) los nombres con algunos valores.

[27] Número consecutivo otorgado por la aduana a los viajes internacionales que son controlados en su sede.

[28] Derivada de "Composición" relación entre clases que denota que una posee una referencia de la otra. Donde la cardinalidad mínima tiene que ser 1.

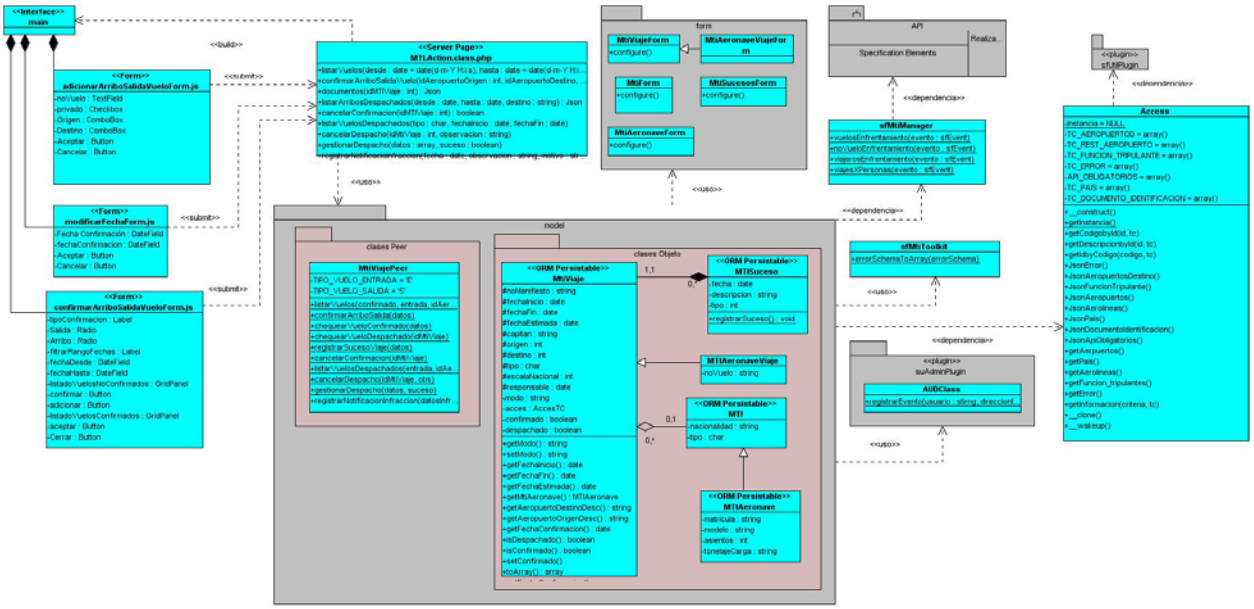

Figura 6 Diagrama de clases del diseño para el requisito funcional "Confirmar Arribos/Salidas internacionales de aeronaves."

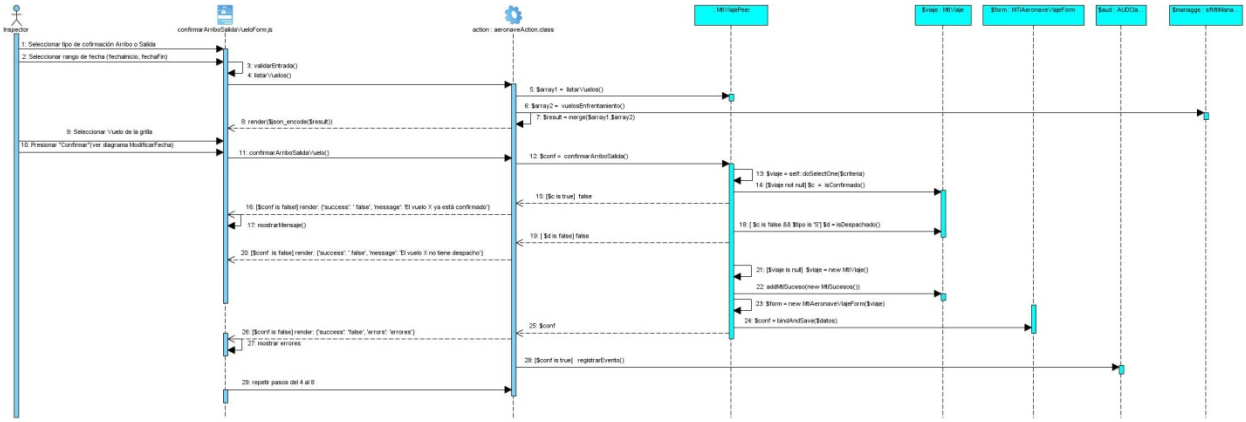

Figura 7 Diagrama de secuencia para el escenario "Confirmar Arribos/Salidas de aeronaves".

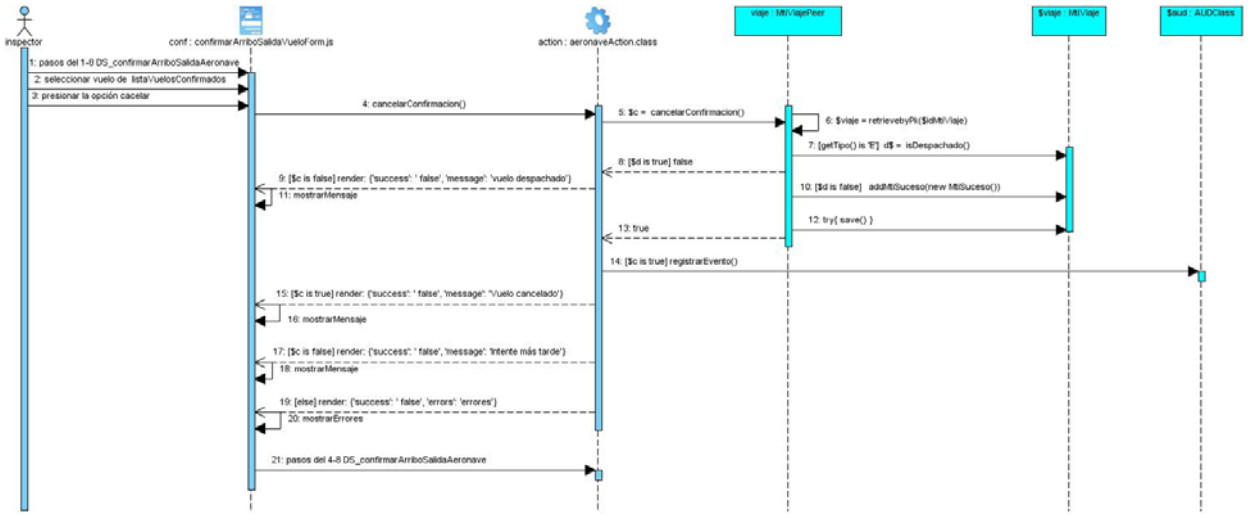

Figura 8 Diagrama de secuencia para el escenario "Cancelar confirmación de Arribos/Salidas de aeronaves".

Requisito: Gestionar Despacho de aeronaves

En el diseño de este requisito se decidió crear la clase *MtiDespacho,* también compartida por los subsistemas para el despacho de Buque y Embarcaciones de Recreo, esta clase, describe un *despacho*, que está *compuesto* además por documentos (*MtiDespachoDocumentos*), estas clases son persistentes en el modelo de datos de la aplicación, y almacenarán los datos inherentes al registro y la modificación de despachos de entrada y salida de los *viajes* de aeronaves. En este requisito participan también las clases *Mti* [29] y *MtiAeronave* debido al significado que tiene conocer el MTI en el que se realiza el *viaje* del *despacho* en cuestión.

[29] Representa el medio de trasporte internacional (aeronave en este caso) en el que es realizado el viaje.

