Universidad de las Ciencias Informáticas

“Facultad 3”

Título: Diseño e implementación del componente Historial de Medios de Transporte Internacional para el sistema Gestión Integral de Aduanas.

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

Autora:

Ivian Laobel Castellano Betancourt.

Tutores:

Ing. Manuel Ramón Almaguer Ochoa.

Ing. Yasel Antonio Romero Piñeiro.

Ciudad de La Habana, Junio 29 de 2011

“Año 54 de la Revolución”

Declaración de Autoría.

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

"Ivian Laobel Castellano Betancourt."

_____

"Ing. Manuel Ramón Almaguer Ochoa."

_____

"Ing. Yasel Antonio Romero Piñeiro"

_____

"Las ideas son hoy el instrumento esencial en la lucha de nuestra especie por su propia salvación.
Los valores fundamentales, entre ellos la ética, se siembran a través de ella".

Fidel Castro Ruz.

Agradecimientos.

En primer lugar a mis padres (Carlos y Julia) por su dedicación, esfuerzo y siempre confiar en mí, todo se lo debo a ustedes. Por siempre apoyarme en todo, respetar mis decisiones. Por ustedes hoy soy ingeniera.

A mi hermana Irian por su preocupación, por ser como un ejemplo y por estar orgullosa de mí.

A mis sobrinos (Jorgito y Cristian) por ser mis tesoros y por sentirme realizada al saber que hoy me ven como ejemplo.

A mi familia por siempre estar preocupados por mis estudios y formación como profesional en especial a mis abuelas (Luisa y Olga) y mis tías Mari y Reyna.

A mis tutores Yasel y Manuel, gracias a ellos hoy este sueño se hace realidad, cuando más necesitaba ayuda ahí estaban ellos, al igual que Yordani, Yaksel, Islandy, Dayan y Eyeris. A todos ustedes…Muchas Gracias.

En general a todos aquellos compañeros del proyecto que tanto apoyo me brindaron…

A mis amistades en Venezuela por estar atentos desde lejos, por el apoyo brindado y sentirme con ellos como en familia en especial a Gerlady, Rodolfo (Chachi) y Zenaida.

A los profesores que han formado parte de mi formación como ingeniera.

A Maidelis, Anyta, Lisandra Tamayo y Yisel por ser como hermanas para mí y siempre poder contar con ellas.

A todos mis amigos…los de antes…ahora…y siempre.

A la Revolución, Fidel y la UCI por haberme dado la oportunidad de formarme como Ingeniera.

Dedicatoria.

A mis padres por apoyarme siempre en todo y darme amor.

A mi hermana, mis sobrinos y mi cuñado.

A mis Abuelos que aunque no estén presentes sé que se sentirían orgullosos de mí.

A mis Abuelas por su grandeza y siempre darme ese amor incondicional.

A mis tías, tíos y demás familiares.

A mis amigos, los que vienen conmigo desde hace mucho y los de ahora.

A todos gracias por formar parte de este momento.

Resumen.

En el departamento de Soluciones para la Aduana se desarrollan aplicaciones para automatizar la Aduana General de la República de Cuba (AGR). El subsistema Despacho de Medios de Transporte Internacional (MTI) es una de ellas; en él se registran todos los buques, aeronaves y embarcaciones de recreo que entran y salen del país. Existen otras aplicaciones del sistema Gestión Integral de Aduanas (GINA) que necesitan conocer toda o parte de la información referente al MTI. Cada uno de los subsistemas involucrados en dicha situación se ven obligados a obtener un registro histórico del recorrido de los MTI en viajes anteriores. Teniendo en cuenta las dificultades descritas se persigue como objetivo desarrollar un componente que gestione el historial de un MTI para centralizar la información referente a éste y facilitar la comunicación entre las aplicaciones del sistema GINA de la AGR. Para ello se realizó un estudio de sistemas en el mundo con objetivos similares a la solución.

En el desarrollo de la solución se utiliza como modelo de desarrollo el definido por el departamento según las particularidades del mismo. Haciendo uso del Lenguaje Unificado de Modelado (UML)[1] se generan los distintos artefactos tanto del diseño como de la implementación. Se valida el diseño haciendo uso de diferentes métricas. Se implementa el sistema y se comprueba la solución con la realización de estrategias de pruebas unitarias de *Symfony*. Como resultado se obtuvo un componente para el sistema GINA capaz de centralizar la información referente al historial de MTI.

Palabras Claves:

Aduana, Componente, Historial, GINA, MTI

[1] Conjunto de notaciones y diagramas estándar utilizados para modelar sistemas orientados a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan.

Índice.

Índice

Índice

Introducción.

La Aduana General de la República de Cuba (AGR) es el órgano oficial que controla el tráfico de mercancías, pasajeros y medios de transporte internacional; tiene en funcionamiento diferentes sistemas informáticos que facilitan el control y la gestión de los procesos aduanales. Estos sistemas difieren en cuanto al lenguaje de programación con que fueron desarrollados, plataformas y gestores de base de datos que utilizan. Con el objetivo de integrar estas aplicaciones y generalizar los procesos de implementación y estandarización de las soluciones de desarrollo surge el Sistema para la Gestión Integral de Aduanas (GINA) (1).

El sistema GINA es la herramienta informática que utiliza la AGR para informatizar sus procesos y es desarrollada en la Universidad de las Ciencias Informáticas (UCI) con la ayuda de los especialistas del Centro de Automatización para la Dirección y la Información (CADI).

En el departamento de Soluciones para la Aduana perteneciente al Centro de Informatización para la Gestión de Entidades (CEIGE) en la Facultad 3, se desarrollan aplicaciones para automatizar la AGR. El subsistema Despacho de Medios de Transporte Internacional (MTI) es una de ellas, en él se registran todos los buques, aeronaves y embarcaciones de recreo que entran y salen del país. Existen otras aplicaciones del sistema GINA que necesitan conocer toda o parte de la información referente al MTI en viajes anteriores, así como las incidencias cometidas, el control aplicado o los hechos ocurridos en la entrada, la permanencia o la salida del país.

Cada uno de los subsistemas involucrados en dicha situación se ven obligados a obtener un registro histórico de los MTI en cuestión. Actualmente este proceso se realiza de forma manual lo que tributa al empleo excesivo de tiempo. Además, no existe un lugar común en el sistema GINA para almacenar dicho historial[2], lo que lleva consigo la existencia de redundancia de información dentro de la base de datos del sistema GINA.

[2] Reseña circunstanciada de los antecedentes de algo o de alguien.

Anteriormente se identificaron requisitos en diferentes subsistemas con comportamientos comunes y que representan funcionalidades implementadas de forma diferente con el fin de gestionar el recorrido de un MTI en el país. Por lo planteado se comienza a analizar que exista información repetida innecesariamente y centralizar el historial de MTI en el sistema GINA se vuelve una necesidad.

A partir de la problemática se formula el siguiente **problema de investigación**: ¿Cómo materializar los requisitos elicitados en el análisis de los procesos de negocio involucrados en el historial de Medios de Transporte siguiendo las pautas de arquitectura establecidas?

Basándose en lo anterior se define como **objeto de estudio**: la gestión de procesos aduanales.

Para resolver el problema planteado se propone como **objetivo general:** diseñar e implementar un componente que gestione el historial de un MTI para centralizar la información referente a éste y facilitar la comunicación entre las aplicaciones del sistema GINA siguiendo la arquitectura definida; siendo el **campo de acción:** la informatización de los procesos relacionados con el Historial de los Medios de Transporte Internacional.

Como **Objetivos específicos** se plantean:

- ✓ Realizar estudio del estado del arte del tema tratado.
- ✓ Realizar el estudio de las tecnologías y herramientas para ser utilizadas en el desarrollo de la solución.
- ✓ Modelar el diseño del componente Historial de un MTI.
- ✓ Implementar el modelo escogido para el componente Historial de un MTI.
- ✓ Validar la solución propuesta.

Con el fin de dar cumplimiento a los objetivos del presente trabajo, se concibieron las siguientes **tareas de investigación**:

- ✓ Evaluación del estado actual de las soluciones informáticas existentes para la gestión aduanera.
- ✓ Evaluación de las tecnologías y herramientas para ser utilizadas en el desarrollo de la solución.
- ✓ Modelación de los artefactos del diseño del componente Historial de un MTI.

- ✓ Validación del diseño.
- ✓ Implementación del modelo de diseño escogido para el componente Historial de un MTI.
- ✓ Validación de la solución propuesta.

El presente trabajo de diploma está estructurado por 3 capítulos:

Capítulo 1. Fundamentación Teórica.

- ✓ Conceptos asociados al dominio del problema.
- ✓ Estudio del estado del arte del tema tratado.
- ✓ Tendencias tecnológicas.

Capítulo 2. Diseño e implementación.

- ✓ Descripción de la solución propuesta.
- ✓ Diseño de la solución.
- ✓ Implementación de la solución.

Capítulo 3. Validación de la solución.

- ✓ Validación del diseño.
- ✓ Validación de la solución propuesta mediante pruebas unitarias con el *framework Symfony*.

Capítulo 1: Fundamentación teórica.

1.1 Introducción.

En el presente capítulo se hace un profundo estudio acerca del dominio del problema, así como el análisis de los conceptos fundamentales tratados en el mismo. Se profundiza en la búsqueda de soluciones que presentes características similares a las necesitadas por la aduana cubana. Otro aspecto importante, es el estudio de las tecnologías y herramientas de desarrollo de software en la actualidad, en específico las que son utilizadas en el departamento de soluciones para la Aduana puesto que este trabajo está enmarcado en el estudio de componentes reutilizables que permiten la comunicación entre aplicaciones.

1.2 Componente de software.

En el desarrollo de software la reutilización de código se ha limitado por muchos años a recurrir al empleo del código ya existente. Actualmente con el nuevo paradigma de programación se ha dado un gran paso para la optimización y ahorro en la construcción de código, permitiendo el ahorro de tiempo, esfuerzo y la evolución de la Ingeniería de Software, con vista a evitar el desarrollo de los sistemas desde cero, mediante la aplicación de técnicas de reutilización de partes del software ya desarrolladas y probadas.

Según *Szyperski*[3]:

"Un componente es una unidad de composición de aplicaciones de software, que tiene un conjunto de interfaces y un conjunto de requisitos, que ha de poder desarrollarse e incorporarse a un sistema, y componerse con otros de forma independiente, en tiempo y espacio." (2)

Por otro lado, existe también una confusión habitual entre el concepto de componente con el de clase, objeto, módulo, entre otros. Con el fin de distinguirlas entre sí, se recogen las definiciones más aceptadas de otras entidades software, así como sus principales diferencias.

[3] *Clemens Szyperski*: Arquitecto de Software en *Microsoft Research*. Su enfoque está en el uso efectivo del software de componentes para construir nuevos tipos de software.

- ✓ **Clase:** no hacen referencia explícita a sus dependencias y requisitos. Las clases suelen construirse mediante herencia de implementación, y por tanto no suelen permitir una instanciación e instalación por separado de la clase base y de sus clases hijas. Esto puede causar problemas si se realiza composición tardía (pudiendo aparecer, por tanto, el problema de la clase base frágil).
- ✓ **Módulo:** conjunto de clases, y en ocasiones junto con otros elementos no orientados a objetos, así como procedimientos y funciones.
- ✓ **Paquetes/Librerías:** conjunto de clases, agrupadas conceptualmente, no suelen ser ejecutables, y pueden ser consideradas como la versión orientada a objetos de las librerías tradicionales.
- ✓ **Subsistema:** agrupación de elementos de modelado que representa una unidad de comportamiento en un sistema físico.
- ✓ **Recurso:** colección no modificable de elementos con tipo. (3)

Una de las **características** más importantes de los componentes es que son reutilizables. Para ello deben satisfacer como mínimo el siguiente conjunto de **requisitos**:

- ✓ **Identificable**: los componentes deben tener una identificación clara y consistente que facilite su catalogación y búsqueda en repositorios de componentes.
- ✓ **Accesible sólo a través de su interfaz**: los componentes deben permitir que sea reemplazado por otro componente que implemente la misma interfaz.
- ✓ **Sus servicios son invariantes**: la implementación de estos servicios de un componente puede ser modificada, pero no deben afectar la interfaz.
- ✓ **Documentado**: los componentes deben tener una documentación adecuada que facilite su búsqueda en repositorios de componentes, evaluación, adaptación a nuevos entornos, integración con otros componentes y acceso a información de soporte. (4)

Para favorecer su reutilización es necesario que un componente sea:

- ✓ **Genérico**: sus servicios pueden ser usados en una gran variedad de aplicaciones.
- ✓ **Auto contenido**: es conveniente que dependa lo menos posible de otros componentes para cumplir su función de forma tal que pueda ser desarrollado, probado, optimizado, utilizado, entendido y modificado individualmente.

- ✓ **Mantenido**: un componente debe estar inmerso en un proceso de mejoramiento continuo que le garantice al integrador nuevas versiones que incluyan correctivos, optimizaciones y nuevas características. esto contribuye a que dicho componente sea seleccionado con mayor frecuencia para formar parte de sistemas de software.
- ✓ **Puede ser reutilizado dinámicamente:** puede ser cargado en tiempo de ejecución en una aplicación. (4)

1.2.1 Conceptos asociados al componente de software.

Los paradigmas presentes en la Ingeniería del Software permiten un aprovechamiento eficiente del código ya construido. Uno de estos paradigmas se conoce como Desarrollo de Software Basado en Componentes (DSBC) (2)

El DSBC es la disciplina fundamentada por componentes el cual se ha convertido actualmente en uno de los mecanismos más efectivos para la construcción de grandes sistemas y aplicaciones de software (3). Dentro de los objetivos que se busca con el DSBC se encuentran, el esfuerzo que requiere implementar una aplicación y los costos del proyecto, reducir el tiempo de trabajo e incrementar el nivel de productividad de los grupos desarrolladores favoreciendo de esta forma la adquisición de las tecnologías que más se adapten a las necesidades de la empresa y de esta forma desarrollar una aplicación de manera personalizada, minimizar los riesgos y maximizar los beneficios en el proceso de desarrollo de software. (5)

1.2.2 Categorías del desarrollo de software basado en componentes.

- ✓ **Desarrollo de componentes:** este proceso implica la adaptación o desarrollo de componentes con el propósito de ser reutilizados en futuras aplicaciones. Su objetivo es producir repositorios de activos que puedan ser reutilizados en el desarrollo de software.
- ✓ **Desarrollo de software con reutilización de componentes:** es un proceso en el cual el desarrollo de una nueva aplicación involucra la reutilización de un conjunto de componentes existentes. Este enfoque maximiza la reutilización de componentes de software existentes y reduce el número de componentes que requieren ser desarrollados en su totalidad. Para ser exitoso, este proceso demanda dos condiciones mínimas:

- La existencia de repositorios o bases de componentes reutilizables.
- Confiabilidad de los componentes y que estos actúen de acuerdo a sus especificaciones. (4)

1.2.3 Beneficios del desarrollo de software basado en componentes.

El DSBC posee varias **ventajas** en cuanto a su utilización, a continuación se describen las mismas:

- ✓ **Reutilización del software**. Como resultado se puede alcanzar un mayor nivel de reutilización de software.
- ✓ **Simplifica las pruebas**. Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
- ✓ **Simplifica el mantenimiento del sistema**. Cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.
- ✓ **Mayor calidad**. Dado que un componente puede ser construido y luego mejorado por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

1.3 Historial de Medios de Transporte Internacional.

Un historial de MTI es un conjunto de hechos que suceden en el transcurso de vida de un MTI. Contar con el registro histórico de estos hechos sería uno de los aspectos más importantes dentro de la AGR puesto que serviría de referencia a la hora de gestionar y controlar los MTI en las Aduanas de Cuba.

1.3.1 Sistemas informáticos para la Aduana.

En el mundo existen diversas herramientas utilizadas para la gestión de los procesos aduanales. Éstas han sido de gran utilidad en el avance de la informática. A continuación se muestran algunos de los sistemas existentes a nivel internacional utilizados con este fin.