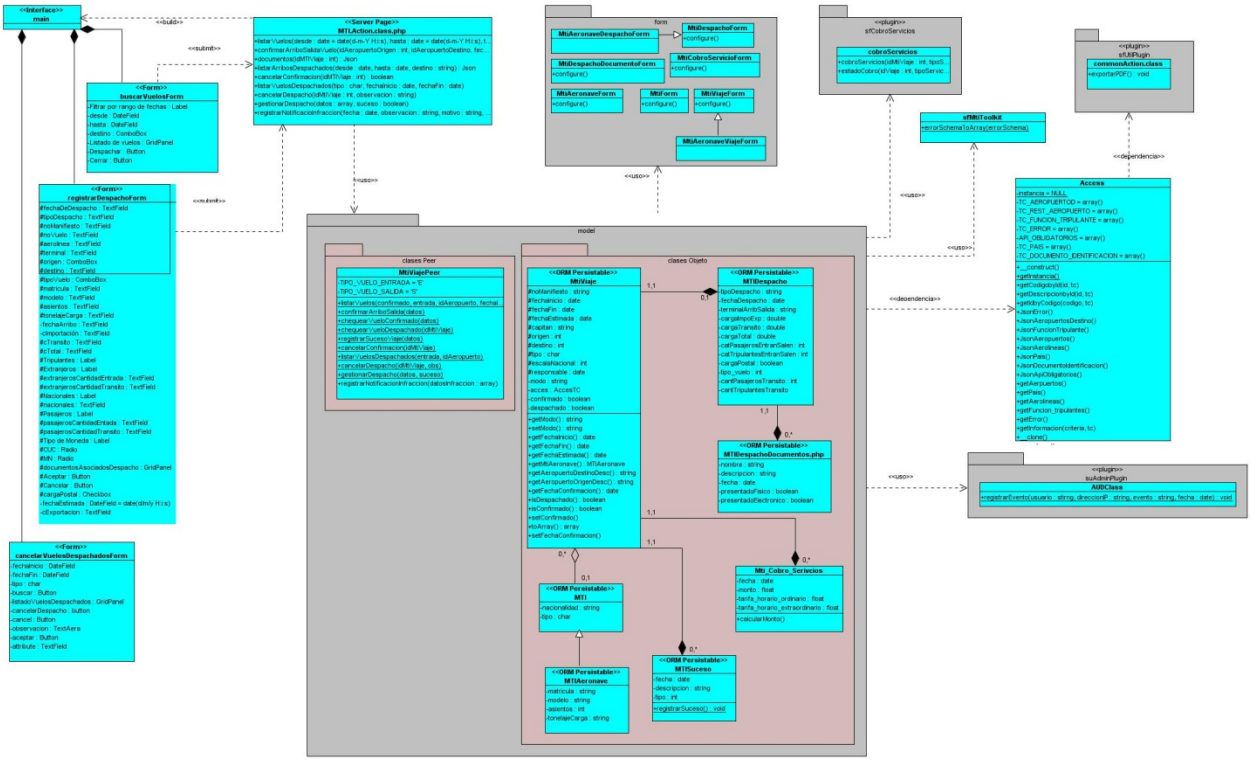

Figura 9 Diagrama de clases del diseño del requisito "Gestionar Despacho de aeronaves."

Para llevar a cabo estas funcionalidades, la aplicación ofrecerá las interfaces necesarias para la interacción con el usuario. El diagrama de la Figura 9 muestra las clases mencionadas y otras necesarias para ejecutar las funcionalidades del requisito.

Los siguientes diagramas muestran el flujo de datos entre objetos que ocurre en la aplicación para ejecutar las diferentes funcionalidades del requisito.

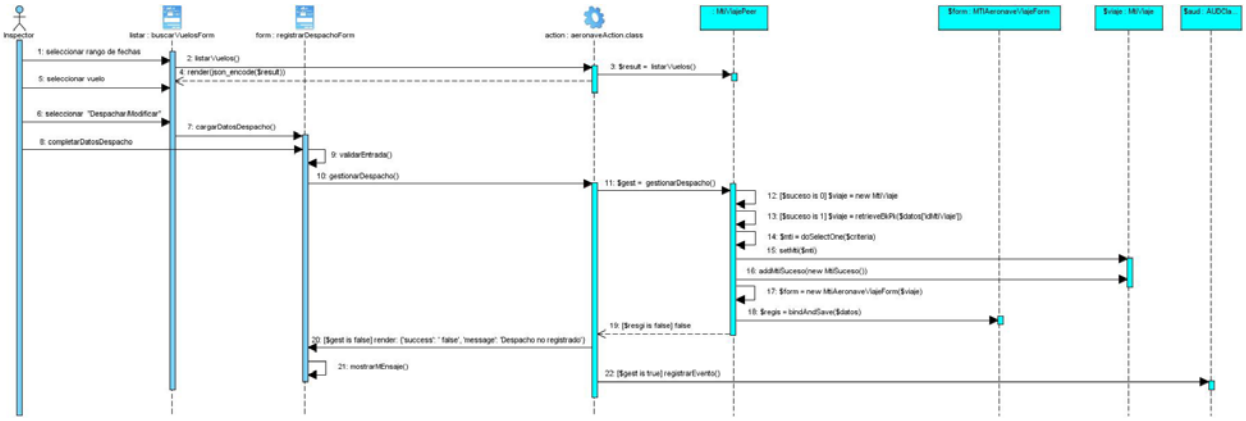

Figura 10 Diagrama de secuencia para el escenario "Gestionar Despacho de aeronaves".

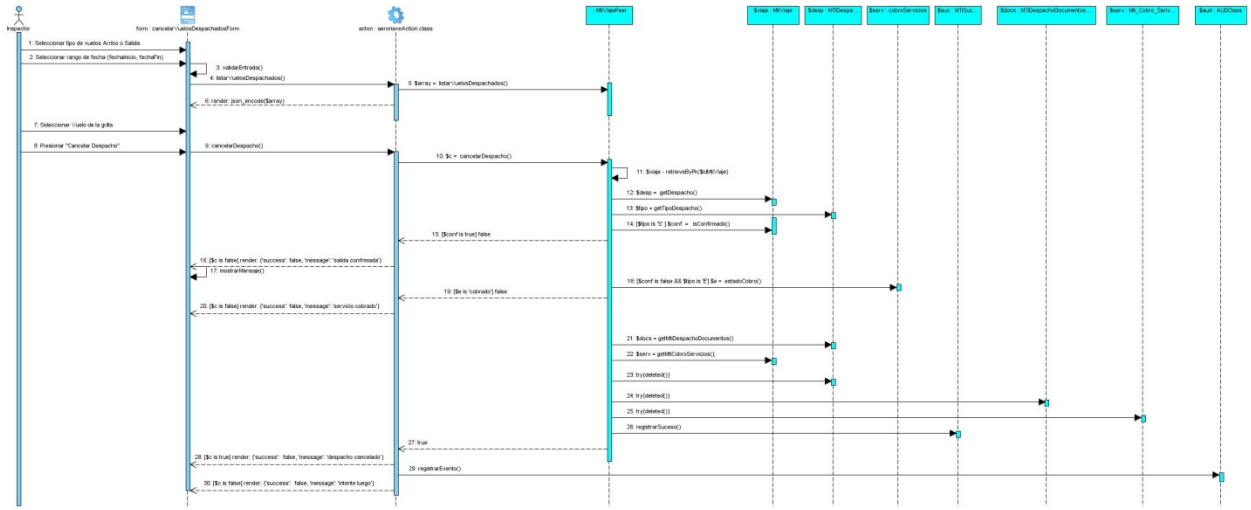

Figura 11 Diagrama de secuencia para el escenario "Cancelar Despacho de aeronaves".

Es válido aclarar que para la realización del escenario “Gestionar Despacho de aeronaves”, el cual contiene los subescenarios para el registro y la modificación de los despachos de entrada y salida, se definió el uso de los formularios embebidos de Symfony (ver sección Formularios), de manera que si se embebe a un objeto de *MtiAeronaveViajeForm* un objeto de: *MtiAeronaveDespachoForm*, *MtiForm*, *MtiAeronaveForm, y de MtiCobroServicioForm* (en el caso del registro o modificación de un despacho de entrada) y al objeto *MtiAeronaveDespachoForm un objeto de MtiDespachoDocumentosForm* se obtiene un mecanismo que es capaz de registrar y modificar un despacho mediante una única funcionalidad (*gestionarDespacho* de la clase *MtiViajePeer*, ver Figura 9) que será la encargada de crear el objeto de *MtiAeronaveViajeForm*, validar el formato y tamaño de los datos y salvar el formulario en la base de datos mediante la llamada al método *bindAndSave* de la clase *MtiAeronaveViajeForm.*

Requisito: Registrar notificación de infracción

Durante el diseño de este requisito se decidió crear la clase *MtiMedida,* esta clase también compartida por los subsistemas para el despacho de Buque y Embarcaciones de Recreo, es la encargada de describir las infracciones que son cometidas durante un viaje. Esta clase está compuesta por el viaje en el que se cometió la infracción, y es persistente en la base de datos. En el diagrama de la Figura 12 se muestras las clases que intervienen en la realización de las funcionalidades del requisito, en la Figura 13 se muestra el flujo de datos entre objetos que debe ocurrir para llevar a cabo el requisito.