Sistema de Ordenamiento Fiscal del Impuestos en Aduanas (SOFIA)

SOFIA en su versión 7.3 es un sistema informático de despacho aduanero que interactúa en forma directa con sus usuarios: Despachantes de Aduana, Empresas de Transporte, Depositarios, Funcionarios de Aduana y con los Organismos vinculados al comercio exterior.

Los objetivos fundamentales del sistema SOFIA son:

- ✓ Mejoramiento de la eficacia administrativa y capacidad de gestión de la Dirección Nacional de Aduanas.
- ✓ Obtención de estadísticas de comercio exterior.
- ✓ Información para la lucha contra el fraude.

Los principios del sistema SOFIA son:

- ✓ Procesamiento descentralizado.
- ✓ Descentralización en el ingreso de la información.
- ✓ Unicidad de criterio de aplicación de la tributación y reglamentación vigente.
- ✓ Descentralización del procedimiento aduanero.
- ✓ Centralización de los datos considerados de interés estadístico en un servidor central.
- ✓ Usuarios (despachantes, depositarios, transportistas, aduaneros) conectados directamente al sistema.

Funcionamiento:

- ✓ SOFIA permite la conexión de los Despachantes de Aduana o de los Agentes de Transporte para la formulación de sus Despachos de Importación/Exportación o Manifiestos desde sus propias oficinas.
- ✓ Existe también la posibilidad de realizar las operaciones desde los Centros Públicos habilitados a tal efecto.
- ✓ El Despachante llega a la Aduana para realizar sus operaciones (presentación del despacho, verificación documental y/o física y posterior retiro de las mercaderías) conforme al canal selectivo asignado.
- ✓ El sistema trabaja electrónicamente para la percepción de tributos con el dinero depositado en un Banco de plaza o en las cajas habilitadas en la DNA, una vez acreditado el monto a la cuenta conjunta del Despachante con el Importador que dispone para afectarse a las liquidaciones de los tributos. (6)

Sistema Aduanero Automatizado (SIDUNEA)

SIDUNEA en su versión 1.18c es la herramienta informática para el control y administración de la gestión aduanera, desarrollada por la Conferencia de las Naciones Unidas sobre el Comercio y el Desarrollo (UNCTAD, por sus siglas en ingles), y que actualmente es usada con éxito en más de 80 países.

SIDUNEA permite realizar un seguimiento automatizado de las operaciones aduaneras y controlar efectivamente la recaudación de los impuestos aduaneros, porque este sistema verifica automáticamente los registros, calcula los impuestos y contabiliza todo lo relativo a cada declaración, con la mínima intervención del factor humano subjetivo.

Entre las ventajas que se pueden obtener con la aplicación del SIDUNEA se encuentran:

- ✓ Optimizar los tiempos y recursos del proceso aduanero.
- ✓ Aplicar la ley con toda justicia.
- ✓ Cobrar correctamente los impuestos y tasas.
- ✓ Detectar los errores en los valores de la declaración.
- ✓ Monitorear el pago de los impuestos
- ✓ Evitar la evasión de impuestos.
- ✓ Minimizar el contrabando.
- ✓ Crear incentivos para el declarante.
- ✓ Administrar efectivamente el proceso de despacho.
- ✓ Poner en práctica un esquema de garantía con la modalidad de pago anticipado, para facilitar el comercio y asegurar el cobro de los derechos aduaneros.
- ✓ Controlar la ruta de comercio por medio de las oficinas de despacho de mercancía de cada aduana. (7)

Conclusiones del estudio:

Teniendo en cuenta las soluciones analizadas anteriormente, a pesar de que brindan una amplia gama de funcionalidades ninguna satisface los requisitos planteados para los procesos del historial de MTI en la AGR. Además, estas soluciones están adaptadas a las necesidades propias del país donde se desarrollaron, son privativas y costosas; por tal motivo no es recomendable que sean empleadas en la

AGR. Por lo tanto se realiza un estudio y análisis para desarrollar una solución donde se ponga de manifiesto las ventajas brindadas por estos sistemas y que se puedan adaptar al sistema aduanal cubano.

1.4 Diseño del software.

El diseño de software abarca un conjunto de principios, conceptos y prácticas que conducen al desarrollo de un sistema o producto de alta calidad. Se encuentra en el núcleo técnico de la respectiva ingeniería y se aplica de manera independiente al modelo del software que se utilice. Una vez que se analizan y especifican los requisitos, el diseño es la última acción de la ingeniería correspondiente dentro de la actividad del modelado. Es la etapa donde se fomentará la calidad de la ingeniería de software. Sin diseño se corre el riesgo de construir un sistema inestable.

1.4.1 Arquitectura de Software.

Según *Roger S. Pressman* la arquitectura es la estructura del sistema, la cual comprende los componentes de software, las propiedades de esos componentes visiblemente externos y las relaciones entre ellos. La arquitectura no es más que la organización de los componentes del sistema de forma que quede registrado la manera en la que colaboran y se relacionan entre ellos. Es una vía en la cual el sistema queda modelado desde distintas perspectivas con el objetivo de lograr y establecer como deberá ser construido el futuro sistema. (8)

Estilos Arquitectónicos.

Un estilo arquitectónico es aproximadamente lo mismo que lo que se acostumbra a definir como patrones arquitectónicos. Un patrón no es más que una solución a un problema en un contexto, que codifica conocimiento específico acumulado por la experiencia en un dominio. (9)

El modelo de arquitectura en capas es uno de los aspectos del desarrollo de software más importante para separar por niveles el desarrollo de aplicaciones web. Este modelo provee la reutilización de código, donde los protagonistas principales son la herencia y el encapsulamiento. El modelo más utilizado actualmente es el modelo de n-capas, en este caso se aborda del más conocido como Modelo-Vista-Controlador (MVC) (**figura 1**), el cual tiene 3 capas. (9)

- ✓ Capa de Presentación o la vista.
- ✓ Capa de Negocio o controlador.

- ✓ Capa de Datos o modelo.

Este patrón separa los datos de una aplicación, la interfaz de usuario y la lógica del negocio en tres componentes distintos. El modelo es donde se encuentran los datos de la aplicación, tiene la principal funcionalidad. La vista muestra la interfaz de usuario. El controlador gestiona la entrada del usuario, funciona como el punto intermedio entre el modelo y la vista. (9)

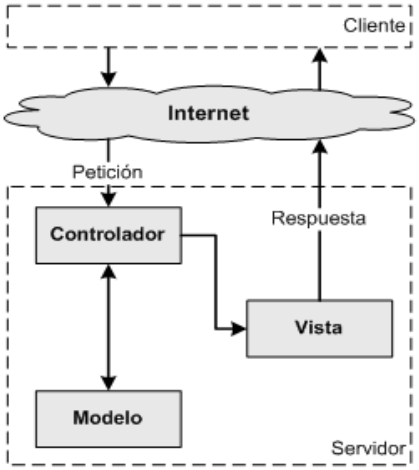

Figura 1. Patrón MVC.

Patrones de diseño.

Un patrón de diseño es una solución a un problema de diseño, una de las características es que debe haber comprobado su **efectividad** resolviendo problemas similares en ocasiones anteriores. Además debe ser **reutilizable**, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

A continuación se detallan los principales patrones de diseño que son implementados por el *framework* de desarrollo *Symfony*, término que se describirá más adelante.

***General Responsibility Assignment Software Patterns* (GRASP)**

Los patrones de Asignación de Responsabilidades (GRASP, por sus siglas en inglés) representan los principios básicos de la asignación de responsabilidades a objetos, expresados en forma de patrones, seguidamente se explican algunos de ellos:

- ✓ **Experto:** la responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada.
- ✓ **Creador:** este patrón como su nombre lo indica es el que crea, guía la asignación de responsabilidades relacionadas con la creación de objetos, se asigna la responsabilidad de que una clase B cree un objeto de la clase A solamente cuando:
    - **B** contiene a **A**.
    - **B** es una agregación (o composición) de **A**.
    - **B** almacena a **A**.
    - **B** tiene los datos de inicialización de **A** (datos que requiere su constructor).
    - **B** usa a **A**.

  A la hora de crear objetos se deben tener en cuenta las características de la clase.

- ✓ **Bajo acoplamiento:** el acoplamiento es una medida de fuerza con que un elemento (está en/ tiene conocimiento de/ confía en/) otros elementos. Este patrón es un principio que asigna la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad, etc.). El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión. Un error muy común es asignarle demasiada responsabilidad y alto nivel de acoplamiento con el resto de los componentes del sistema.
- ✓ **Controlador:** permite asignar la responsabilidad de manejar los mensajes eventos del sistema a una clase que puede representar el sistema total o la organización.
- ✓ **Alta cohesión:** la cohesión es una medida de la fuerza con la que se relacionan las clases y el grado de focalización de las responsabilidades de un elemento. Cada elemento del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable, una clase con baja cohesión hace muchas cosas no relacionadas o hace demasiado trabajo. (10)

***Gang-of-Four* GoF**

Los patrones Pandilla de los cuatro (GoF, por sus siglas en inglés) se descubren como una forma indispensable de enfrentarse a la programación; se clasifican en 3 categorías basadas en su propósito: creacionales, estructurales y de comportamiento. A continuación se describen algunos de estos patrones.

- ✓ **Instancia Única:** garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.
- ✓ **Comando**: encapsula una petición en un objeto, permitiendo así entre otras cosas parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las mismas.
- ✓ **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. (11)

Luego del estudio realizado acerca de los distintos patrones de diseño existentes se pudo comprender cuáles implementa el *framework Symfony*, de ellos se mencionaron anteriormente los que serán utilizados en el desarrollo del presente trabajo, más adelante se explicará como los implementa dicho *framework* teniendo en cuenta las particularidades de la presente solución.

1.5 Métricas para validar el diseño.

Una métrica es una medida estadística que se aplica a todos los aspectos de calidad de software, permiten describir variados casos de medición. Se caracterizan por ser simples y fáciles de calcular. Con las métricas para diseño, el diseñador obtiene mayor visión interna, y proporciona la evolución del diseño a un nivel de calidad superior.

Métricas orientadas a clases.

Una clase es la unidad principal de todo sistema orientado a objeto (OO). Por lo que las medidas y métricas para una clase individual, la jerarquía de clases, y las colaboraciones de clases resultan sumamente valiosas para un ingeniero de software que necesite estimar la calidad de un diseño.

Métricas propuestas por *Lorenz y Kidd*[4].

Lorenz y Kidd dividen las métricas basadas en clases en cuatro categorías: tamaño, herencia, valores internos y valores externos. Las métricas orientadas al tamaño para las clases se centran en el recuento de atributos y operaciones para cada clase individual y los valores promedio para el sistema orientado a objetos como un todo. Las métricas basadas en la herencia se enfocan en la forma en que las operaciones se reutilizan en la jerarquía de clases. Las métricas para valores internos de clase examinan la cohesión y los aspectos orientados al código, mientras que las métricas orientadas a valores externos, examinan el acoplamiento y la reutilización. (11)

Tamaño operacional de clase (TOC): métrica que se basa esencialmente en la cantidad de funcionalidades que presenta la clase.

Relaciones entre Clases (RC): métrica que se basa en la cantidad de relaciones que presenta cada clase con el resto. Se establece relación entre dos clases cuando se tiene instancias, atributos y llamadas a funcionalidades de una clase en otra.

Para la validación del diseño en el departamento Soluciones para la Aduana se utiliza la métrica Tamaño Operacional de Clases (TOC), esta métrica mide la calidad de acuerdo a los atributos responsabilidad, complejidad de implementación y reutilización, teniendo en cuenta el promedio de operaciones por clases. Además se emplea la métrica Relación entre Clases (RC), para definir el nivel de dependencia existente entre las clases. Estas métricas serán utilizadas en el presente trabajo y estarán argumentadas en el **capítulo 3**.

1.6 Metodologías, herramientas de modelado y lenguajes para el desarrollo de software.

Para un buen desarrollo de sistemas informáticos es necesaria la adecuada selección de las tecnologías, las herramientas y los lenguajes a utilizar, pero además es muy importante conocer la metodología o el

[4] *Lorenz, M. et al, 1994] Lorenz, M. & Kidd, J. "Object-Oriented Software Metrics". Prentice Hall*. 1994. Texto recomendado para introducirse en la medición de atributos de entidades desarrolladas con metodología orientada a objetos.

modelo de desarrollo a seguir puesto que serviría de guía para el desarrollo del software y no permitiría que éste fracase.

Debido al tiempo de desarrollo del proyecto Aduana, se han ido tomando decisiones acerca de las herramientas a utilizar en el desarrollo de las aplicaciones. Seguidamente se mencionan y se explican detalladamente las tecnologías, las herramientas y los lenguajes para el desarrollo de software, así como también el modelo de desarrollo de software a seguir según las particularidades y adaptaciones del departamento Soluciones para la Aduana del centro CEIGE. Cada una de estas herramientas se encuentran argumentadas y justificadas en el documento "**Línea Base de la Arquitectura del departamento de Soluciones para la Aduana del centro CEIGE**" para el desarrollo de sus aplicaciones. (9)

1.6.1 Metodología de desarrollo.

Actualmente para el desarrollo de aplicaciones web se hace imprescindible realizar un tratamiento especial durante el proceso de desarrollo de software, se han perfeccionado las metodologías para el desarrollo de los sistemas. Cada metodología plantea especificaciones para el ciclo de vida de los proyectos, escogiendo inicialmente una de ellas según las características del producto a desarrollar.

Modelo de desarrollo de software.

En el departamento de Soluciones para la Aduana se definió un modelo de desarrollo de software a seguir en el ciclo de vida de proyectos, titulado: "**Modelo de Desarrollo de Software del Departamento de Soluciones para la Aduana**", dicha definición incluye los flujos de trabajo, encuentros y reuniones básicas, los roles y sus responsabilidades así como la interacción entre ellos. Este modelo está regido por el programa de mejora para alcanzar el nivel 2 de CMMI[5] y contiene una recopilación de buenas prácticas de diferentes metodologías (12). Además se ha venido utilizando como guía en las diferentes etapas y es por esta razón que también se utilizará en el presente trabajo.

[5] Modelo de calidad del software que clasifica las empresas en niveles de madurez.

El modelo de desarrollo planteado propone en el ciclo de vida de los proyectos 9 etapas de desarrollo (ver **figura 2**), estas se mencionan a continuación:

- ✓ Estudio preliminar: es donde comienza el proyecto, se realizan los acuerdos con el cliente y se planifica como se desarrollará lo acordado.
- ✓ Modelación del negocio: es donde se realiza todo el proceso de negocio, se modelan todos los procesos y se comienza la captura de requisitos[6].
- ✓ Requisitos: es donde se identifican los requisitos de software y a su vez se refinan y se evalúan según su complejidad y prioridad. Además se realizan especificaciones por cada uno de ellos.
- ✓ Análisis y diseño: se realiza un análisis profundo del sistema a desarrollar y se generan los artefactos del análisis y el diseño que guiará la implementación.
- ✓ Implementación: se definen los entandares de codificación con los cuales se implementara el sistema. Se implementa toda la lógica de negocio diseñada.
- ✓ Pruebas piloto: se realizan pruebas internas del sistema antes de incorporarlo al equipo central de calidad para que realice la pruebas de liberación, tratando de garantizar que se detecten la menor cantidad de no conformidades. Se resuelven las no conformidades obtenidas.
- ✓ Pruebas internas: se realizan pruebas internas del sistema antes de incorporarlo al equipo central de calidad para que realice la pruebas de liberación, tratando de garantizar las detecciones de la menor cantidad de no conformidades. Se solicita los servicios de pruebas internas al jefe de calidad del centro.
- ✓ Pruebas de liberación: se solicita los servicios de pruebas de liberación al centro de calidad.
- ✓ Despliegue: se realiza el despliegue del proyecto.