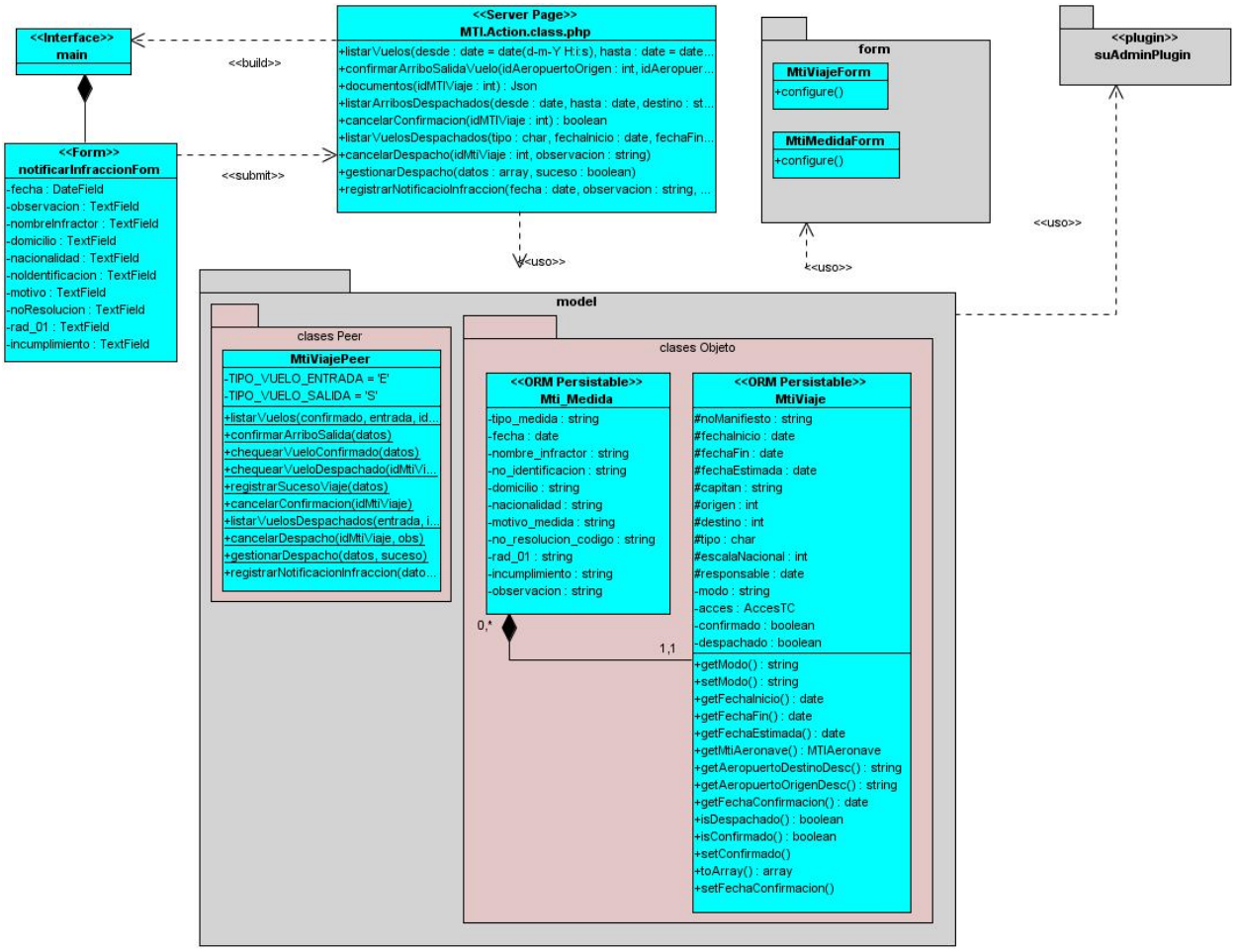

Figura 12 Diagrama de clases del diseño del requisito "Registrar notificación de infracción"

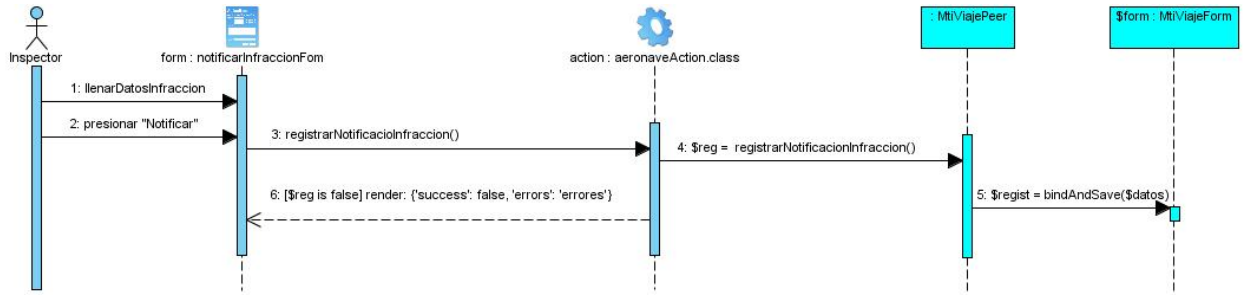

Figura 13 Diagrama de secuencia del requisito "Registrar notificación de infracción".

Requisito: Poner al cobro los servicios

Durante el registro o modificación de un despacho realizado a un viaje de entrada se debe poner al cobro un servicio (el arribo del viaje al aeropuerto) en el sistema para la recaudación de fondos de GINA; para llevar a cabo esta operación durante el diseño se definió que un servicio será embebido a un viaje durante la funcionalidad “registrar despacho de entrada”, de

manera que se tenga evidencia de los servicios ofrecidos por viajes. Usando los formularios embebidos se garantiza que cuando se modifique un despacho de entrada, si existen cambios en el mismo que afectan los datos del servicio correspondiente, el mismo será modificado también (esto último teniendo en cuenta que si el servicio ya fue cobrado no puede ser modificado el despacho).

Por otro lado la cancelación de un despacho de entrada, que implica la cancelación del servicio correspondiente, solo se puede realizar si el servicio no ha sido cobrado (ver Figura 11).

Requisito: Otorgar número de manifiesto

Para otorgar el número de manifiesto antes descrito en la descripción del requisito (ver Otorgar número de manifiesto) se consultará una función ya implementada que se encarga de generar el número consecutivo al último otorgado.

Diagrama de clases para manejar la lógica del negocio. Descripción.

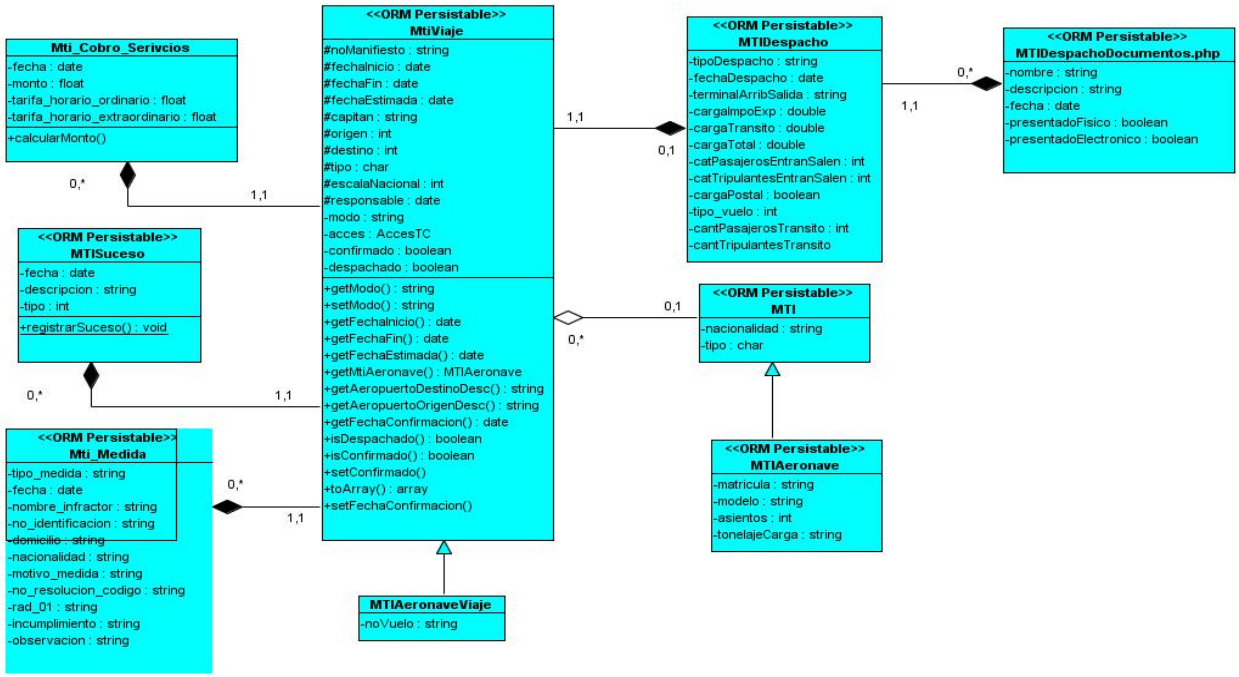

Figura 14 Diagrama de clases de la lógica del negocio (clases objeto)

Nombre	*Mti <<persistente>>*
Descripción	Su misión es describir los MTI de forma general. Constituye una generalización en términos de relaciones entre clases.