Este modelo define además las responsabilidades y actividades que tendrán asignados cada uno de los roles dentro del proyecto en desarrollo. (12) Dentro de los artefactos que se generan a partir del modelo de desarrollo se encuentran: el **modelo de procesos** (escritos en notación BPMN[7] logrando mayor comprensión para los usuarios, tanto del negocio, como para los analistas implicados y los que realizan el diseño de la solución), el **modelo conceptual**, el **diagrama de clases** del sistema, el cual compone las

[6] Condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo.
[7] *Business Process Modeling Notation*

clases, los atributos, los métodos y la visibilidad, también se reflejan las relaciones como la herencia, la composición, la agregación, la asociación, el uso y la cardinalidad. Otro artefacto es el **diagrama de secuencia orientado a actividades**, en el mismo se definen las relaciones entre las clases y los usuarios así como el flujo de cada una de las actividades, representa el flujo de las operaciones por calles, comentarios, actividades y sus relaciones más detallados, también posibilita mayor entendimiento a los programadores sobre lo que se desee implementar tanto en la interfaz de usuario como en la lógica del negocio. Este tipo de diagrama es el encargado de describir el funcionamiento de cada uno de los Requisitos Funcionales (RF). Dependiendo de la complejidad de cada requisito puede existir más de un diagrama. Además de los anteriores, se generan también el **diagrama de paquetes** y el **diagrama de componentes** donde se recogen todas las clases, componentes, interfaces, diagramas, paquetes, entre otros. La relación entre los paquetes puede ser por importación o por uso. La importación ocurre cuando se necesitan todos los elementos de determinado paquete y el uso se utiliza cuando es necesario acceder a los paquetes sin necesidad de modificarlo. Por último se realiza el **diseño de la base de datos** el cual contiene el esquema de la base de datos de donde se genera el **modelo de datos**, con las tablas del sistema y sus relaciones. (12)

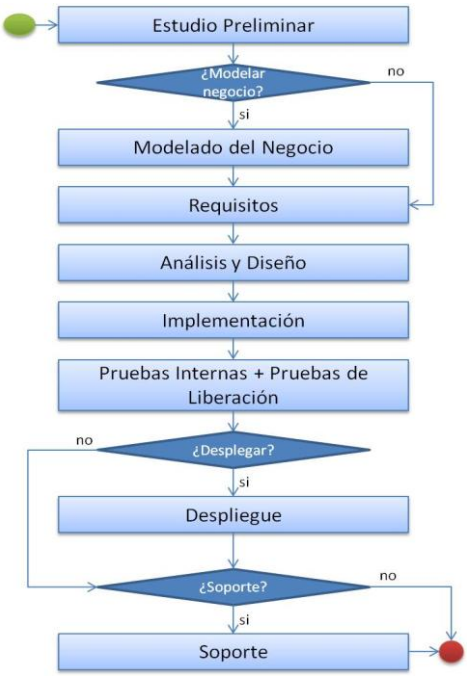

Figura 2. Ciclo de vida de los proyectos del departamento Soluciones para la Aduana.

1.6.2 Lenguaje de modelado.

En el departamento de Soluciones para la Aduana se escoge como lenguaje de modelado el lenguaje unificado de modelado (UML, por sus siglas en inglés), el cual es el más utilizado en la actualidad para modelar sistemas. Es un lenguaje gráfico que permite construir modelos para comprender mejor el sistema que se está desarrollando. Ayuda a visualizar como se quiere que sea el sistema. Permite especificar la estructura y comportamiento del sistema. Proporciona plantillas que guían la construcción del sistema y permite documentar las decisiones que se adoptan.

UML suministra mecanismos de extensibilidad, los cuales permiten a sus usuarios clarificar su sintaxis y su semántica. Puede tanto ajustarse a un sistema, proyecto o proceso de desarrollo específico si es necesario. (13)

1.6.3 Herramientas *CASE*.

Las Herramientas *CASE*[8] permiten aumentar la productividad en el desarrollo de software. Facilita un mejor intercambio de ideas con el cliente y entre los propios integrantes del equipo de desarrollo, permitiendo modelar los procesos de negocio de las empresas. Entre las más conocidas se encuentran el *Rational Rose* y el *Visual Paradigm*.

En el presente trabajo se utilizará *Visual Paradigm* puesto que es la herramienta *CASE* que se utiliza en el departamento Soluciones para la Aduana debido a las facilidades que brinda y se muestran a continuación.

Visual Paradigm

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software, análisis y diseño orientados a objetos, pruebas y despliegue. El software de modelado UML ayuda y apresura la construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y

[8] *Computer Aided Software Engineering*, Ingeniería de Software Asistida por Computadora.

generar documentación. La herramienta UML *CASE* también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.

Características:

- ✓ Producto de calidad.
- ✓ Soporta aplicaciones Web.
- ✓ Varios idiomas.
- ✓ Generación de código para *Java*[9] y exportación como HTML.
- ✓ Fácil de instalar y actualizar.
- ✓ Compatibilidad entre ediciones.
- ✓ Se integra con las siguientes herramientas *Java*:
  - *Eclipse/IBM WebSphere*.
  - *Jbuilder*.
  - *NetBeans IDE.*
  - *OracleJdeveloper.*
  - *BEA Weblogic*. (14)

1.6.4 Lenguajes de programación.

En el mundo se ha generalizado el empleo de aplicaciones web para automatizar empresas o entidades. En el desarrollo de estas se hace necesario describir un conjunto de acciones consecutivas y funcionalidades que se deben ejecutar, para lograr describir estas acciones se utilizan los lenguajes de programación.

1.6.4.1 Lenguaje de programación del lado del servidor.

Un lenguaje del lado del servidor es un lenguaje de programación que es reconocido, ejecutado e interpretado en el servidor y que se envía al cliente en un formato comprensible para él. Para definir un lenguaje del lado del servidor a utilizar se deben tener en cuenta la complejidad del mismo, si el lenguaje

[9] Lenguaje de programación orientado a objetos.

es capaz de resolver el problema, la velocidad de solución de problemas, los recursos que requiera, entre otras características.

Para el desarrollo del sistema GINA en el departamento Soluciones para la Aduana se utiliza el lenguaje PHP en su versión 5.3.5 (9).

PHP

Hypertext Preprocessor (PHP) es un lenguaje de programación interpretado, diseñado para la creación de páginas web dinámicas. El objetivo principal del lenguaje es que facilita el rápido desarrollo de las páginas por parte de los desarrolladores. Además es compatible con gran cantidad de base de datos así como *Oracle*, MySQL, entre otras. Es multiplataforma y libre, por lo que se presenta como una alternativa de fácil acceso para todos y permite las técnicas de programación orientada a objetos. (15)

1.6.4.2 Lenguaje de programación del lado del cliente.

HTML

Hipertext Markup Lenguage o Lenguaje de marcado de hipertexto (HTML) se utiliza para crear páginas *web*, el lenguaje HTML es un estándar reconocido en todo el mundo y cuyas normas define un organismo sin ánimo de lucro llamado *World Wide Web Consortium* (http://www.w3.org/) más conocido como W3C. Como se trata de un estándar reconocido por todas las empresas relacionadas con el mundo de Internet, una misma página HTML se visualiza de forma muy similar en cualquier navegador[10] de cualquier sistema operativo (SO)[11].

El propio W3C define el lenguaje HTML como "un lenguaje reconocido universalmente y que permite publicar información de forma global". Desde su creación, el lenguaje HTML ha pasado de ser un lenguaje utilizado exclusivamente para crear documentos electrónicos a ser un lenguaje que se utiliza en muchas aplicaciones electrónicas como buscadores, tiendas online y banca electrónica. (16)

---

[10] Aplicación que interpreta la información de archivos y sitios web ya sea en un servidor en internet o en un servidor local.

[11] Programa o conjunto de programas que en un sistema informático gestiona los recursos de hardware y provee servicios a los programas de aplicación, y se ejecuta en modo privilegiado respecto de los restantes.

CSS

Cascading Style Sheets (CSS): hojas de estilo en cascada es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML. CSS es la mejor forma de separar los contenidos y su presentación. Es imprescindible para crear páginas web complejas.

La separación de los contenidos y su presentación tiene numerosas ventajas, puesto que obliga a crear documentos HTML/XHTML bien definidos y con significado completo (también llamados *"*documentos semánticos*"*). Además, mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes.

Mientras que el lenguaje HTML/XHTML se utiliza para marcar los contenidos, es decir, para designar lo que es un párrafo, lo que es un titular o lo que es una lista de elementos, el lenguaje CSS se utiliza para definir el aspecto de todos los contenidos, es decir, el color, tamaño y tipo de letra de los párrafos de texto, la separación entre titulares y párrafos, la tabulación con la que se muestran los elementos de una lista, etc. (17)

JavaScript

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas. Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario. Técnicamente, *JavaScript* es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con *JavaScript* se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios. La integración de *JavaScript* y XHTML es muy flexible, ya que existen al menos tres formas para incluir código *JavaScript* en las páginas web.

Desde su aparición, *JavaScript* siempre fue utilizado de forma masiva por la mayoría de sitios de Internet. La aparición de *Flash* disminuyó su popularidad, ya que *Flash* permitía realizar algunas acciones

imposibles de llevar a cabo mediante *JavaScript*. Sin embargo, la aparición de las aplicaciones AJAX[12] programadas con *JavaScript* le ha devuelto una popularidad sin igual dentro de los lenguajes de programación web. En cuanto a las limitaciones, *JavaScript* fue diseñado de forma que se ejecutara en un entorno muy limitado que permitiera a los usuarios confiar en la ejecución de los *scripts*[13]. (18)

1.6.5 *Framework* de desarrollo.

Un *framework* es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, en base a la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto. (19)

Un *framework* o marco de trabajo simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Además, un *framework* proporciona estructura al código fuente, forzando al desarrollador a crear código más legible y más fácil de mantener. Por último, un *framework* facilita la programación de aplicaciones, dado que encapsula operaciones complejas en instrucciones sencillas. (20)

Framework Symfony

Symfony es un *framework* de desarrollo completo diseñado para optimizar el desarrollo de aplicaciones web. Es creado completamente con PHP 5 patentado bajo licencia MTI[14]. Aporta clases y herramientas orientadas a reducir el tiempo de desarrollo de una aplicación web, añade una nueva capa por encima de PHP y es independiente del sistema gestor de base de datos. También proporciona una estructura al código fuente y permite encapsular operaciones complejas en instrucciones sencillas. Está basado en el patrón arquitectónico MVC (ver **figura 3**).

[12] Acrónimo de ***A****synchronous* ***J****avaScript* ***A****nd* ***X****ML,* técnica o tecnología de desarrollo web para crear aplicaciones interactivas.
[13] Programa usualmente simple, que por lo regular se almacena en un archivo de texto plano.
[14] Licencia de Software Libre originaria del Instituto de Tecnología de *Massachusetts*

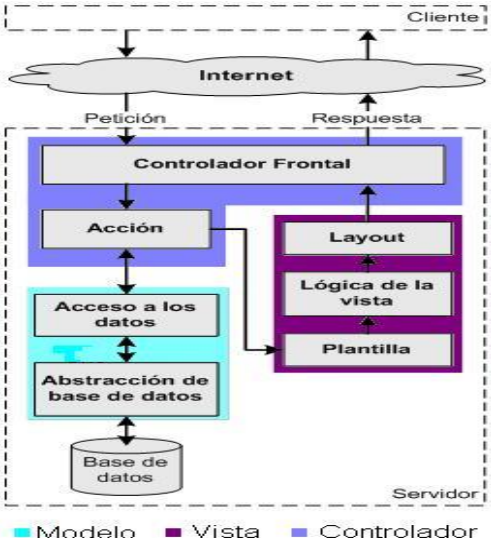

Figura 3. MVC en el *framework Symfony.*

El framework Symfony automatiza la mayoría de elementos comunes de los proyectos web, como por ejemplo:

- ✓ La capa de internacionalización que incluye el *framework Symfony* permite la traducción de los datos y de la interfaz, así como la adaptación local de los contenidos.
- ✓ La capa de presentación utiliza plantillas y *layouts*[15] que pueden ser creados por diseñadores HTML sin ningún tipo de conocimiento del *framework* de desarrollo. Los *helpers*[16] incluidos permiten minimizar el código utilizado en la presentación, puesto que encapsulan grandes bloques de código en llamadas simples a funciones.
- ✓ Los formularios incluyen validaciones automatizadas y relleno automático de datos, lo que asegura la obtención de datos correctos y mejora la experiencia del usuario.
- ✓ Los datos incluyen mecanismos de escape que permiten una mejor protección contra los ataques producidos por datos corruptos.

[15] Almacena el código HTML que es común a todas las páginas de la aplicación, para no tener que repetirlo en cada página.
[16] Funciones de PHP que devuelven código HTML y que se utilizan en las plantillas.

- ✓ La gestión de la *caché*[17] reduce el ancho de banda utilizado y la carga del servidor.
- ✓ La autenticación y la gestión de credenciales simplifican la creación de secciones restringidas y la gestión de la seguridad de usuario.
- ✓ El sistema de enrutamiento y las URL[18] limpias permiten considerar a las direcciones de las páginas como parte de la interfaz, además de estar optimizadas para los buscadores.
- ✓ El soporte de e-mail incluido y la gestión de APIs[19] permiten a las aplicaciones web interactuar más allá de los navegadores.
- ✓ Los listados son más fáciles de utilizar debido a la paginación automatizada, el filtrado y la ordenación de datos.
- ✓ Los *plugins* y las factorías permiten realizar extensiones a medida del *framework Symfony*.
- ✓ Las interacciones con *Ajax* son muy fáciles de implementar mediante los *helpers* que permiten encapsular los efectos *JavaScript* compatibles con todos los navegadores en una única línea de código. (20)

En el desarrollo del sistema GINA se utiliza el *framework* de desarrollo *Symfony* en su versión 1.2.8, dado que presenta amplia documentación tanto de su utilización como de su núcleo, también tiene gran aceptación a nivel mundial y tendencia a utilizarlo por sus facilidades de optimización. Además es válido destacar que está basado en el estilo arquitectónico MVC, lo que posibilita que cada capa del *framework* no dependa de otras. Otro aspecto importante es que está desarrollado completamente con PHP 5, lenguaje muy reconocido y fácil de aprender. Es Independiente del sistema gestor de bases de datos. Sigue la mayoría de mejores prácticas y patrones de diseño para la *web*.

Plugins

En el *framework* S*ymfony* existe un mecanismo que facilita la reutilización de código, que permite además encapsular ese código en una clase y se almacena en algún directorio *lib/* del proyecto, este mecanismo

[17] Conjunto de datos duplicados de otros originales:
[18] Localizador de recursos uniforme, secuencia de caracteres, de acuerdo a un formato modélico y estándar, que se usa para nombrar recursos en Internet para su localización o identificación.
[19] Interfaz de programación de aplicaciones, conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

se ejecuta a través de los *plugins*. Permiten agrupar todo el código disperso por diferentes archivos y utilizar este código en otros proyectos. También permiten encapsular filtros[20], clases, *mixins*[21], archivos de configuración, *helpers*, esquemas, tareas, módulos y extensiones para el modelo.

Plugin:

- ✓ Extensión encapsulada.
- ✓ Permite aprovechar componentes ya desarrollados anteriormente dentro del proyecto.
- ✓ Permite añadir al núcleo del *framework Symfony* extensiones realizadas con anterioridad. (20)

Estructura de un Plugin

Los *plugins* se crean mediante el lenguaje PHP. Si se entiende la forma en la que se estructura una aplicación, es posible comprender la estructura de un plugin.

Estructura de archivos de un plugin

El directorio de un plugin se organiza de forma muy similar al directorio de un proyecto. Los archivos de un plugin se deben organizar de forma adecuada para que el *framework Symfony* pueda cargarlos automáticamente cuando sea necesario. La **figura 4** muestra la estructura de carpetas y archivos de un *plugin* en el *framework Symfony*. La carpeta principal (color gris) del directorio de archivos indica el nombre del plugin, dentro de esta carpeta se encuentra todo lo referente al *plugin* creado. La carpeta *config/* del segundo nivel contienen las configuraciones del plugin o componente. Dentro de la carpeta *lib*[22]*/* se encuentran todas las clases principales pertenecientes al componente y la creadas por el propio *framework Symfony*. La carpeta *modules* contiene el/los módulos que componen el *plugin* y dentro de uno de estos las carpetas *action, config* y *templates*, donde se guardan las funcionalidades del componente, el archivo *view.yml*[23] de la aplicación y las plantillas respectivamente. Por último la carpeta web contiene las capetas *css*, *images* y *js*, en estas se encuentran los archivos de la vista o interfaz de usuario.