Atributo	Tipo	Descripción
tipo	String	Definirá el tipo de MTI que representa.
nacionalidad	String	Definirá el país del que proviene la aeronave.

Tabla 3 Descripción de la clase Mti.

Nombre	*MtiAeronave <<persistente>>*	
Descripción	Su misión es describir los MTI aéreos. Constituye una especialización de la clase *Mti* en términos de relaciones entre clases.	
Atributo	**Tipo**	**Descripción**
matrícula	String	Representa una cadena de texto otorgada a cada aeronave, por tanto tendrá un valor único para cada instancia de esta clase.
modelo	String	Cadena de texto que representa el modelo de la aeronave.
asientos	Int	Representa el número de asientos de la aeronave.
tonelajeCarga	String	Representa la cantidad de carga que puede transportar la aeronave, en toneladas.

Tabla 4 Descripción de la clase MtiAeronave.

Nombre	*MtiViaje <<persistente>>*	
Descripción	Su misión es describir un viaje de un MTI. Un viaje no es más que la llegada de un MTI desde un origen a un destino y la salida del propio MTI hacia otro destino, este último hecho hace que dos instancias de esta clase representen un viaje para el negocio que se maneja. Siempre teniendo en cuenta que solo se controlan arribos y salidas internacionales. Constituye una generalización en términos de relaciones entre clases.	
Atributo	**Tipo**	**Descripción**
noManifiesto	String	Número consecutivo otorgado por la aduana a cada viaje que controla. Un viaje del negocio (dos instancias de *MtiViaje* para el sistema) tiene un único número de manifiesto.
fechaInicio	Date	Fecha de la salida oficial de una aeronave de un aeropuerto. Representa la fecha de confirmación para el caso de los viajes de entrada.
fechaFin	Date	Fecha de llegada oficial de una aeronave a un aeropuerto. Representa la fecha de confirmación para el caso de los viajes de salida.
fechaEstimada	Date	Representa la fecha estimada de llegada de una aeronave a un aeropuerto para el caso de los viajes de entrada o la fecha estimada de salida de una aeronave de un aeropuerto para el caso de los viajes de salida.
tipo	Char	Carácter que tendrá valor 'E' para el caso de los vuelos de entrada y 'S' para el caso de los vuelos de salida.
origen	String	Representa el aeropuerto origen del vuelo.
destino	String	Representa el aeropuerto destino del vuelo.
capitán	String	Representa los datos del nombre del capitán.
responsable	String	Representa la aerolínea responsable del vuelo.
escalaNacional	String	Aeropuerto nacional en el cual el vuelo ha hecho o hará una escala.

Tabla 5 Descripción de la clase *MtiViaje.*

Nombre	*MtiAeronaveViaje*	
Descripción	Su misión es describir un viaje de una aeronave. Es una especialización en términos de relaciones entre clases.	
Atributo	**Tipo**	**Descripción**
noVuelo	String	Representa el número del vuelo e identifica un viaje unido al destino y la fecha de entrada o salida.

Tabla 6 Descripción de la clase *MtiAeronaveViaje.*

Nombre	*MtiDespacho <<persistente>>*	
Descripción	Su misión es describir un despacho realizado a la entrada o salida de un vuelo.	
Atributo	**Tipo**	**Descripción**
tipoDespacho	Char	Carácter que tendrá valor 'E' para el caso de un despacho realizado a un vuelo de entrada, y valor 'S' para el caso de un despacho realizado a vuelo de salida.
fechaHora	Date	Fecha en que se realizó el despacho.
terminal	Int	Representa la terminal en que se realizó el despacho.
cargaImpoExpo	String	Representa la cantidad de carga de exportación para el caso de los viajes de salida e importación para el caso de los vuelos de entrada (Toneladas).
cargaTransito	String	Representa la cantidad de carga en tránsito que transporta la aeronave (Toneladas).
cargaTotal	String	Representa la carga total que transporta la aeronave (Toneladas).
cargaPostal	Boolean	Representa si el vuelo trae carga postal o no.
pasajerosEntranSalen	Int	Representa el número de pasajeros que entra o sale en el vuelo.
tripulantesEntranSalen	Int	Representa el número de tripulantes que entra o sale en el vuelo.
pasajerosTransito	Int	Representa el número de pasajeros en tránsito que transporta el vuelo.
tripulantesTransito	Int	Representa el número de tripulantes en tránsito que transporta el vuelo.
tipoVuelo	String	Representa el tipo de viaje en dependencia de lo que transporte (mixto, pasajeros, carga, otros…).

Tabla 7 Descripción de la clase *MtiDespacho.*

Nombre	*MtiDespachoDocumentos <<persistente>>*	
Descripción	Su misión es describir los documentos que son presentados en un despacho.	
Atributo	**Tipo**	**Descripción**
nombre	String	Representa el nombre del documento.
descripcion	String	Representa una descripción del documento.
fecha	Date	Representa la fecha en que se presentó el documento.
presentadoFisico	Boolean	Representa si el documento fue presentado físico en

		el momento del despacho.
presentadoElectronico	Boolean	Representa si el documento fue presentado electrónicamente previo al despacho.

Tabla 8 Descripción de la clase *MtiDespachoDocumentos.*

Nombre	*MtiCobroServicios <<persistente>>*	
Descripción	Su misión es describir los servicios que son ofrecidos durante los despachos.	
Atributo	**Tipo**	**Descripción**
fecha	Date	Representa la fecha en que se ofreció el servicio.
monto	Float	Representa el monto que debe ser cobrado por el servicio ofrecido.
HorarioOrdinario	Boolean	Representa si el servicio se ofreció en horario laborable.
HorarioExtraordinario	Date	Representa si el servicio se ofreció fuera del horario laborable..

Tabla 9 Descripción de la clase *MtiCobroServicios*

Nombre	*MtiSucesos <<persistente>>*	
Descripción	Su misión es describir los diferentes eventos que ocurren a un viaje.	
Atributo	**Tipo**	**Descripción**
fecha	Date	Representa la fecha en que se ocurrió el suceso.
tipo	String	Representa el tipo de suceso.
descripcion	Text	Representa una observación que pueda ser almacenada referente al suceso.

Tabla 10 Descripción de la clase *MtiSucesos.*

Nombre	*MtiMedida <<persistente>>*	
Descripción	Su misión es describir las diferentes violaciones que se cometen durante un viaje por parte de los responsables del mismo.	
Atributo	**Tipo**	**Descripción**
fecha	Date	Representa la fecha en que fue aplicada la medida
Nacionalidad	String	Representa la nacionalidad del infractor
Incumplimiento	String	Representa el tipo de infracción que se cometió.
Observacion	String	Representa una observación que se necesite almacenar acerca de la infracción.

Tabla 11 Descripción de la clase *MtiMedida.*

Estructura del paquete de objetos peer

La Figura 15 muestra la estructura del paquete de clases peer del modelo (ver Figura 4), específicamente las clases personalizadas durante el diseño de la aplicación.

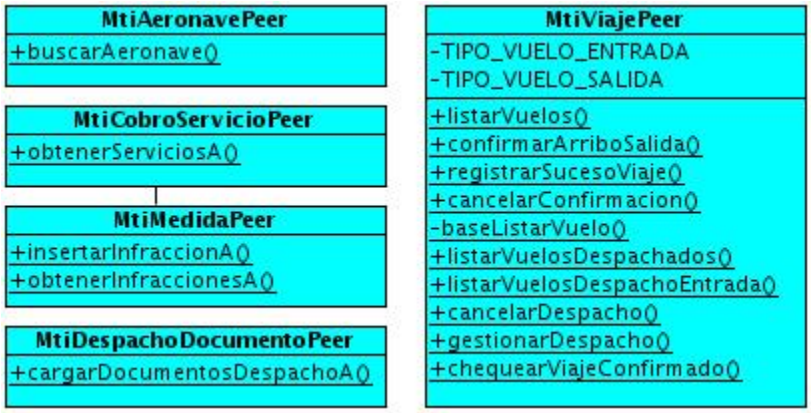

Figura 15 Diagrama de clases del paquete de clases peer.

Formularios

Como ya se abordó en el primer capítulo del presente documento, para la validación y el almacenamiento de los datos de entrada a la aplicación, se hará uso del marco de trabajo para formularios que facilita Symfony para esta gestión. El mismo para cada una de las diferentes clases objetos, que manejan la lógica del negocio, y las relaciones entre ellas, crea dos clases que heredan de la clase *BaseFormPropel.class* que hereda de *sfFormPropel.class* que hereda de *sfForm.class,* las cuales brindan las funcionalidades y herramientas necesarias para modificar las propiedades de los objetos. Este marco de trabajo para formularios provee además *validadores* que encapsulan todo el código necesario para validar los datos introducidos por el usuario, lo cual facilita la tarea de conservar la integridad de los datos (5). El diagrama de clases de la Figura 16 muestra el paquete *form* que contiene el par de clases generadas por cada una de las clases propias de la lógica del negocio de la aplicación, las mismas ya han sido referenciadas anteriormente en los diferentes diagramas de clases de diseño y diagramas de secuencia.