[20] Mecanismo de seguridad en *Symfony*, por el que debe pasar cada petición antes de ejecutar la acción.
[21] Grupo de métodos o funciones que se juntan en una clase para que otras clases hereden de ella.
[22] Almacena las clases y librerías externas en *Symfony*. Se suele guardar todo el código común a todas las aplicaciones del proyecto
[23] Es donde se define las opciones de la vista en una aplicación *Symfony*.

Generalmente se utiliza solamente la carpeta *js* puesto que es la que contiene los archivos **.js*; las imágenes y hojas de estilo pueden estar en la carpeta web del directorio principal de la aplicación.

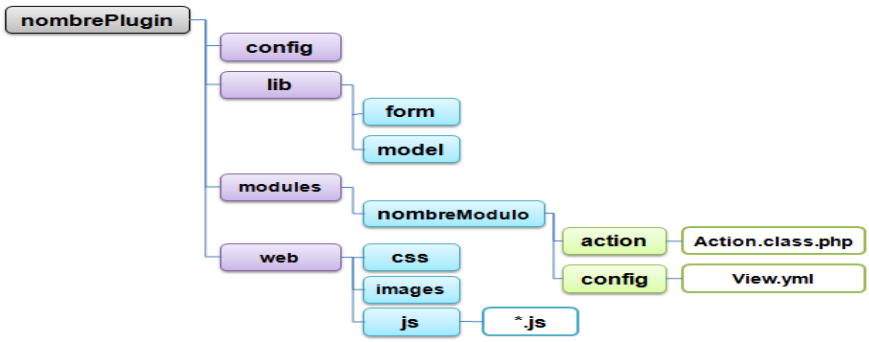

Figura 4. Estructura de carpetas de un *plugin.*

Para el desarrollo del componente que se pretende obtener se empleará este mecanismo del *framework Symfony*. De esta forma facilitaría su reutilización, proporcionando a otras aplicaciones del sistema GINA el empleo del mismo.

Para acceder al plugin desde cualquier módulo del proyecto en el *framework Symfony* sería habilitando archivo *setting.yml* que se encuentra en el directorio *nombreAplicacion/config/settings.yml, como sigue:*

```
all:
.settings:
enabled_modules: [default, sfMiPluginModule]
```

Ext JS

Ext JS empezó siendo un conjunto de librerías y extensiones para *Yahoo! User Interface* (YUI). Con el tiempo se convirtió en un *framework* de desarrollo independiente y a principios de 2007 se creó una compañía para comercializar y dar soporte al mismo. Tiene dos tipos de licencias, Licencia Pública General Reducida (LGPL) y comercial. Básicamente, *Ext JS* se utiliza en los proyectos que se deseen; pero solo se puede obtener soporte si se tiene licencia comercial. (21)

Es un *framework* que permite potenciar la capa Interfaz de Usuario (IU) de *Javascript* en las aplicaciones. Además ayuda a la comunicación entre el cliente y el servidor mediante JSON[24], AJAX y XML[25]. *Ext JS* es neutral al lenguaje que se use en el servidor. Siempre que el resultado se envíe a la página en el formato adecuado, *Ext JS* no se preocupará de lo que pase en el servidor. Utilizar un motor de *render*[26] como *Ext JS* permite además tener algunas **ventajas**:

- ✓ **Existe gran balance entre el Cliente y el Servidor:** la carga de procesamiento se distribuye, permitiendo que el servidor, al tener menor carga, pueda manejar más clientes al mismo tiempo.
- ✓ **Comunicación asincrónica[27]:** en este tipo de aplicación el motor de *render* puede comunicarse con el servidor sin necesidad de estar sujeta a un *clic* o una acción del usuario, dándole la libertad de cargar información sin que el cliente se dé cuenta.
- ✓ **Eficiencia de la red.** el tráfico de red puede disminuir al permitir que la aplicación elija que información desea transmitir al servidor y viceversa, sin embargo la aplicación que haga uso de la pre-carga de datos puede que revierta este beneficio por el incremento del tráfico. (21)

En la realización de la solución se pretende utilizar *ExtJS* en su versión 3.0 debido a las ventajas que ofrece como por ejemplo su integración con el *framework Symfony* facilita la independencia entre la capa de presentación y la controladora.

1.6.6 Sistema gestor de base de datos.

Un sistema gestor de base de datos (SGBD) es una colección de programas cuyo objetivo es servir de interfaz entre la base de datos, el usuario y las aplicaciones. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. Un SGBD permite definir los datos a distintos niveles de abstracción y manipular dichos datos, garantizando la seguridad e integridad de los mismos. Algunos ejemplos de SGBD son Oracle, DB2, PostgreSQL, MySQL, MS SQL Server, etc.

[24] Estándar utilizado para que un script de cliente se comunique con un servidor.
[25] Metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C).
[26] Utilizado en ExtJS para insertar el formato HTML
[27] Suceso que no tiene lugar en total correspondencia temporal con otro suceso.

Oracle es un Sistema Gestor de Base de Datos con características objeto-relacionales, que pertenece al modelo evolutivo de Sistema Gestor de Base de Datos. Sus características principales son las siguientes:

- ✓ Entorno cliente/servidor.
- ✓ Gestión de grandes base de datos.
- ✓ Usuarios concurrentes.
- ✓ Alto rendimiento en transacciones.
- ✓ Sistemas de alta disponibilidad.
- ✓ Disponibilidad controlada de los datos de las aplicaciones.
- ✓ Gestión de la seguridad.
- ✓ Autogestión de la integridad de los datos.
- ✓ Opción distribuida.
- ✓ Portabilidad.
- ✓ Compatibilidad.. (22)

En el presente trabajo se utiliza el Sistema Gestor de Base de Datos Oracle en su versión *11g debido a las facilidades que brinda*. El SGBD Oracle es considerado uno de los más potentes a nivel mundial, es fabricado por *Oracle Corporation* y utiliza la arquitectura cliente/servidor. Ha incorporado en su sistema el modelo objeto-relacional, pero al mismo tiempo garantiza la compatibilidad con el tradicional modelo relacional de datos. Así ofrece un servidor de bases de datos híbrido. Es uno de los más conocidos y ha alcanzado un buen nivel de madurez y de profesionalidad. Se destaca por su soporte de transacciones, estabilidad y escalabilidad.

1.7 Conclusiones parciales.

A partir del estudio realizado acerca de los sistemas existentes se concluye que no son factibles para ser empleados en la solución debido a que no cumplen con las legislaciones de la AGR, no son compatibles con las políticas que sigue el país de migrar a plataforma libre, son sistemas privativos, costosos y solo están adaptados a los países donde fueron desarrollados. La utilización del Modelo de desarrollo y arquitectura definida por el departamento soluciones para la Aduana del CEIGE posibilitó que se escogieran las tecnologías y herramientas para ser utilizadas en la solución de la presente investigación.

Capítulo 2: Fundamentación teórica.

2.1 Introducción.

En este capítulo se presentan las características del sistema, se desarrolla el diseño y se realiza la implementación que dará solución al problema planteado en el presente trabajo. Durante el diseño se obtendrán los distintos diagramas que guiará la implementación del sistema. Durante la implementación se obtendrá el código fuente de algunas clases, tanto de la parte del negocio como de las interfaces gráficas, así como también, se muestran algunas pantallas de la aplicación y estándares de codificación utilizados.

2.2 Descripción de la solución.

De forma general el sistema GINA integra las aplicaciones que informatizan los procesos de la AGR. En trabajos previos, se identificaron los principales problemas que convergen en los procesos aduanales. En este caso, se puede mencionar la creación de un componente que permita gestionar el registro histórico de los medios de transporte internacional, facilitando a los especialistas del área de lucha contra el fraude u otra área de la AGR mayor accesibilidad y organización cuando se controle un MTI.

Actualmente en la AGR no se cuenta con una aplicación que informatice los procesos involucrados con la solución del presente trabajo, siendo el desarrollo de un componente que centralice el registro histórico de un MTI y sea utilizado por varios de los subsistemas que integran el sistema GINA de la AGR.

Como resultado final el componente propuesto debe establecer la comunicación entre las aplicaciones del sistema GINA y centralizar en este sistema el registro histórico del recorrido de los medios de transporte internacional en el país.

2.2.1 Subsistemas involucrados en la solución propuesta.

Entre los subsistemas que se desarrollan en el sistema GINA se encuentra el subsistema Despacho de Medios de Transporte Internacional para la gestión de los MTI, estos a su vez, son controlados por diferentes aéreas de la AGR, razón por la cual se pretende desarrollar con el presente trabajo el componente Historial de MTI favoreciendo su posterior consulta por los demás subsistemas. Algunos de

los subsistemas que están relacionados con lo anteriormente mencionado son el Control de MTI y Asuntos Legales.

Despacho de Medios de Transporte Internacional (MTI).

El subsistema Despacho de Medios de Transporte Internacional permite la recepción de todos los medios de transporte internacionales que entran y salen del país en cualquier momento, posibilitando la gestión de estos dentro de las Aduanas del país y favoreciendo a los especialistas de determinada área a tener mayor control de los mismos cuando sea necesario. El componente Historial de MTI a desarrollar sería de gran importancia para su utilización en este subsistema puesto que permitirá gestionar todos los hechos realizados por el MTI.

Control de Medios de Transporte Internacional.

El subsistema Control de Medios de Transporte Internacional es un módulo del subsistema Lucha Contra el Fraude (LCF). Es el encargado de realizar las inspecciones e investigaciones a todos los MTI que entran y salen del país. Con este proceso se identifican las infracciones cometidas por el mismo y el control aplicado; para facilitar el trabajo de los especialistas de esta área lo más conveniente sería la utilización del componente Historial de MTI, pues le brindará mayor información acerca de todas las operaciones realizadas por los MTI en viajes anteriores ya sean positivas o negativas. Además, se ganaría tiempo y organización cuando se realice dicho proceso.

Asuntos Legales.

En el subsistema Asuntos Legales es donde se lleva a cabo la gestión documental de la AGR. Se realizan las sanciones, procedimientos de revisión, entre otras tareas que pueden aplicarse a un MTI en determinado momento ya sea por alguna infracción cometida por el mismo. Para ello se hace necesario utilizar el componente que se obtendrá en la solución propuesta en caso de que se pretenda aplicar alguna multa u otra sanción al MTI.

2.3 Requisitos del sistema.

El modelo de desarrollo propuesto por el departamento de Soluciones para la Aduana define varias etapas, una de ellas es el diseño, pero para poder pasar a esta etapa, primeramente se identificaron los requisitos funcionales del software (RF).

A continuación (ver **tabla 1**) se especifican los requisitos identificados más importantes:

No.	Requisito		Descripción
1	**Obtener hechos de un MTI**.	**Obtener árbol de Hechos.**	Dado el identificador de un MTI, el sistema busca en la base de datos el historial de hechos positivos o negativos de un MTI realizados en viajes anteriores, ya sea de entrada, salida o permanencia en el país.
		Obtener datos generales MTI	Dado el identificador de un MTI el sistema muestra en pantalla los datos general del MTI
		Obtener detalles de hecho.	Dado el identificador de un "hecho" asociado a un MTI el sistema muestra por pantalla los detalles del hecho seleccionado.
2	**Registrar hecho de un MTI.**	**Registrar hecho aeronave.**	Dado el identificador de una aeronave, el sistema registra en la base de datos algún hecho ocurrido que esté relacionado con la misma.
		Registrar hecho buque	Dado el identificador de un buque, el sistema registra en la base de datos algún hecho ocurrido que esté relacionado con el mismo.
		Registrar hecho embarcación de recreo.	Dado el identificador de una embarcación de recreo, el sistema registra en la base de datos algún hecho ocurrido que esté relacionado con la misma.

Tabla 1. Requisitos funcionales del componente Historial de MTI.

2.4 Patrones de diseño implementados.

Para la implementación con el *framework Symfony* se utilizan varios patrones, situándolos en las capas del modelo y el controlador. Seguidamente se describen dichos patrones.

Patrones GRASP

- ✓ **Alta cohesión**: el *framework Symfony* permite la organización del trabajo en cuanto a la estructura del proyecto y la asignación de responsabilidades con una alta cohesión. Esta caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Se evidencia que

las clases son reutilizables mediante la agrupación de funcionalidades. La clase *Actions*[28], es formada por varias funcionalidades estrechamente relacionadas, siendo la responsable de definir las acciones para las plantillas y colaborar con otras para realizar diferentes operaciones e instanciar objetos.

- ✓ **Bajo acoplamiento:** asigna la responsabilidad de mantener el control sobre el flujo de eventos del sistema, a clases específicas. Es muy utilizado para mantener organizadas todas las funcionalidades, posibilitando agrupar los procedimientos semejantes, para hacer menos engorroso el proceso de validación y la seguridad. El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y recurre a ellas. La clase *Actions* hereda de *sfActions* únicamente para alcanzar un bajo acoplamiento. Las clases que implementan la lógica del negocio y de acceso a datos se encuentran en el modelo, las cuales que no tienen asociaciones con las de la vista o el controlador, lo que proporciona que la dependencia en este caso sea baja.
- ✓ **Creador:** guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental es encontrar un creador que se conecte con el objeto producido en cualquier evento. En la clase *Actions* se encuentran las acciones definidas para el sistema y se ejecutan en cada una de ellas. En dichas acciones se crean los objetos de las clases que representan las entidades, lo que evidencia que la clase *Actions* es "creador" de dichas entidades.
- ✓ **Experto:** la responsabilidad de realizar una labor es de la clase que tiene o puede tener los atributos. Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada. Es uno de los patrones más utilizados en el *framework Symfony*. La librería *Propel*[29], para mapear la Base de Datos y para realizar su capa de abstracción en el modelo, encapsula toda la lógica de los datos y genera las clases con todas las funcionalidades comunes de las entidades. Las clases de abstracción de datos poseen un grupo de funcionalidades que están relacionadas directamente con la entidad que representan y contienen la información necesaria de la tabla que representan.

[28] Es la clase controladora utilizada en el *framework Symfony* para realizar las funcionalidades del sistema.
[29] *Kit* de mapeo objeto-relacional (ORM) de código abierto escrito en PHP. Es además una parte integral del *framework Symfony*.

- ✓ **Controlador:** evento generado por actores externos. Se asocian con operaciones del sistema, como respuestas a los eventos del sistema, tal como se relacionan los mensajes y los métodos. Normalmente un controlador delega en otros objetos el trabajo que se necesita hacer, coordina o controla la actividad. Se emplea la página "*lcf_dev.php"* como controlador frontal, que se encarga de tramitar todas las peticiones que se realizan a través de ella para direccionarla al resto de las plantillas. Además todas las peticiones *web* son manipuladas por un solo controlador frontal (*sfActions*), que es el punto de entrada único de toda la aplicación en un entorno determinado. (23)

Patrones GOF

- ✓ **Instancia única (*singleton*):** solamente admite una instancia de una clase. Los objetos necesitan un único punto de acceso global. Es el caso del controlador frontal, donde hay una llamada a la función *sfContext::getInstance()* que garantiza que siempre se acceda a la misma instancia.
- ✓ **Comando (*comand*):** este patrón permite solicitar una operación a un objeto sin conocer realmente el contenido de esta operación, ni el receptor real de la misma. Para ello se encapsula la petición como un objeto. Se observa en la clase *sfWebFrontController*, en el método *dispatch().* Esta clase está por defecto y es la encargada de establecer el módulo y la acción que se va a usar según la petición del usuario.
- ✓ **Fábrica (*factory*):** tiene como propósito crear objetos, permitiendo al sistema identificar que clase se debe instanciar en tiempo de ejecución. Asegura que sólo exista una instancia de una clase específica en un sistema a desarrollar y la creación de un mecanismo de acceso global a dicha instancia. Por ejemplo, cuando el *framework* necesita crear un nuevo objeto, busca en la definición el nombre de la clase que se debe utilizar para esta tarea. (23)

2.5 Modelo de diseño.

En el diseño se modela el sistema y se estructura de acuerdo a la arquitectura definida para que soporte todos los requisitos funcionales y no funcionales. Para poder dar el paso al diseño es necesario previamente realizar un análisis del sistema que se desea obtener, o sea, el modelo de análisis, que proporciona una comprensión detallada de los requisitos.