El marco de trabajo para formularios da la posibilidad de incluir ciertos objetos *form a otros objetos *form mediante la función que aparece a continuación:

```
$this->embedForm('embebed', new embebedForm());
```

Esto no es más que una herramienta que facilita en gran medida el almacenamiento de formularios como si fueran objetos y que resulta de gran utilidad para manejar de una forma cómoda y segura las relaciones de composición y agregación entre clases. Gracias a los formularios embebidos la gestión de los despachos de entrada y salida (registrar y modificar) se reduce a una única llamada al método *bindAndSave()* de la clase *MtiAeronaveViajeForm* a la cual se le embeben el resto de los objetos *Form que son necesarios para la gestión de los despachos.

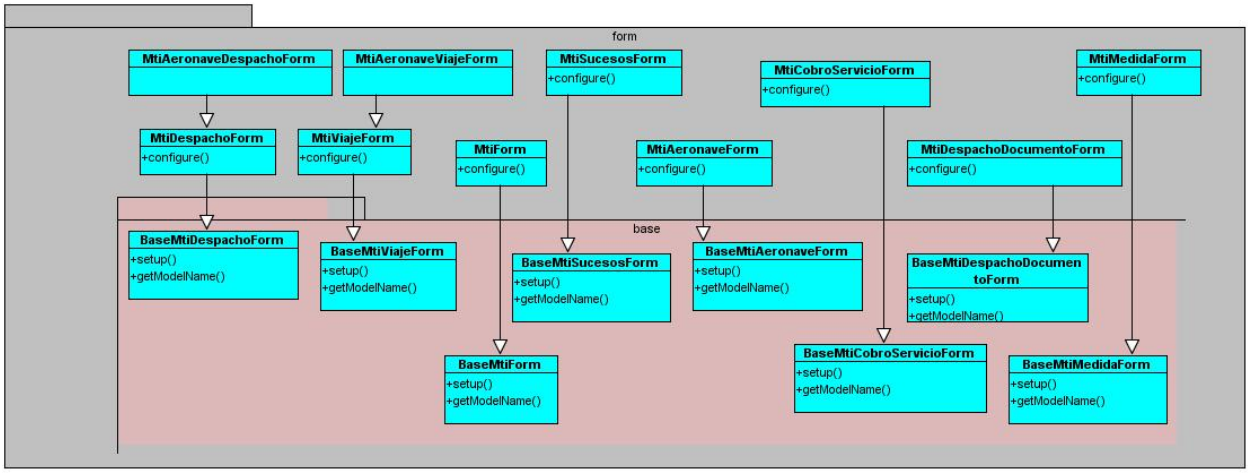

Figura 16 Diagrama de clases del paquete form.

Patrones de diseño implementados en la solución

Experto:

Este patrón consiste en asignar una responsabilidad a una clase que cuenta con la información necesaria para cumplirla. La implementación de este patrón se evidencia en la realización de la clase *MtiCobroServicio,* pues la misma es la encargada de calcular el monto a cobrarse por el servicio ofrecido, auxiliándose de la información que esta conoce del viaje y el despacho.

Instancia única:

El subsistema para el despacho de medios de transporte aéreos necesita consultar de forma reiterada cientos de datos que son gestionados por el plugin *sfUtilPlugin*, con el objetivo de agilizar estas consultas se realizó la implementación de la clase *Acces* mediante el patrón *Singlenton*, esta clase carga en memoria un grupo de arreglos con los datos más usados que son útiles para el despacho (y para el resto de los subsistemas de sfMtiPlugin), de manera que

sean recuperados una única vez y puedan ser consultados desde todos los subsistemas, disminuyendo el tiempo de ejecución de las distintas funcionalidades.

Alta cohesión

Cada elemento de nuestro diseño debe realizar una labor única dentro del sistema, la implementación de ciertas responsabilidades en las clases *Peer que puedan ser reutilizadas es un ejemplo de alta cohesión. La incorporación de funciones reutilizables por los distintos subsistemas de la aplicación a las clases *sfMtiToolkit* y *Acces* también es una evidencia de alta cohesión en la solución propuesta.

Modelo de datos de la aplicación

El modelo de datos de la Figura 17, fue creado a partir de las clases que fueron definidas como persistentes durante las anteriores actividades del diseño. Este diagrama permite describir tanto las estructuras de datos (el tipo de los datos y la forma en que se relacionan) como las restricciones de integridad (condiciones que los datos deben cumplir para reflejar correctamente la realidad deseada).

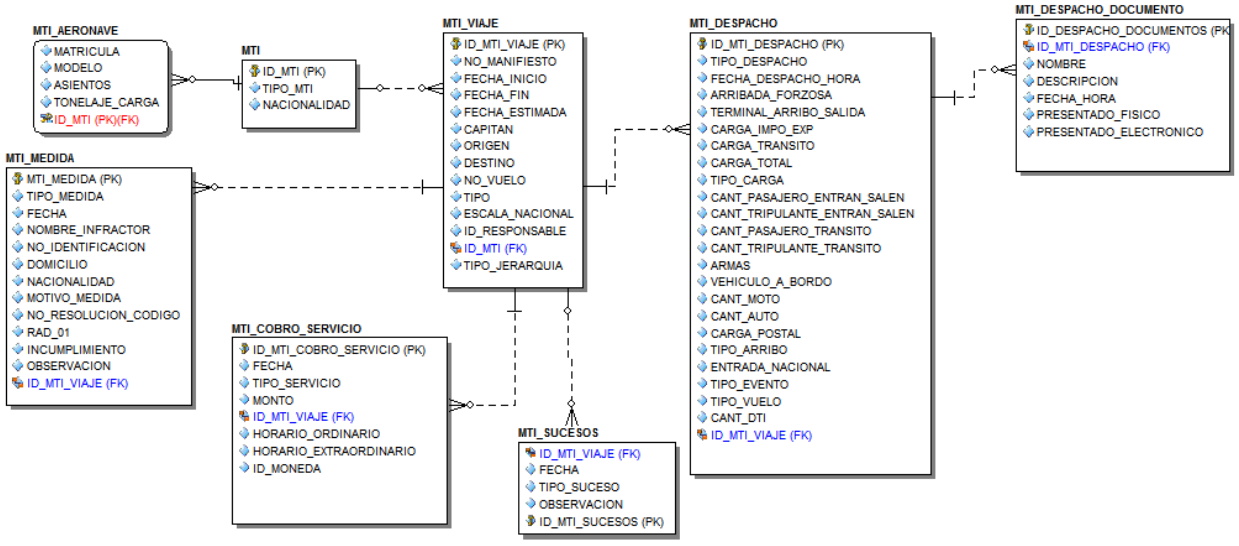

Figura 17 Porción del modelo entidad relación de la aplicación

Conclusiones Parciales

En este capítulo se hizo una breve descripción de la arquitectura del sistema GINA así como los rasgos particulares definidos para el sistema de despacho de MTI aéreos. Se definió además el diseño de la aplicación, detallándose los principales artefactos generados a partir de

este flujo de trabajo que constituirán la entrada necesaria para el flujo de trabajo de implementación.

Capítulo 3 Implementación y prueba de la solución propuesta

Introducción

Durante el desarrollo de este capítulo se mostrará la forma en que se llevó a cabo la creación del subsistema, se describirá la forma en que se utilizaron las diferentes herramientas de Symfony, así como la aplicación de otros procedimientos para aplicar las buenas prácticas recomendadas para la implementación de aplicaciones plasmadas en el estudio del arte.

Como una segunda sección del capítulo se documentarán las pruebas realizadas a la aplicación como forma de validación de los resultados obtenidos con el presente trabajo.

Desarrollo

Implementación de la solución propuesta.

Creando el módulo para aeronaves dentro de *sfMtiPlugin*

Para Symfony un módulo representa una página web o un conjunto de ellas con propósitos estrechamente relacionados (13); su creación dentro de una aplicación específica o un plugin, como es el caso (sfMtiPlugin), se realiza a través de un intérprete de líneas de comandos, después de ubicarse en el directorio raíz de la aplicación (GINA) y con el comando ***'$symfony generate:module sfMtiPlugin aeronaves',*** de esta forma se obtendrá una estructura de directorios para el módulo como la que sigue:

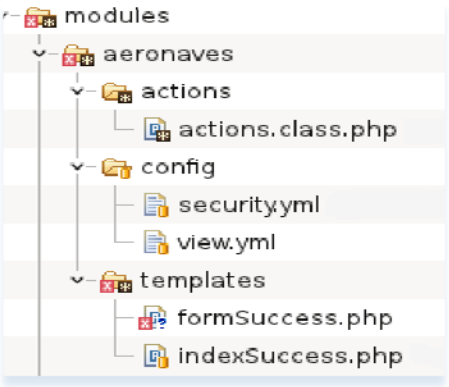

Figura 18 Estructura de directorios del módulo aeronaves

Creando el schema.xml

El archivo *schema.xml* ubicado en la carpeta de configuración de un proyecto de Symfony es usado para almacenar toda la estructura de los datos de la misma, partiendo siempre de la configuración establecida para la base de datos definida en el archivo *database.yml* también del proyecto (para profundizar acerca de la función de este archivo consultar la "Guía definitiva de Symfony1.2"). Para la generación del *schema.xml*, mediante un intérprete de líneas de comandos después de ubicarse en el directorio raíz del proyecto (GINA) se utiliza el comando ***'$symfony propel:build-schema –xml***'.

Creando el modelo

La estructura de datos almacenado en el *schema.xml* es la espina dorsal para la generación del modelo; la creación del mismo se realiza mediante un intérprete de líneas de comandos desde la raíz del proyecto (GINA) utilizando el comando ***'$symfony propel:build-model***', arrojando como resultado la creación del paquete *model* descrito en el capítulo anterior.

Creando los formularios

Para la creación de los formularios, con el apoyo también de un intérprete de comandos desde la raíz del proyecto (GINA) mediante la línea ***'$symfony propel:build-forms –connection='mti'***, obteniendo la estructura de directorios y de clases del paquete *form* descrito en el capítulo anterior

Estándar de codificación utilizado

Para que la codificación sea más entendible y más fácil de mantener se utilizará la "Propuesta de estándar de codificación, Dpto. de Aduana. CEIGE". El mismo tiene como objetivo identificar de forma sencilla cual es el objetivo y las funcionalidades que brinda cada una de las clases y demás componentes de Software dada su nomenclatura; además debe servir de guía para los desarrolladores que tienen que continuar el desarrollo de las aplicaciones luego.

Mensajes informativos, de éxito o de error mostrados al usuario

Para las excepciones lanzadas por la aplicación se tendrán predefinidos en el archivo *app.yml* de la aplicación los diferentes mensajes de error, guardados en variables, que serán mostrados al usuario, lo mismo se aplicará para el resto de los mensajes informativos o de éxito en la ejecución de acciones. Este tratamiento tiene como objetivo controlar los diferentes mensajes

que serán mostrados a los usuarios de la aplicación para garantizar que se muestre exactamente lo que se desea y no información indebida que pueda ser utilizada por usuarios mal intencionados. También tiene como objetivo localizar en un lugar único todos estos mensajes, de forma tal que sean fáciles de obtener, de reutilizar y de modificar en caso de ser necesario.

Comentarios de seguridad en el código

Debido a que los términos a manejar, las diferentes validaciones y verificaciones que deberán llevarse a cabo durante la codificación de la aplicación el código deberá ser comentado como sigue:

- Antes de la declaración de una función se deberá especificar en forma de comentario que objetivo tiene y que se obtendrá de la misma, así como una descripción breve de los parámetros de entrada y salida de la misma.
- Dentro de las funciones deberán ser comentadas las validaciones significativas que constituyen decisiones de seguridad.

Los comentarios de seguridad deberán regirse también por las especificaciones del estándar de codificación antes citado.

Reacción ante fallas.

- Caso de fallos

El subsistema para el despacho de MTI aéreos depende en gran medida de muchos otros subsistemas y plugins para su efectivo funcionamiento, con el objetivo de mantener su correcta ejecución en caso de fallos de estos sistemas externos se aplicará un tratamiento exhaustivo de las excepciones lanzadas por los mismos para garantizar que no sea dañada la usabilidad de la aplicación y que el usuario no se vea afectado por este tipo de errores.

Uso mínimo de privilegios y control de acciones de los usuarios

El subsistema en cuestión debe llevar un puntual control de las acciones de cada usuario en el sistema debido a que hay diferentes operaciones como el “cancelar Despacho” que solamente podrá ser ejecutado por trabajadores de la aduana con privilegios específicos debido a las consecuencias que podría traer un mal manejo de esta acción; con el objetivo de que los usuarios interactúen con la aplicación haciendo un uso mínimo de privilegios el sistema se

ejecutará bajo el control del *plugin* para la gestión de usuarios y permisos denominado *suAdminPlugin*. El sistema registrará además, haciendo uso de un servicio del propio *SuAdminPlugin*, las trazas de todas las acciones realizadas en el sistema por los usuarios del mismo.

Diagrama de componentes

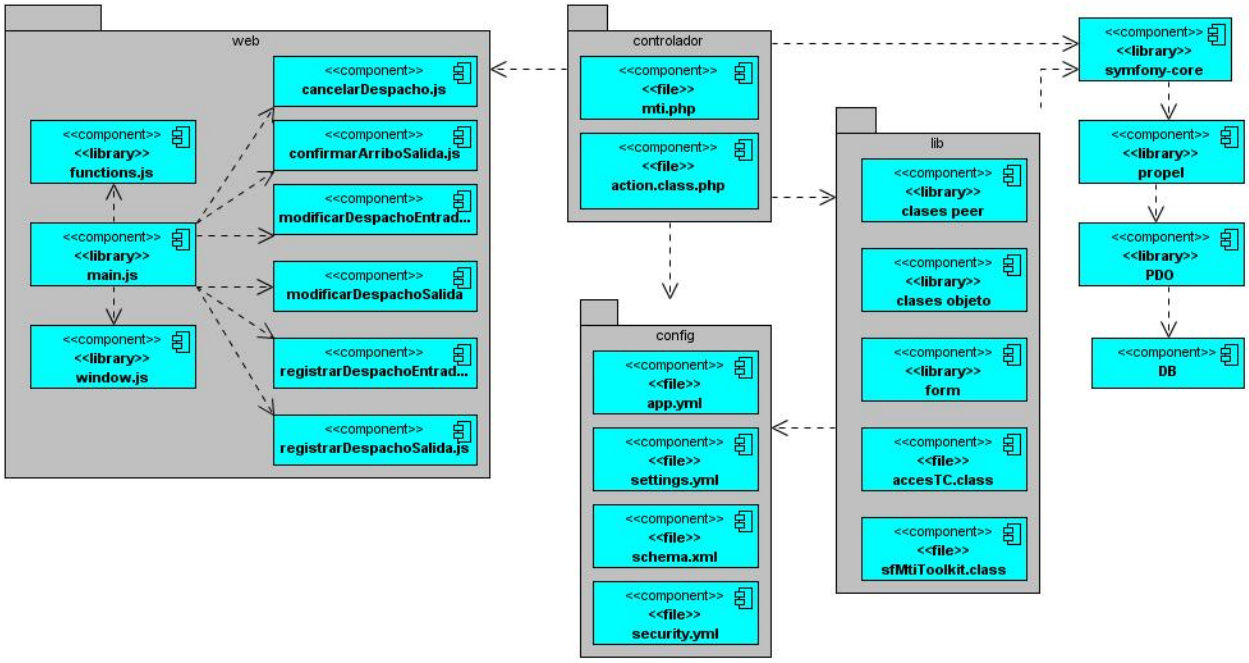

Figura 19 Diagrama de componentes de la aplicación.

Ejemplo de implementación. Requisito "Confirmar arribos/salidas internacionales de aeronaves", escenario "Confirmar Arribos/Salidas de aeronaves".

La interfaz de la Figura 20 muestra el resultado de la interpretación del cliente (Firefox v4.0) del archivo *confirmarArriboSalida.js*. Este archivo no es más que la implementación del formulario *confirmarArriboSalidaVueloForm* que forma parte del diagrama de clases del diseño de la Figura 6 elaborado para el requisito en cuestión.

Figura 20 Vista del formulario "confirmarArriboSalidaVueloForm.js".

Para confirmar un arribo, se deben tener listados previamente los viajes sin confirmar dentro del rango de fechas solicitado por el usuario, la figura anterior muestra el resultado de un filtrado de ejemplo; para la realización de este filtrado, se definió durante el diseño que debía existir una acción codificada en la clase *action.class* denominada "listarVuelos" debe ser invocada desde la vista mediante una petición al servidor recibida y decodificada por el controlador frontal de la aplicación y reenviada al *action.class*, después de ser presionado por el usuario el botón "Buscar"(ver Figura 6, Figura 7). El resultado de la llamada a esta acción debe ser un objeto Json (consultar sección "Arquitectura del sistema Gina" del capítulo 2), que contendrá los datos que más tarde deben cargarse en la tabla identificada en la interfaz como "Listado de vuelos sin confirmación" (ver Figura 20).