El diseño tiene como propósito:

- ✓ Comprender los aspectos relacionados con los requisitos y restricciones de los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución, concurrencia y tecnologías de interfaz de usuario.
- ✓ Crear una entrada adecuada y un punto de partida para la implementación, capturando los requisitos o subsistemas individuales, interfaces y clases.
- ✓ Descomponer la implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo. (24)

2.5.1 Diagrama de clases del diseño.

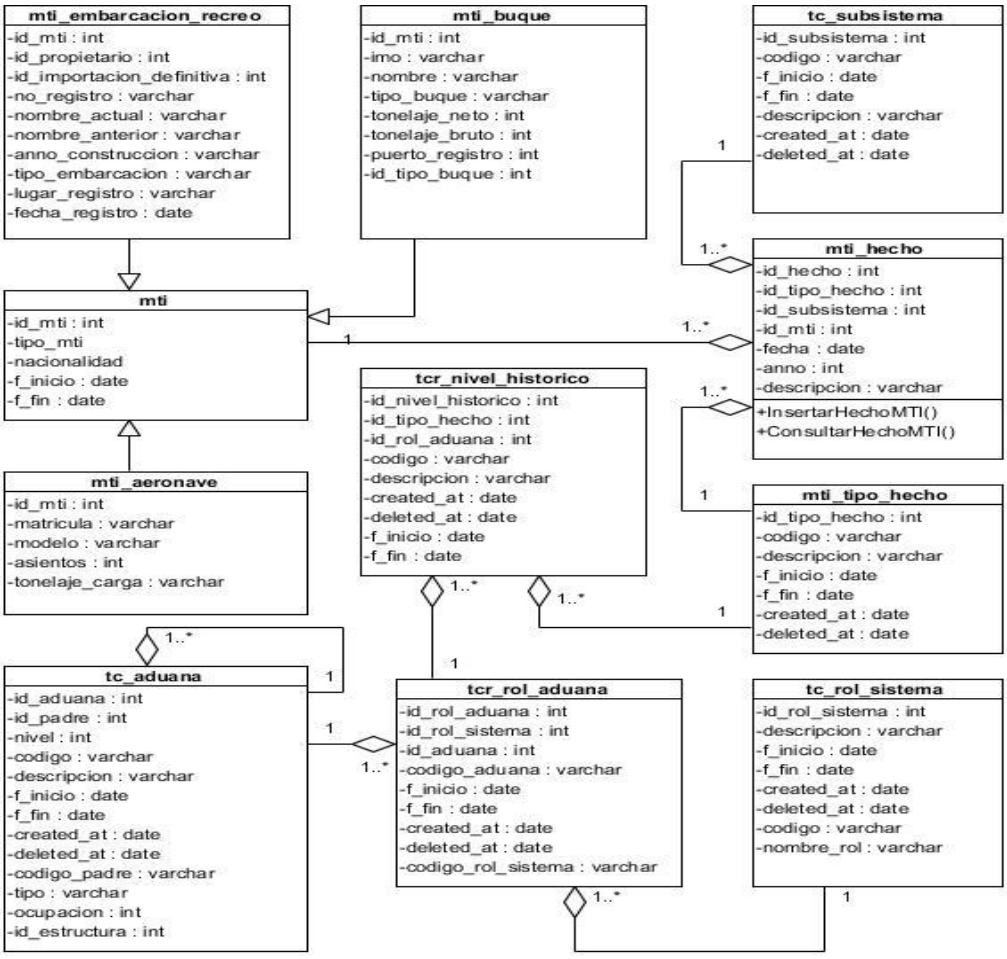

Figura 5. Diagrama de clases del diseño.

En la **figura 5** se muestra el diagrama de clases planteado para dar solución al problema. En el mismo se recogen todas las clases, sus relaciones y atributos, involucrados para el desarrollo del componente propuesto. Las clases *mti_hecho, mti_tipo_hecho* y *tcr_nivel_historico* son tablas que pertenecen al esquema historial de MTI. Las clases tc*_subsistema, tc_aduana, tcr_rol_aduana* y *tc_rol_sistema* son entidades nomencladoras y están fuera del esquema historial MTI. Por último, las clases *mti*, *mti_embarcacion_recreo*, *mti_aeronaves* y *mti_buques* son las entidades que contienen los medios de transporte internacional en la AGR.

2.5.2 Diagrama de paquetes.

La **figura 6** muestra el diagrama de paquetes del componente Historial de MTI. El mismo tiene contenido los paquetes involucrados en el componente, relacionados entre ellos. El paquete MTI contiene tres paquetes internos los cuales son: módulos del subsistema MTI, éste, al igual que los demás subsistemas del sistema GINA (LCF y Asuntos Legales), hacen uso del paquete principal *sfHistorialMti*. En el paquete *Symfony Lib* se encuentran las distintas clases del núcleo del *framework* S*ymfony*. Y por último el paquete *sfHistorialMti* hace uso de los paquetes: TC, *sfUtil*, Administración y S*ymfony Lib*, además contiene cuatro paquetes que están detallados seguidamente.

- ✓ *Config*: contiene las configuraciones de la aplicación, ya sean de la base de datos como los servicios brindados a otras aplicaciones.
- ✓ *Web*: agrupa todas las clases de *javascript*, clases css y las imágenes para la presentación de la aplicación.
- ✓ *Lib*: se encuentran las clases del componente.
- ✓ *sfHistorial*: tiene como misión establecer la comunicación entre las aplicaciones del sistema GINA y centralizar la información del historial de MTI.

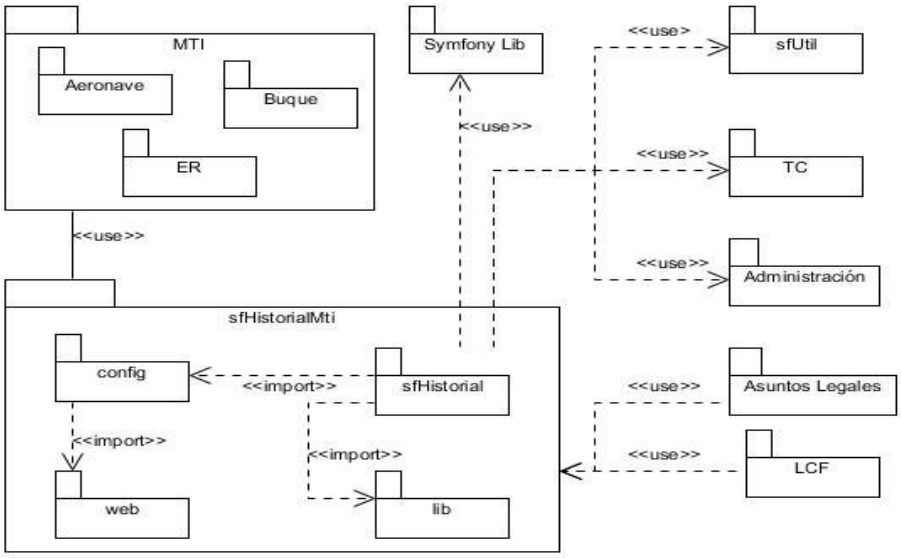

Figura 6. Diagrama de Paquetes Historial MTI.

En la **figura 7** se muestra como está estructurado el plugin *sfHistorialMti,* desde el directorio principal hasta los archivos del plugin. GINA es el nombre del proyecto, en el directorio del segundo nivel se encuentran las carpetas principales del proyecto. La carpeta plugin sería donde se encuentran los componentes, en este caso, se visualiza el plugin *sfMtiPlugin* que consta de 4 carpetas las cuales se encuentran en el **epígrafe 1.6.5** cuando se explica la **figura 4**.

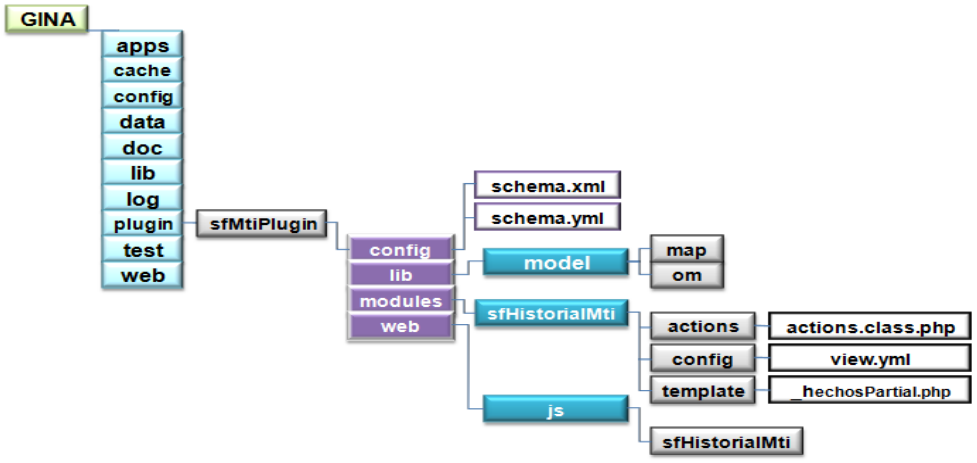

Figura 7. Estructura física en *Symfony* del componente Historial de MTI.

2.5.3 Diagrama de secuencia orientado a actividades.

A continuación se muestra el diagrama de secuencias orientado a actividades de componentes visuales correspondiente al RF: Obtener datos generales de MTI (Ver **figura 8**). Donde se describe el proceso de obtención de los datos generales de un MTI. Este diagrama es protagonizado por la interfaz de usuario y el controlador donde se visualiza como el usuario interactúa con el negocio mediante la interfaz gráfica, describe además otros componentes útiles para el diseñador.

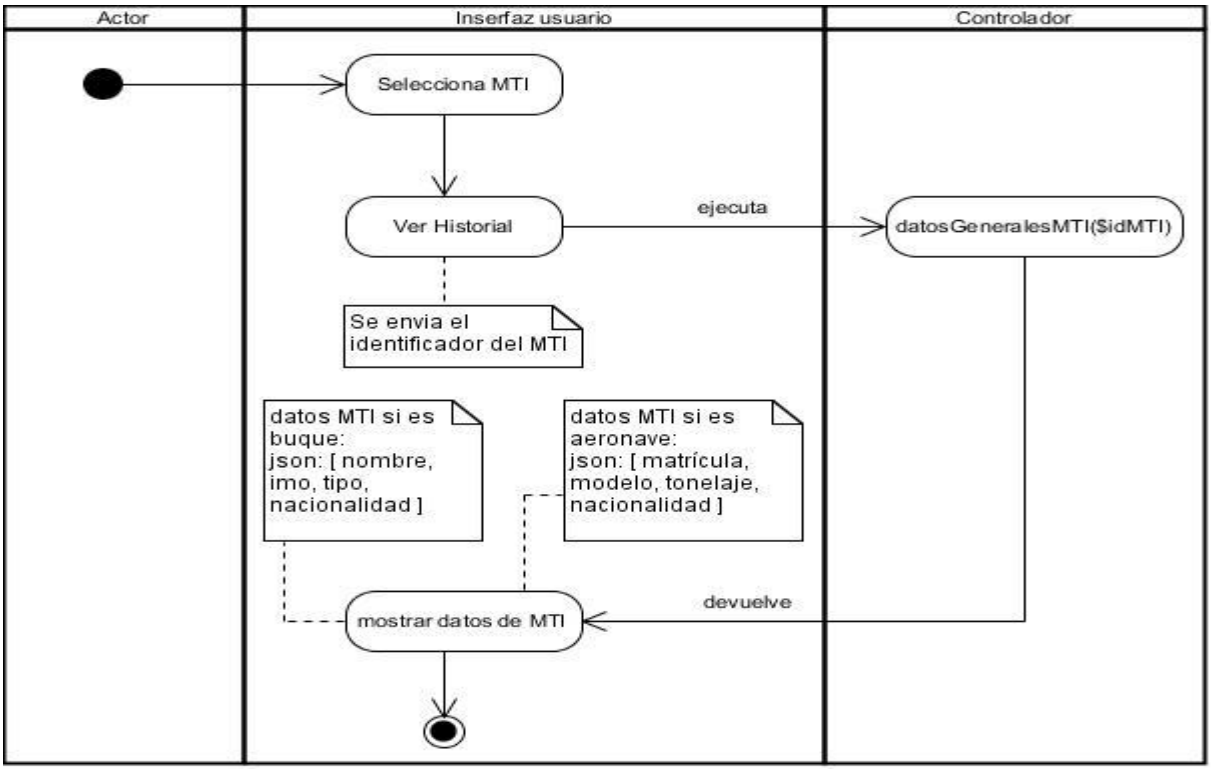

Figura 8. Diagrama de secuencias orientado a actividades de los componentes visuales.

Seguidamente se muestra el diagrama de secuencias orientado a actividades del negocio generado en correspondencia con el RF: Obtener datos generales de MTI (ver **figura 9**). El mismo a diferencia de la **figura 8**, se detallan los pasos de la lógica de negocio que debe seguir el programador. Las calles se componen por las clases que interactúan en la funcionalidad o las funcionalidades correspondientes al RF. En este diagrama se comienza a partir de seleccionar ver el historial de determinado MTI, luego es

enviado el identificador del MTI siendo capturado por la clase *actions.class.php*, en esta se procede a obtener los datos generales del MTI consultando a la base de datos, posteriormente se crea una instancia del MTI en dependencia del tipo (Buque, Aeronave o Embarcación de Recreo), para finalmente ser enviado en formato JSON hacia Interfaz.

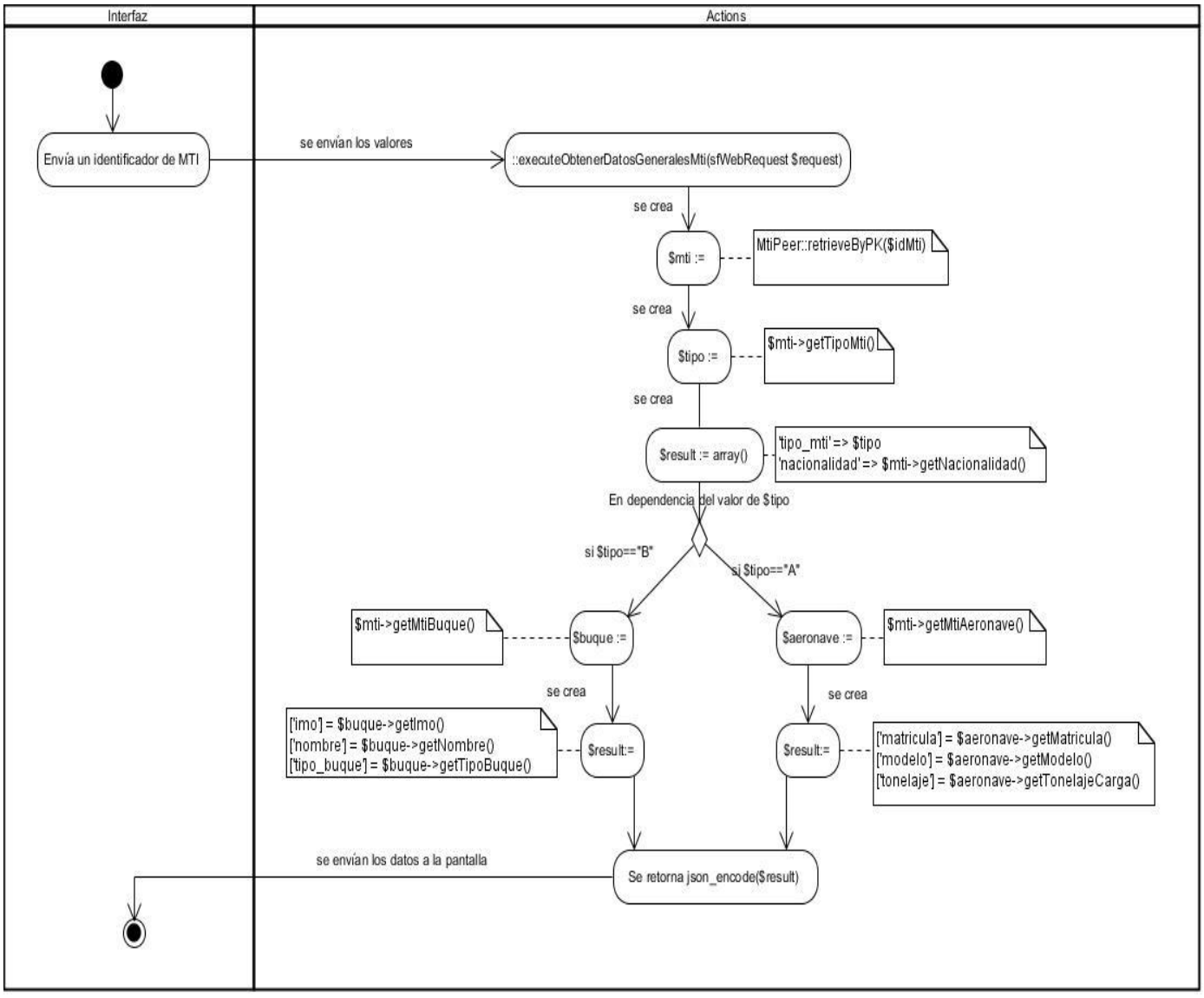

Figura 9. Diagrama de secuencias orientado a actividades del negocio.