Para la propia confirmación de un arribo, se definió también durante el diseño que debe existir un acción denominada *"confirmarArriboSalida"* que debe ser llamada desde la vista mediante otra petición al servidor después de ser presionado por el usuario el botón *"Confirmar"*. El resultado de esta llamada debe ser un objeto *Json* que contenga un mensaje informativo o de error, que le indique al usuario si tuvo éxito o no la ejecución de la operación solicitada y por qué en caso de fracaso (ver Figura 6,Figura 7,Figura 21).

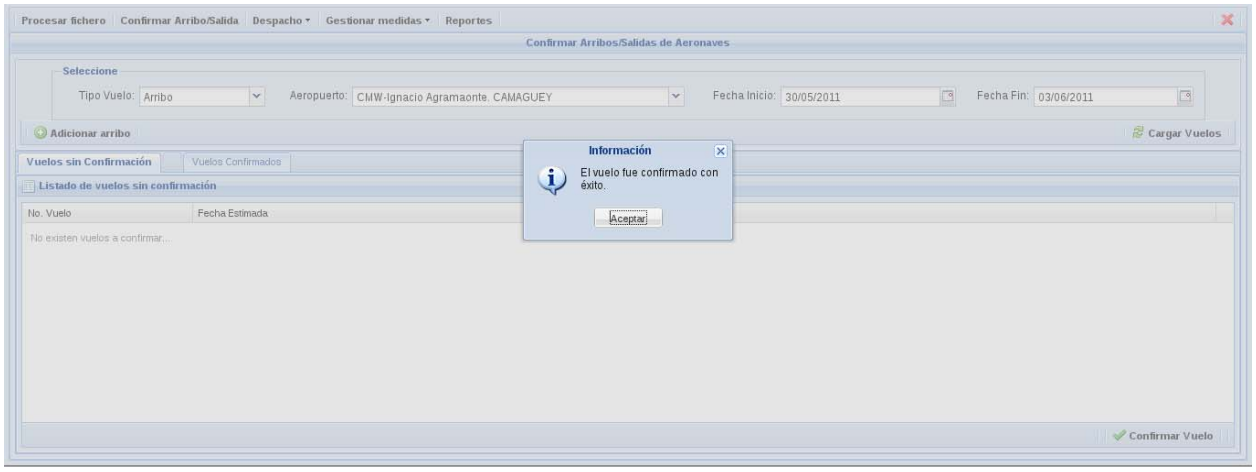

Figura 21 Vista del formulario "confirmarArriboSalidaVueloForm.js".

La clase controladora *action.class* no es la encargada de acceder directamente a los datos que son codificados a objetos *Json* y devueltos a la vista para ser mostrados al usuario, sino que sirve de puente (entre la vista y el modelo), y delega las diferentes responsabilidades a las clases del modelo. Por tanto durante el diseño se definieron las diferentes funcionalidades de acceso a datos necesarias para listar los próximos vuelos sin confirmar y confirmar propiamente el arribo de los mismos al aeropuerto. Las funcionalidades especificadas para asumir estas responsabilidades y que son accedidas desde *action.class* son: "*listarVuelos*" *y "confirmarArriboSalida"* de la clase MtiViajePeer (ver Figura 6 y Figura 7). El requisito culmina cuando es actualizado el listado de vuelos sin confirmar donde no deben reaparecer los vuelos ya confirmados (ver Figura 21).

Prueba de la solución propuesta. Caja Negra

Caso de prueba #1

Descripción general

Requisito: "Confirmar Arribos/Salidas internacionales de aeronaves". Escenario: "Confirmar Arribo/Salida". El sistema permite listar los próximos arribos o salida a confirmar según los criterios de búsqueda y permite confirmar el arribo o salida del viaje.

Condiciones de ejecución

El usuario debe estar autenticado en el sistema. El usuario debe seleccionar el menú "Confirmar Arribo/Salida".

Sección 1

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo central
Listar próximos arribos o salidas	EC 1.1 Buscar sin especificar criterio de búsqueda.	El sistema no permite buscar los viajes sin un criterio de búsqueda especificado.	-Se presiona el botón Buscar. -El sistema muestra los campos obligatorios para el criterio de búsqueda y un mensaje informativo.
	EC 1.2 Buscar especificando solamente tipo de vuelo.	El sistema no permite buscar los viajes sin un criterio de búsqueda válido.	-Se presiona el botón Buscar. -El sistema muestra los campos obligatorios para el criterio de búsqueda y un mensaje informativo.
	EC 1.3 Buscar especificando solamente tipo de vuelo y aeropuerto.	El sistema permite buscar los próximos arribos o salidas registrados con la fecha estimada de llegada o salida igual a la fecha actual.	-Se selecciona el tipo de vuelo y el aeropuerto. -Se presiona el botón Buscar. -El sistema lista los datos relativos a los viajes sin confirmar.
	EC 1.4: Buscar especificando tipo de vuelo, el aeropuerto y el rango de fechas.	El sistema permite buscar los viajes cuya fecha estimada de llegada o salida se encuentra dentro del rango de fechas especificado.	-Se introduce el tipo de vuelo, el aeropuerto y el rango de fechas. - Se presiona el botón Buscar. -El sistema lista los datos relativos a los viajes sin confirmar.

Tabla 12 Caso de prueba. Sección 1.

Juego de datos a probar y resultados

ID del escenario	Escenario	Tipo de vuelo	Aeropuerto	Fecha inicio	Fecha Fin	Respuesta del sistema	Resultado de la prueba
EC 1.1	Buscar sin especificar criterio de búsqueda.	vacío	vacío	vacío	vacío	El sistema debe mostrar en rojo los campos obligatorios para el criterio de búsqueda y mostrar un mensaje informando que debe llenar el formulario de criterios de búsqueda.	El sistema muestra los campos obligatorios en rojo y muestra un mensaje.(ver Anexo 1)
EC 1.2	Buscar especificand	Arribo	vacío	vacío	vacío	El sistema debe mostrar los	El sistema muestra los

	o solamente tipo de vuelo.					campos del aeropuerto y del rango de fechas en rojo y mostrar un mensaje informando que debe completar el formulario de criterios de búsqueda.	campos del aeropuerto y del rango de fechas en rojo y muestra un mensaje informando que debe completar el formulario de criterios de búsqueda.(ver Anexo 2)
EC 1.3	Buscar especificando solamente tipo de vuelo y aeropuerto.	Arribo	Ignacio Agramonte	vacío	vacío	El sistema debe buscar los próximos arribos sin confirmar cuya fecha estimada de llegada sea la fecha actual.	El sistema muestra los próximos arribos existentes sin confirmar, cuya fecha estimada de llegada es la fecha actual.(ver Anexo 3)
EC 1.4	Buscar especificando tipo de vuelo, el aeropuerto y el rango de fechas.	Arribo	Ignacio Agramonte	01/12/2008	fecha actual	El sistema debe buscar los próximos arribos sin confirmar cuya fecha estimada de llegada esté dentro del rango de fechas especificado.	El sistema muestra los próximos arribos sin confirmar cuya fecha estimada de llegada está dentro del rango de fechas especificado.(ver Anexo 4)

Tabla 13 Caso de prueba. Sección 1. Juego de datos a probar y resultados.

Sección 2

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo central
Confirmar Arribo	EC 2.1 Confirmar sin especificar viaje.	El sistema no permite confirmar un viaje que no se halla especificado.	-Se presiona el botón Confirmar. -El sistema muestra un mensaje informativo al usuario, aclarando que debe seleccionar un

			viaje a confirmar.
	EC 2.2 Confirmar especificando un viaje.	El sistema permite confirmar el arribo del viaje seleccionado. (si no ha sido confirmado desde otra estación de trabajo)	-Se selecciona el viaje a confirmar. -Se presiona el botón Confirmar -El sistema confirma el arribo y muestra un mensaje de éxito.

Tabla 14 Caso de prueba. Sección 2

Juego de datos a probar y resultados

ID del escenario	Escenario	Viaje(No Vuelo)	Respuesta del sistema	Resultado de la prueba
EC 2.1	Confirmar sin especificar viaje.	vacío	El sistema debe mostrar un mensaje aclarando que debe seleccionar un viaje de la lista.	El sistema muestra un mensaje aclarando que debe seleccionar un viaje de la lista. (ver Anexo 5)
EC 2.2	Confirmar especificando un viaje.	cu0002	El sistema debe mostrar un mensaje informando que el vuelo ha sido confirmado con éxito.	El sistema muestra un mensaje informando que el vuelo ha sido confirmado con éxito.(ver Anexo 6)

Tabla 15 Caso de prueba. Sección 2. Juego de datos a probar y resultados.

Caso de prueba #2

Descripción general

Requisito: “Gestionar despacho de aeronaves”. Escenario: “Registrar despacho de entrada”. El sistema permite listar los arribos confirmados según los criterios de búsqueda, permite cargar los datos del viaje en un formulario y completar los datos necesarios del despacho a ser registrado.

Condiciones de ejecución

El usuario debe estar autenticado en el sistema. El usuario debe seleccionar el menú "Gestionar despacho".

Sección 1

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo central
Listar arribos confirmados	EC 1.1 Buscar arribos sin especificar criterios de búsqueda	El sistema no permite listar arribos confirmados sin especificar criterio de búsqueda.	-Se presiona el botón "Listar" -El sistema muestra los datos obligatorios en rojo y no permite hacer la operación.
	EC 1.2 Buscar arribos especificando solamente el aeropuerto destino.	El sistema lista los arribos filtrados por la fecha actual y el aeropuerto destino especificado.	-Seleccionar aeropuerto destino. -El sistema muestra una lista con los arribos confirmados filtrados por el aeropuerto destino especificado.
	EC 1.3 Buscar arribos especificando aeropuerto destino y rango de fechas.	El sistema permite listar los arribos confirmados filtrados por el aeropuerto destino especificado y el rango de fechas seleccionado.	-Seleccionar aeropuerto destino. -Seleccionar fecha inicio y fecha fin del rango. -El sistema muestra en una lista con los arribos confirmados filtrados por los datos seleccionados.

Juego de datos a probar y resultados

ID del escenario	Aeropuerto	Rango de fechas	Respuesta del sistema	Resultado de la prueba
1.1	vacío	vacío	El sistema debe mostrar el campo del aeropuerto en rojo.	El sistema no muestra el campo del aeropuerto en rojo y no permite ejecutar la búsqueda. (ver

				Anexo 7)
1.2	Ignacio Agramonte-Camagüey	vacío	El sistema debe listar los arribos confirmados cuya fecha de confirmación se corresponda con la fecha actual.	El sistema lista un arribo. (ver Anexo 8)
1.3	Ignacio Agramonte-Camagüey	25/04/2011 a 03/06/2011	El sistema debe listar los arribos confirmados cuya fecha de confirmación esté dentro del rango especificado.	El sistema lista un arribo. (ver Anexo 9)

Sección 2

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo central
Registrar despacho de entrada a arribos confirmados.	EC 2.1 Seleccionar arribo de estado confirmado.	El sistema carga en un formulario los datos del viaje y del MTI asociado(si tiene)	-Seleccionar arribo de la lista. -Presionar botón “Despachar”.
	EC 2.2 Registrar sin completar los datos obligatorios del despacho (matrícula, origen, aerolínea, tipo de vuelo).	El sistema no permite registrar un despacho.	-Se presiona el botón “Registrar despacho” -El sistema muestra los campos obligatorios en rojo y no permite registrar un despacho.
	EC 2.3 Registrar completando los datos obligatorios del despacho (matrícula, origen, aerolínea, tipo de vuelo).	El sistema permite registrar un despacho.	-Se presiona el botón “Registrar despacho”. -El sistema muestra un cartel informando al usuario que el despacho ha sido registrado con éxito.

Juego de datos a probar y resultados

ID del escenario	Matricula	Origen	Aerolínea	Tipo de vuelo	Respuesta del sistema	Resultado de la prueba
2.1					El sistema debe cargar en un formulario los datos del viaje y	El sistema carga en un formulario los datos del viaje seleccionado y del MTI

					del MTI asociado	asociado. (ver Anexo 10)
2.2	vacío	vacío	vacío	vacío	El sistema debe mostrar los campos obligatorios en rojo y no permite registrar un despacho.	El sistema muestra los campos obligatorios en rojo y no permite registrar un despacho. (ver Anexo 11)
2.3					El sistema debe mostrar un mensaje informando que el despacho se registró con éxito.	El sistema muestra un mensaje informando que el despacho se registró con éxito. (ver Anexo 12)

Al concluir las pruebas realizadas a la aplicación no se detectaron no conformidades que señar.

Conclusiones parciales

En este capítulo se describió como se llevó a cabo la construcción de la aplicación y se documentaron pruebas de caja negra realizadas a la misma.

Conclusiones

Al concluir la realización de este trabajo se cumplieron los objetivos propuestos con el mismo, se elaboró el modelo del diseño, se describieron rasgos arquitectónicos particulares del subsistema, se elaboró el modelo de datos y se obtuvo el código fuente de la aplicación, otro elemento a destacar es la integración con el subsistema API y otros no menos significativos, lográndose la construcción de una versión funcional de un sistema informático para llevar a cabo el despacho de los arribos y salidas de los MTI aéreos, que responde al 100% de los requisitos funcionales exigidos por el cliente, ajustándose de esa forma a las necesidades primarias que propiciaron la realización de este trabajo:

- Bajo costo de producción.
- Ajuste a las normativas aduaneras de la AGR.
- Informatización de los procesos de negocio relativos al despacho de los MTI aéreos.

La informatización de procesos que demandan alto nivel de seguridad y rendimiento, requieren un trabajo minucioso que no pase por alto detalles que puedan afectar el resultado esperado; la informatización de los procesos aduanales en particular requiere de un trabajo en conjunto, para lograr la fortaleza y la integralidad que requiere un sistema que se encargue de manejar procesos tan complejos e interdependientes como son los que se llevan a cabo en las sedes de la AGR de todo el país.

Recomendaciones

- Aplicar técnicas de validación del diseño de la aplicación que contribuyan a predecir posibles fallas en la seguridad y el rendimiento del sistema.
- Aplicar pruebas de estrés u otros tipos a la aplicación para asegurar su rendimiento en un ambiente real de explotación.
- Refactorizar con el objetivo de optimizar la aplicación en términos de rendimiento y tiempo de respuesta a las peticiones, así como encontrar posibles fallas no previstas.
- Optimizar la integración con el subsistema API, de manera que se le pueda notificar al usuario si este último ha tenido alguna falla por lo cual no se ha podido extraer la información de los próximos arribos de los cuales se tiene la información adelantada de pasajeros.
- Continuar la construcción del sistema para la recepción y el procesamiento de la información adelantada de cargas para un óptimo funcionamiento de la aplicación.
- Describir un proceso de desarrollo de software adecuado a las necesidades del proyecto Aduana para guiar la construcción de las aplicaciones de GINA, centrado en la seguridad, debido a los requisitos no funcionales referentes que la misma exige.

Referencias bibliográficas

1. **Fidel Castro Ruz.** Decreto Ley De Aduanas. *Gaceta Oficial de la República de Cuba.* 27 de Mayo de 1996.

2. **OWASP Foundation .** *OWASP TESTING GUIDE.* 2008.

3. **Howard, Michael y LeBlanc's, David.** *Writing Secure Code.* 2003.

4. **Rodríguez, José A. Cobo.** *Título: Línea Base Arquitectónica para el Polo Sistemas Tributarios y de Aduanas.* Ciudad de La Habana, Cuba : s.n., 2010.

5. **Potencier, Fabien.** El framework de formularios. *El tutorial Jobeet.* 2009.

6. Capítulo 17. Personalizar Symfon 17.4. Plugins. [aut. libro] Francois Zaninotto y Fabien Potencier. *Symfony 1.2, la guía definitiva.* 2008.

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

8. **Aldás, Daniel E. Mena y Andrade, Maritza A. Cadena.** *GUIA PRACTICA PARA EL USO DE PATRONES DE DISEO EN EL DESARROLLO DE SOFTWARE.* Quito, Ecuador : s.n., 2010.

9. Capítulo 6. El Controlador. [aut. libro] Francois Zaninotto y Fabien Potencier. *Symfony 1.2, la guía definitiva.* 2008.

10. **Potencier, Fabien.** *Los formularios de Symfony 1.2.*

11. **Fowler, Martin.** *Patterns of Enterprise Application Architecture.* s.l. : Addison Wesley, 2002.

12. **SEI.** Software Engineering Institute. [En línea] [Citado el: 07 de 04 de 2011.] http://www.sei.cmu.edu/architecture/start/community.cfm.

13. **Acuña, Karenny Brito.** *SELECCIÓN DE METODOLOGÍAS DE DESARROLLO PARA APLICACIONES WEB EN LA FACULTAD DE INFORMÁTICA DE LA UNIVERSIDAD DE CIENFUEGOS.* Cienfuegos, Cuba : s.n., 2009.

14. Capítulo 2. Explorando el interior de Symfony, 2.1. El patrón MVC. [aut. libro] Francois Zaninotto y Fabien Potencier. *Symfony 1.2, la guía definitiva.* 2008.

15. Capítulo 2. Explorando el interior de Symfony, 2.2. Organización del código, 2.2.1 Estructura del proyecto: Aplicaciones, Módulos y Acciones. [aut. libro] Francois Zaninotto y Fabien Potencier. *Symfony 1.2, la guía definitiva.* 2008.

16. Capítulo 8. El Modelo. [aut. libro] Francois Zaninotto y Fabien Potencier. *Symfony 1.2, la guía definitiva.* 2008.