Ver otros diagramas en el anexo 2.

2.5.4 Diseño de la Base de Datos.

A continuación se muestra el Diagrama Entidad-Relación correspondiente al componente Historial MTI (ver **figura 10**), sus 11 tablas, las cuales tienen implícitas sus atributos y relaciones entre ellas.

Las tablas *mti_hecho, mti_tipo_hecho* y *tcr_nivel_historico* pertenecen al esquema historial de MTI. Las tablas tc*_subsistema, tc_aduana, tcr_rol_aduana* y *tc_rol_sistema* son entidades nomencladoras y están fuera del esquema historial MTI. Y las tablas *mti*, *mti_embarcacion_recreo*, *mti_aeronaves* y *mti_buques* son las encargadas de almacenar los datos de los medios de transporte internacional de la AGR.

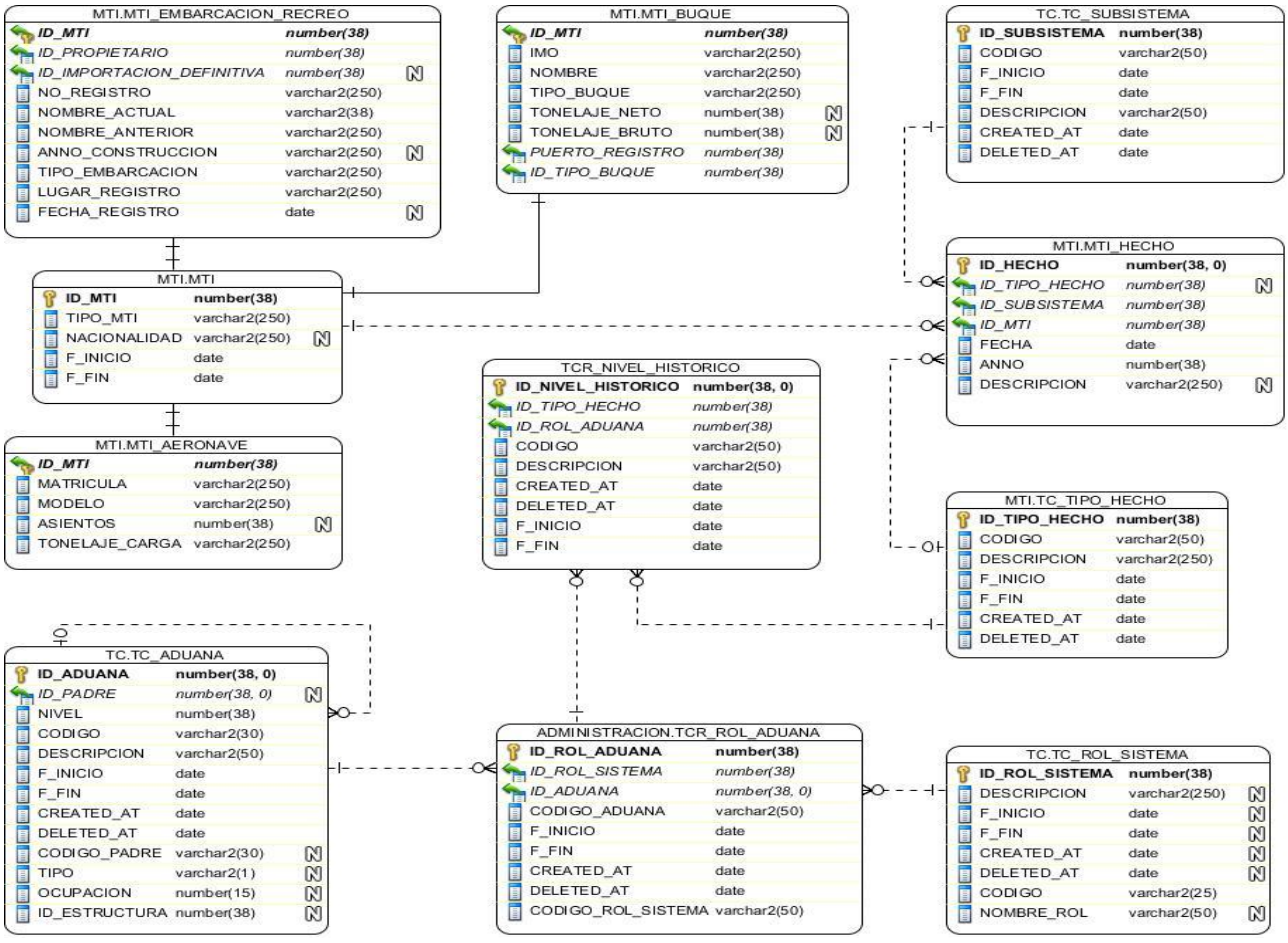

Figura 10. Diagrama Entidad-Relación.

En el anexo #1 se encuentran las descripciones de las tablas más importantes.

2.6 Implementación del sistema.

Partiendo del modelo del diseño obtenido se procede a la implementación de la solución propuesta. La mayor parte de la arquitectura del sistema es definida en el diseño, para luego desarrollar lo definido anteriormente, ya sean las clases, código fuente y ejecutables. El modelo de implementación describe cómo los elementos del diseño y las clases, se implementan en términos de componentes. Además describe cómo se organizan estos componentes de acuerdo a los mecanismos de estructuración en el entorno de implementación y los lenguajes de programación utilizados.

2.6.1 Modelo de despliegue.

El diagrama de despliegue (ver **figura 11**) es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo.

Figura 11. Diagrama de despliegue.

2.6.2 Diagrama de componentes.

El diagrama de componentes es donde se representa de forma general las relaciones entre los componentes físicos utilizados para el desarrollo de una aplicación. Generalmente un componente físico puede contener clases, ejecutables, documentos, binarios, bibliotecas, ficheros o tablas. Seguidamente se muestra el diagrama de componentes obtenido luego del desarrollo de la solución (ver **figura 12**).

El mismo contiene el paquete **interfaces *JavaScript***, donde recoge las clases de interfaz de usuario de la aplicación. El paquete de **abstracción de datos *Symfony*** es el encargado de agrupar las clases de configuración del *framework Symfony.* El paquete **común** contiene las clases *CSS* y *JavaScript* pertenecientes al *framework Ext JS.* El paquete de **configuración** es el encargado de contener los archivos de configuración del acceso a datos databases.yml y propel.ini; además del fichero de

configuración de las vistas *view.yml*. El paquete ***Templates*** contiene la plantilla *_hechoPartial.php* empleada para mostrar detalladamente la descripción del historial de MTI. Además, este diagrama contiene el componente ***Actions***, el cual contiene el archivo *actions.class.php* siendo este la clase controladora, donde se realizan todas las funcionalidades de la aplicación correspondiente.

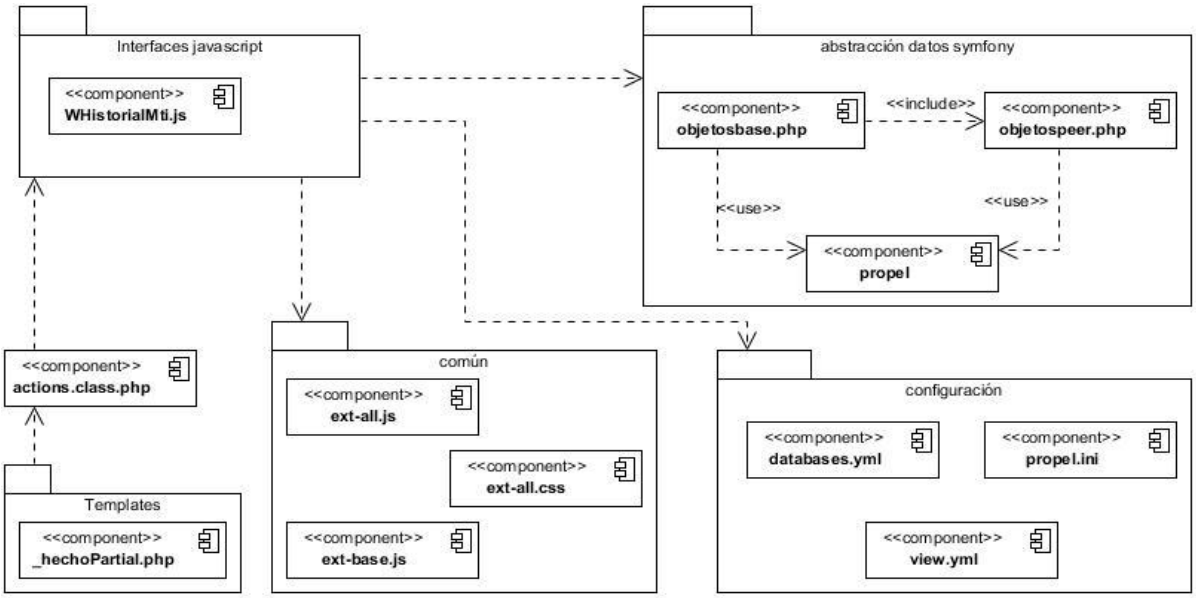

Figura 12. Diagrama de componentes.

2.6.3 Estándares de codificación.

Un estándar de codificación no es más que una guía a seguir por los desarrolladores de un proyecto, con el propósito de centrar un estilo propio del equipo a la hora de generar el código fuente de una aplicación.

El departamento de Soluciones para la Aduana define un estándar de codificación que sirve de apoyo al personal del proyecto Aduana para identificar de forma sencilla cuál es el objetivo y las funcionalidades que brinda cada una de las clases, funciones y demás componentes de software dada su nomenclatura. Además debe servir de guía para posteriores implementaciones o modificaciones del sistema. Comprende todo el código generado bajo la tecnología y el lenguaje PHP y que utilicen la arquitectura regida por la utilización del *framework* arquitectónico *Symfony*.

Regla general

Cuando se incluyen abreviaturas en mayúsculas no se debe incluir su nombre completo, sino que se utiliza el primer nombre en mayúscula y el resto en minúsculas.

Correcto: *getHtmlStatistic*
Incorrecto: *GetHtmlStatistic*

Aplicaciones

- ✓ Las aplicaciones deben tener nombres que dejen reflejado bien claramente cuál es el propósito de la misma, ya en una palabra o siglas.
    - En caso de ser mediante siglas se pondrán todas en mayúsculas.
- ✓ Se debe evitar mientras sea posible la utilización de palabras compuestas o la utilización de varias palabras, en caso de que sea palabras compuestas se utilizará la notación ***UpperCamelCase***[30].

Módulos

- ✓ Deben referirse a los nombres de tablas en caso de que se trate de un módulo generado por el CRUD[31].
- ✓ En caso de que sean módulos del negocio de la aplicación debe cumplir con las mismas reglas de codificación de los nombres de las aplicaciones.

Acciones

El framework Symfony trae su propia nomenclatura para las clases de las acciones y sus funciones, pero no especifica claramente cuál es el nombre que se le debe poner a cada una de las funcionalidades o acciones del modelo que serán accedidas por el usuario.

- ✓ Dentro de las especificaciones del *framework,* cada una de las acciones debe comenzar con la palabra ***execute***.
- ✓ Todos los nombres de acciones deben estar en la nomenclatura "***CamelCase*"** comenzando por la palabra ***execute***.

[30] Es un estilo de escritura que se aplica a frases o palabras compuestas.
[31] Acrónimo de Crear, Obtener, Actualizar y Borrar (del original en inglés: ***C****reate,* ***R****ead,* ***U****pdate and* ***D****elete*).

- ✓ En caso de ser acciones referentes a un módulo de CRUD de una tabla deben ser nombres específicos como ***executeNuevo***, ***executeEditar***, etc.
- ✓ Los nombres de las acciones deben especificar con la menor cantidad de palabras el objetivo de la acción, de ser posible estar en infinitivo. Debe especificar bien claro lo que se pretende ejecutar con la acción pero sin especificar los parámetros que recibe.

Ejemplo:

Correcto: usuario/editar
Incorrecto: usuario/editar_dado_id

Plugins

El *framework Symfony* especifica que la notación de los *plugins* siempre debe estar atada al sufijo *Plugin*. Para nombrar el plugin se debe iniciar con el prefijo ***sf*** y seguido del nombre que debe contener la menor cantidad posible de palabras y se le coloca el sufijo ***plugin***.

Ejemplo:

Correcto: *sfMtiPlugin*
Incorrecto: *Mti*

Nombre de las clases

- ✓ Los nombres de las clases deben estar expresados en notación *UpperCamelCase*.
- ✓ No se deben utilizar guiones bajos en su nombre "_".
- ✓ Deben expresar con claridad cuál es el alcance y la responsabilidad de la clase.
- ✓ Los nombres de las clases no deben estar atados a las clases de las que se deriva, cada clase debe tener un significado por ella misma, no en dependencia de la clase de la que deriva.
- ✓ En los nombres compuestos por más de tres palabras se debe revisar el diseño, no sea que se le estén dando a la clase más responsabilidades de las que realmente tiene.

Nombres de los archivos de las clases

Los nombres de los archivos de las clases deben estar compuestos por el nombre de la clase seguido de un punto y la palabra "*class*" y la extensión del archivo ".*php*".

Ejemplo:

MiClase.class.php

Variables

- ✓ Los nombres de las variables deben expresar claramente el contenido de la misma.
- ✓ Pueden estar referidas en singular o plural.
- ✓ Se definen al principio de las estructuras donde son utilizadas.
- ✓ En caso de que no se le asigne un valor inicial se deben inicializar con un valor que indique el tipo de dato más general al que debe pertenecer.
- ✓ Los tipos de datos cadena son definidos con comillas dobles (“).
- ✓ Los tipos de datos de caracteres se definen con comillas simples (‘).
- ✓ En caso de que se espere almacenar tipos de datos diversos no se inicializa.

2.6.4 Tratamiento de Errores.

En el desarrollo de una aplicación el tratamiento de errores es fundamental ya que se logra el correcto funcionamiento del sistema y permite identificar los errores que pueden aparecer cuando el usuario interactúa con el sistema.

Para garantizar un correcto tratamiento de errores en el sistema se tratan todos los errores que puedan presentarse durante la comunicación con la base de datos y para lograrlo se realiza la validación que compruebe la corrección de los datos. Uno de los objetivos principales es que el usuario introduzca la menor cantidad de valores posibles para evitar incoherencias de los mismos. Para estos datos se implementan funciones que validen y en caso de errores mostrar mensajes con la información de todos los errores cometidos durante la gestión (inserción, eliminación, modificación y obtención) de datos.

2.6.5 Comunicación entre las capas.

El *framework Symfony* está basado en el patrón MVC, el cual divide una aplicación en tres capas (ver **epígrafe 1.4.1**). Mediante este modelo, el *framework Symfony* interactúa fácilmente con estas capas, facilitando mayor organización e independencia entre las mismas (ver **figura 3**).

Entre el cliente y el controlador frontal y viceversa se utiliza tecnología JSON, por medio de esta los datos pasados por el usuario son serializados[32] en objetos PHP, esto permite interactuar con el servidor de la aplicación con cualquier tipo de interfaz y además da seguridad en la comunicación pues los implementadores de interfaz de usuario no van a tener acceso al comportamiento de los objetos que son enviados desde la capa controladora. También se combina esta tecnología con AJAX para aumentar la velocidad y el dinamismo de la interacción con la aplicación siempre que sea posible. Se utiliza AJAX con comunicación basada en XML para cuando no es posible enviar los datos por JSON. (20)

La comunicación entre las capas del Modelo y el Controlador se realiza por el envío de objetos con la información necesaria para el manejo de las peticiones del usuario y de las acciones a realizar, por medio de la tecnología mapeo de objetos a bases de datos (ORM[33]). En la comunicación con la Base de Datos se utiliza *PHP Data Objects (*PDO[34]), librería que trae el *Propel*[35] para la abstracción de la Base de Datos. (20)

Creando el plugin.

Para la obtención del componente desarrollado fue necesaria la creación de un *plugin* que permite establecer la comunicación entre los diferentes subsistemas del sistema GINA y estos puedan acceder a los servicios brindados por dicho componente.

Seguidamente se muestran algunos ejemplos de la aplicación donde se evidencia la comunicación entre las capas partiendo de los RF identificados en la fase de análisis.

Ejemplo RF: Consultar Historial MTI

La pantalla principal del componente Historial de MTI (ver **figura 16**) parte en el momento que el usuario seleccione un MTI de los presentados en la **figura 13** para consultar su registro histórico y a su vez dicho

[32] Proceso de convertir el estado de un objeto en un formato que se pueda almacenar o transportar por la red.
[33] Serie de objetos que permiten acceder a los datos y que contienen en su interior cierta lógica de negocio.
[34] Proporciona una interfaz entre el código PHP y el código SQL de la base de datos, permitiendo cambiar fácilmente de sistema gestor de bases de datos.
[35] Proporciona persistencia para los objetos y un servicio de consultas.

usuario esté autenticado en uno de los subsistemas del sistema GINA, para determinar los privilegios de acceso que tendría éste en la aplicación. En caso de que el usuario desee consultar el historial, debe presionar el botón **Ver Historial**, pero si antes de este paso el usuario no selecciona ninguno de los MTI mostrados en el listado, entonces saldría un mensaje de error (ver **figura 15**) informando al usuario que debe seleccionar un MTI.

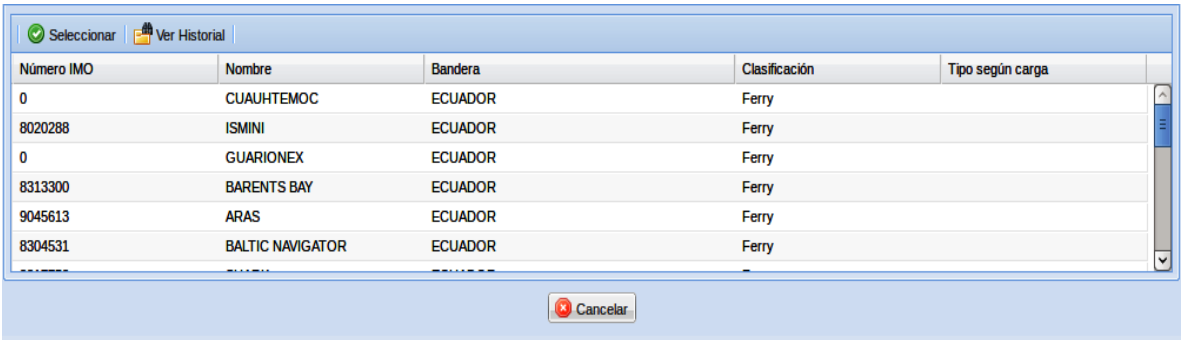

Figura 13. Listado de MTI a realizar el historial.

La **figura 13** muestra un listado tomado como ejemplo de la pantalla buscar MTI, en el módulo *controlMti* del subsistema LCF, en este caso dicho módulo obtendría mayor facilidad para realizar la búsqueda puesto que el componente Historial de MTI le facilitaría mayor accesibilidad a la información relacionada al MTI cuando se desee realizar el control deseado al MTI.

La **figura 14** contiene un segmento de código perteneciente a la clase *CBuscarMti.js* del módulo *controlMti* del subsistema LCF. En él se muestra un ejemplo de una instancia del *plugin* realizado con el presente trabajo (*WHistorialMti)*, para que dicho subsistema haga uso del componente.

```
text: 'Ver Historial',
iconCls: 'iconFolderFind',
scope: this,
handler: function(){
var g = Ext.getCmp( this.idGridBusqueda );
var record = g.getSelectionModel().getSelected();
if (record != null) {
var historial = new HistorialMti();
historial.show();
Ext.Msg.show({
```

```
title:'Informaci&oacute;n',
msg: "Mostrar el Historial del MTI: "+record.data.noMatricula,
width: 400,
buttons: Ext.Msg.INFO,
icon: Ext.MessageBox.INFO
});
} else {
Ext.Msg.show({
title:'Informaci&oacute;n',
msg: "Por favor seleccione un MTI.",
width: 300,
buttons: Ext.Msg.INFO,
icon: Ext.MessageBox.INFO
});
}}
```

Figura 14. Segmento de código donde se instancia el historial de MTI.

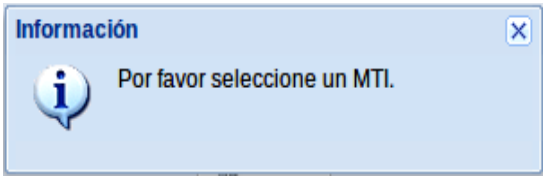

Figura 15. Mensaje de error para el usuario.

La pantalla principal de la aplicación está compuesta por dos secciones. La segunda consta de un panel donde se informa al usuario los datos generales del MTI seleccionado, ya sean la foto y los datos característicos de la embarcación de recreo, del buque o la aeronave; estos datos se mostrarán en dependencia del tipo de MTI que se seleccione con anterioridad.

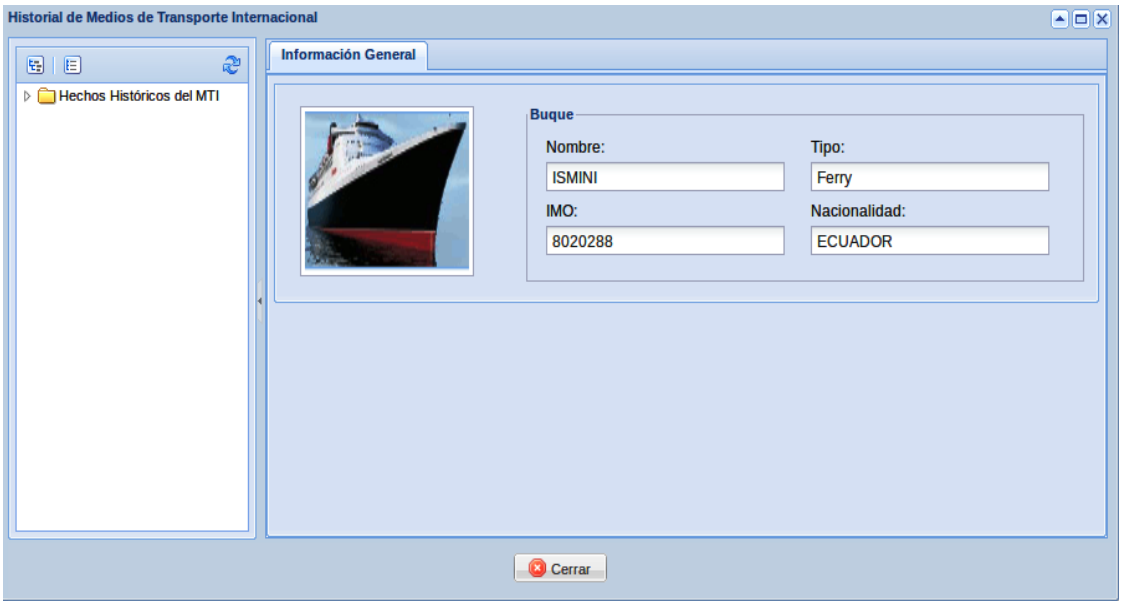

Figura 16. Pantalla Principal del componente Historial de MTI.

La primera sección es el árbol donde se mostrará un registro histórico de los hechos ocurridos asociados al MTI seleccionado por el usuario, este registro histórico será cargado de forma dinámica desde la base de datos, primeramente estará organizado por los subsistemas donde se registraron los hechos, luego por años de ocurrencia, después el tipo de hecho que ocurrió, la fecha exacta y por último el hecho a consultar, el cual tiene asociada una descripción que aportaría más detalles al interesado y se mostraría en una pestaña (ver **figura 17**) de la segunda sección de la pantalla.

Figura 17. Detalles del hecho ocurrido relacionado al MTI.

A continuación se muestra en la **figura 18** el fragmento de código perteneciente a la pantalla principal, el cual evidencia la aplicación del **Modelo-Vista-Controlador**, en la petición *ObtenerDatosGeneralesMti* correspondiente al RF: Consultar hechos de un MTI.

La **Vista** se encarga de obtener los datos mediante la propiedad "*url*" y enviar la información deseada mediante los parámetros de envío, desde las interfaces hacia el controlador, a través de peticiones mediante la tecnología *Ajax* o JSON.

```
Ext.Ajax.request({
        url: 'sfHistorialMti/ObtenerDatosGeneralesMti',
        params: {
        idMti: this.idMti
        },
        success: function(a){
        mask.hide();
        var objeto = Ext.decode(a.responseText);
        Ext.getCmp(obj.idForm).getForm().setValues(objeto);
        Ext.Msg.hide();
        },
        failure: function(){
        mask.hide();
```

```
        win.destroy();
});
```

Figura 18. Petición *Ajax* desde la interfaz WHistorialMti.js (Vista).

Luego el ***Controlador*** se encarga de obtener los datos enviados desde la *Vista* (ver **figura 19**) *y* según la petición, realizar la acción correspondiente, la cual accediendo al **Modelo** (ver **figura 20**) devolverá la respuesta determinada.

```
        $idMti = $this->getRequestParameter('idMti');
        $mti = MtiPeer::retrieveByPK($idMti);
        $tipo = $mti->getTipoMti();
        $component = SysComponentsLocator::getInstance();
        $tcComponent = $component->getComponent('tc');
        $params = array('nombreTc' => TcPaisPeer::TABLE_NAME, 'id' => $mti->getNacionalidad());
        $pais = $tcComponent->executeService(new sfEvent(null, 'tc.obtenerRegistroXid', $params));
        $result = array(
        'tipo_mti' => $tipo,
        'nacionalidad' => $pais->getDescripcion()
        );
        switch ($tipo) {
        case 'B':
        $buque = $mti->getMtiBuque();
        $result['imo'] = $buque->getImo();
        $result['nombre'] = $buque->getNombre();
        $result['tipo_buque'] = $buque->getTipoBuque();
        break;
```

Figura 19. Segmento de código de la acción *ObtenerDatosGeneralesMti* (Controlador).

```
        $nac = $this->getNacionalidad();
        if(is_numeric($nac)){
        $c = new Criteria();
        $c->add(TcPaisPeer::ID_PAIS,$nac);
        $access = new Access(Access::TC_PAIS,$c);
        $access->ejecutarConsulta();
        $resultado = $access->getResultado();
        if(!empty($resultado)){
        $pais = $resultado[0];
        $nac = $pais->getDescripcion();
        }
        }
```

Figura 20. Segmento de código de la clase donde se captura la nacionalidad del MTI (Modelo).

2.7 Conclusiones parciales.

Los patrones utilizados durante el diseño de la solución permitieron que se alcanzara en esta etapa un buen diseño funcional, adecuado para los requisitos que satisfacen la solución.

Con la obtención de cada uno de los artefactos del diseño definidos por el Departamento de Soluciones para la Aduana del CEIGE se logró fundamentar la implementación del sistema.

El diseño y la implementación de las interfaces y clases del negocio permitieron obtener una herramienta capaz de centralizar el historial de MTI y que exista comunicación entre las diferentes aplicaciones del sistema GINA.

Capítulo 3: Validación de la Solución.

3.1 Introducción.

Existe en la actualidad grandes cantidades de proyectos de software que no llegan a cumplir los objetivos propuestos, puesto que no se desarrollan en el tiempo estimado. El factor determinante de dicha situación es la omisión de pasos importantes en el ciclo de vida de desarrollo siendo la causa del fracaso de estos, influyendo considerablemente en el descontento del cliente y del equipo de trabajo. Teniendo en cuenta lo expresado anteriormente se procede a realizar en el presente capítulo la validación de la solución para probar el cumplimiento del objetivo inicial del sistema, así como la comprobación de las funcionalidades utilizadas que dan respuesta a los diferentes requisitos planteados.

3.2 Validación del diseño.

Luego del diseño realizado se procede a su validación mediante la aplicación de las métricas orientadas a las clases ya explicadas en el **epígrafe 1.5**, de las cuales se evidencia las métricas propuestas por *Lorenz* y *Kidd*.

Del conjunto de métricas planteadas por *Lorenz* y *Kidd* se aplicaron al diseño propuesto:

- ✓ **Tamaño operacional de clase (TOC)**
- ✓ **Relaciones entre clases (RC)**

3.2.1 Métrica tamaño operacional de clase.

La métrica **TOC** es muy usada por diseñadores de software, con una amplia documentación y literatura, fácil de calcular y bastante efectiva. Se basa en el número de métodos asignados a una clase y evalúa los siguientes atributos de calidad:

Atributo de Calidad	Modo en que lo afecta
Responsabilidad	El aumento del TOC implica el aumento de la responsabilidad asignada a la clase.
Complejidad de	El aumento del TOC implica el aumento de la

implementación	complejidad de implementación de la clase.
Reutilización	El aumento del TOC implica la disminución del grado de reutilización de la clase.

Tabla 2. Evaluación de los atributos de calidad, métrica tamaño operacional de clase.

Para un total de 11 clases que pertenecen al diseño, se obtiene tras aplicar la métrica, valores promedio de 0.54 operaciones por clase.

Total de Clases	Promedio de operaciones
11	0.54

Tabla 3. Cantidad de clases y promedio de operaciones.

Para poder realizar una evaluación de las métricas es necesario conocer los valores de los umbrales para evaluar los atributos de calidad. Los umbrales aplicados en el diseño fueron los propuestos por algunos especialistas para esta métrica que es basarse en el promedio de operaciones obtenido por clases. A continuación de describen cada uno de los umbrales.

	Categoría	Criterio	
Responsabilidad	Baja	<=Promedio	<= 0.54
	Media	>Promedio y <= 2*Promedio	>0.54 y <=1.08
	Alta	>2*Promedio	>1.08
Complejidad	Baja	<=Promedio	<= 0.54
	Media	>Promedio y <= 2*Promedio	>0.54 y <=1.08
	Alta	>2*Promedio	>1.08
Reutilización	Baja	>2*Promedio	>1.08
	Media	>Promedio y <= 2*Promedio	>0.54 y <=1.08

	Alta	<=Promedio	<= 0.08

Tabla 4. Valores de los umbrales para la métrica: tamaño operacional de clase.

Luego de representar las clases en correspondencia con el umbral especificado se obtuvo como resultado que 9 de ellas son de tamaño pequeño, 1 mediana y 1 grande, esto puede ser visualizado en la **figura 21**:

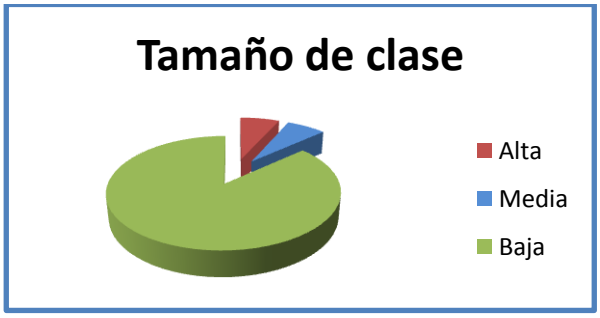

Figura 21. Representación del tamaño de clase.

A continuación se muestran los resultados obtenidos al aplicar los umbrales definidos para cada uno de los atributos de calidad.

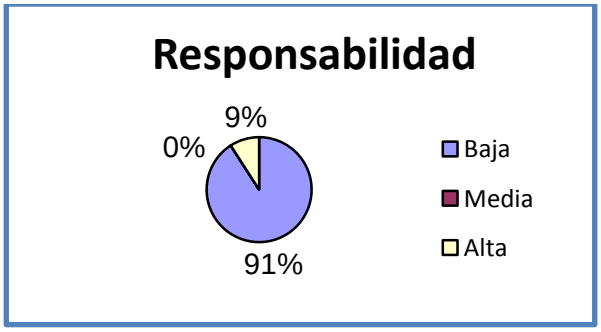

Figura 22. Representación de las clases según la responsabilidad.

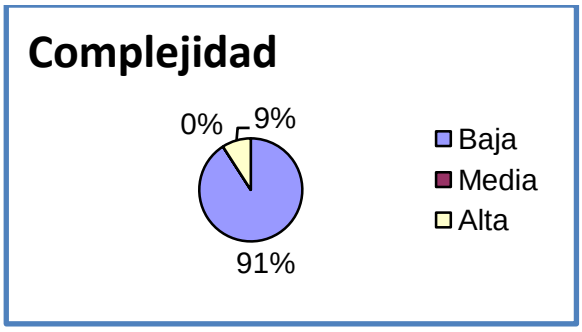

Figura 23. Representación de las clases según la complejidad.

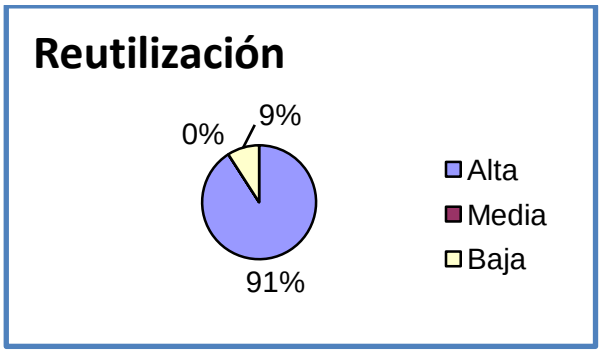

Figura 24. Representación de las clases según la reutilización.

Analizando los resultados obtenidos bajo la medición de la métrica TOC se puede observar que un 91 % de las clases presentan valores bajos de responsabilidad, lo que hace que carezcan de mucha complejidad a la hora de la implementación y por lo tanto se tornan más reutilizables. Sólo un 9 % de las clases muestran valores altos de responsabilidad y complejidad lo que incurre en un bajo grado de reutilización de las mismas.

A partir de los valores obtenidos se puede concluir que el diseño realizado es efectivo ya que va a permitir, en el mayor por ciento de sus clases, una baja responsabilidad evitando darle a la mayoría de las clases demasiada responsabilidad lo que aumenta la reutilización de las mismas, logrando evitar a la vez un alto por ciento de complejidad en la implementación, factor que puede impedir el correcto funcionamiento de un diseño.

3.2.2 Métrica Relaciones entre clases.

La métrica RC trata de encontrar todas las relaciones que existan entre la clase A con una o varias clases B. Estas relaciones se definen por los atributos o métodos que utilice la clase A de la clase B al menos una vez para darle respuesta a alguna petición.

En el proceso de aplicación de esta métrica fue necesario definir de todas las clases persistentes del diseño, las que se apoyan en las operaciones o datos que no se encuentran dentro de sus responsabilidades.

Por medio de esta métrica se evalúan los atributos de calidad que se describen a continuación:

Atributo de Calidad	Modo en que lo afecta
Acoplamiento	El aumento del RC implica el aumento del acoplamiento de la clase.
Complejidad de mantenimiento	El aumento del RC implica el aumento de la complejidad del mantenimiento de la clase.
Reutilización	El aumento del RC implica la disminución en el grado de reutilización de la clase.

Tabla 5. Evaluación de los atributos de calidad, métrica relaciones entre clases.

Para un total de 11 clases que pertenecen al diseño, se obtuvieron tras aplicar la métrica, valores promedio de 1.90 asociaciones de uso por clase. (Ver **tabla 6**)

Total de Clases	Promedio asociaciones de uso
11	1.9

Tabla 6. Cantidad de clases y promedio asociaciones de uso.

Los valores de los umbrales para aplicar esta métrica se basan en el promedio de asociaciones de uso por clases y se describen a continuación.

	Categoría	Criterio	
Acoplamiento	Ninguno	0	
	Bajo	1	
	Medio	2	
	Alto	>2	
Complejidad de mantenimiento	Bajo	<=Promedio	<= 1.9
	Medio	> Promedio y <= 2*Promedio	>1.9 y <=3.8
	Alto	> 2*Promedio	>3.8
Reutilización	Bajo	>2*Promedio	>3.8
	Medio	> Promedio <= 2*Promedio	>1.9 y <= 3.8
	Alto	<=Promedio	<=1.9

Tabla 7 Valores de los umbrales para la métrica relaciones entre clases.

Luego de ser evaluadas cada una de las clases en correspondencia con los valores de los umbrales definidos para cada uno de los atributos de calidad se obtienen los siguientes resultados:

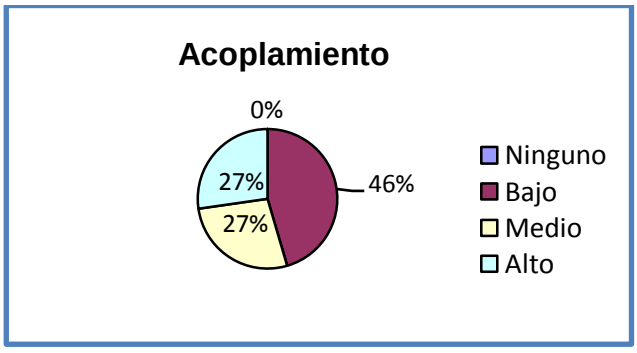

Figura 25. Representación de las clases según el acoplamiento.

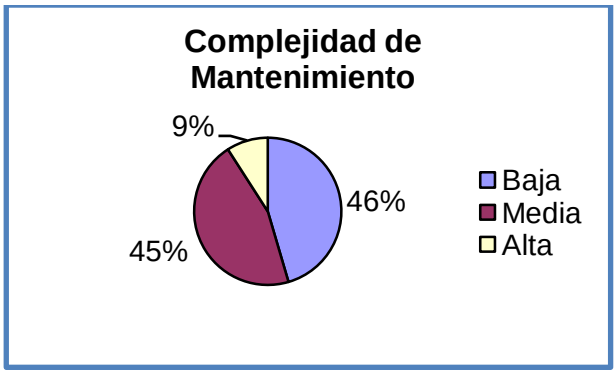

Figura 26. Representación de las clases según complejidad de mantenimiento.

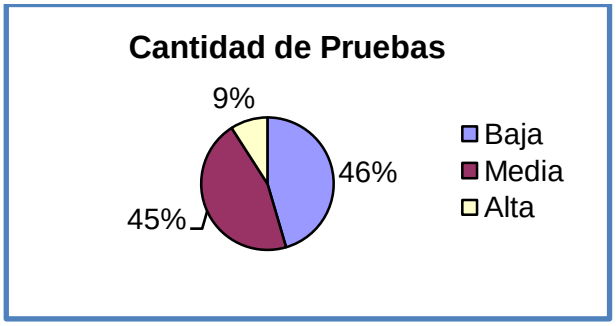

Figura 27. Representación de las clases según cantidad de pruebas.

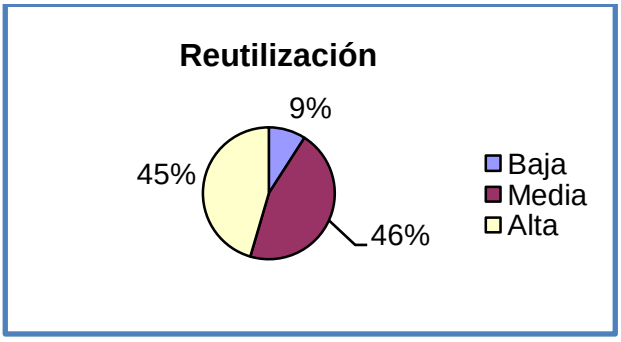

Figura 28. Representación de las clases según la reutilización.

La evaluación de los resultados obtenidos durante la aplicación de la métrica RC, demuestra que el diseño propuesto no presenta problemas para poder realizar un adecuado funcionamiento. Al lograrse un valor de

acoplamiento de un 73% que oscila entre bajo y medio, lo que representa un valor aceptable desde el punto de vista de implementación del software, además la complejidad de mantenimiento es baja a un 46%, dado estos valores es posible la reutilización de las operaciones con un valor entre medio y alto de 91%, lo que representa un aspecto clave para mejorar la obtención de respuestas.

3.3 Pruebas de Software.

En el desarrollo del software, las posibilidades de error son innumerables. Pueden darse por una mala especificación de los RF, uso indebido de las estructuras de datos, errores en las clases de IU, etc. En el desarrollo del software debería de existir alguna actividad que garantice la calidad.

La prueba es un conjunto de actividades que se plantean con anticipación y se realizan de forma sistemática. La prueba de software es un elemento de un tema más amplio que suele denominarse verificación y validación. Verificación es el conjunto de actividades que aseguran que el software implemente correctamente una función específica. Validación es un conjunto diferente de actividades que aseguran que el software construido corresponde con los requisitos del cliente. (8)

3.3.1 Estrategias de pruebas.

La estrategia que se ha de seguir a la hora de evaluar dinámicamente un sistema software debe permitir comenzar por los componentes más simples y más pequeños e ir avanzando progresivamente hasta probar todo el software en su conjunto. Más concretamente, los pasos a seguir son:

- ✓ Pruebas Unitarias. Comienzan con la prueba de cada módulo.
- ✓ Pruebas de Integración. A partir del esquema del diseño, los módulos probados se vuelven a probar combinados para probar sus interfaces.
- ✓ Prueba del Sistema. El software ensamblado totalmente con cualquier componente hardware que requiere se prueba para comprobar que se cumplen los requisitos funcionales.
- ✓ Pruebas de Aceptación. El cliente comprueba que el software funciona según sus expectativas. (8)

Para validar la solución se empleó la estrategia de pruebas unitarias, para ello se describe en que consisten las mismas.

Pruebas Unitarias: aseguran que un único componente de la aplicación produce una salida correcta para una determinada entrada. Este tipo de pruebas validan la forma en la que las funciones y métodos trabajan en cada caso particular. Las pruebas unitarias se encargan de un único caso cada vez, lo que significa que un único método puede necesitar varias pruebas unitarias si su funcionamiento varía en función del contexto. (20)

La prueba de unidad es la primera fase de las estrategias de pruebas y se realizan sobre cada módulo del software de manera independiente. El objetivo es comprobar que el módulo, entendido como una unidad funcional de un programa independiente, está correctamente codificado. En estas pruebas cada módulo será probado por separado y lo hará, generalmente la persona que lo creó. Este tipo de pruebas son altamente recomendadas llevarlas a cabo antes de realizar las pruebas funcionales, debido a que las pruebas unitarias permiten encontrar los errores más evidentes y fáciles de corregir, mientras en la etapa de pruebas funcionales el sistema debería estar bastante estable y con muy pocos errores críticos. (8)

3.3.1.1 Pruebas Unitarias aplicadas a la aplicación.

Las pruebas unitarias se desarrollaron haciendo uso del *framework Symfony*, utilizado en la implementación del componente. Estas pruebas son archivos PHP normales cuyo nombre termina en *Test.php* y que se encuentran en el directorio *test/unit/* de la aplicación. Su sintaxis es sencilla y fácil de leer. Cada prueba unitaria consiste en una llamada a un método de la instancia de *lime_test33*[36]. El último parámetro de estos métodos siempre es una cadena de texto opcional que se utiliza como resultado del método. En el presente epígrafe se muestran las pruebas unitarias del *framework Symfony* aplicadas a las funcionalidades más significativas definidas en la implementación de la aplicación.

Ejemplo 1:

En el caso que se muestra a continuación se explica la prueba unitaria aplicada a la funcionalidad ***DatosGeneralesMti*** (**tabla** 8) partiendo de un listado de MTI. Dicha funcionalidad cuenta con un

[36] Objeto del framework de pruebas lime que utiliza Symfony para realizar sus pruebas unitarias.

escenario de prueba y permite devolver los datos generales correspondiente al MTI seleccionado (*IdMti*) y que se encuentra almacenada en base de datos.

Prueba	*DatosGeneralesMti.*
Entrada	Identificador del MTI a seleccionar partiendo del listado de MTI.
Salida	Un arreglo con los datos generales del MTI.
Condición	Si la cadena devuelta coincide con lo esperado se considera satisfactoria.

Tabla 8. Parámetros aplicados a las pruebas unitarias realizadas.

Teniendo en cuenta lo descrito en la **tabla** 8 se muestra a continuación los resultados obtenidos luego de aplicar la prueba unitaria a dicha funcionalidad. Se tomó el MTI con identificador 414 (*$idMti = 414*), la comparación de los resultados se muestran en la **tabla** 9.

No.	Descripción	Resultado esperado	Resultado obtenido
1	Tipo de resultado	array	array
2	Nombre del MTI	ISMINI	ISMINI
3	Tipo	Ferry	Ferry
4	IMO	8020288	8020288
5	Nacionalidad	ECUADOR	ECUADOR
6	URL de la foto	js/images/buque.gif	js/images/buque.gif

Tabla 9. Resumen de resultados al aplicar las pruebas unitarias del *framework Symfony* al escenario de pruebas 1.1.

Las pruebas unitarias del *framework Symfony* para la funcionalidad *DatosGeneralesMti* se definieron de la siguiente forma:

```
$idMti = 414;
$result = DatosGeneralesMti($idMti);
$t->isa_ok($result, 'array');
$t->is($result['nombre'], ISMINI);
$t->is($result[Tipo], Ferry);
$t->is($result[imo], '8020288');
```

```
$t->is($result['nacionalidad'], 'ECUADOR');
$t->is($result['image'], 'js/images/buque.gif');
```

El ejecutar estas pruebas el resultado arrojado es el siguiente:

```
www\GINA>symfony test: unit HistorialMti

1..6
ok 1 - se obtuvo un objeto.
ok 2 - el nombre del MTI obtenido es el esperado.
ok 3 - el tipo de MTI obtenido es el esperado.
ok 4 - el imo del MTI obtenido es el esperado.
ok 5 - la nacionalidad del MTI obtenida es la esperada.
ok 6 - la foto del MTI obtenida es la esperada.
Looks like everything went fine.
```

El resultado anterior demuestra que las seis pruebas realizadas se ejecutaron correctamente.

Ejemplo 2:

En el caso que se muestra a continuación se explica la prueba unitaria aplicada a la funcionalidad ***MostrarHechoHistorial*** (**tabla** 10) partiendo de un listado de hechos. Dicha funcionalidad cuenta con un escenario de prueba y permite devolver una descripción avanzada correspondiente al hecho seleccionado (*IdHecho*) asociado a un MTI y que se encuentra almacenada en base de datos.

Prueba	*MostrarHechoHistorial.*
Entrada	Identificador del hecho a seleccionar partiendo del árbol que contiene el registro historial de determinado MTI.
Salida	Un arreglo con los datos a mostrar en la descripción avanzada del hecho.
Condición	Si la cadena devuelta coincide con lo esperado se considera satisfactoria.

Tabla 10. Parámetros aplicados a las pruebas unitarias realizadas.

Teniendo en cuenta lo descrito en la **tabla** 10 se muestra a continuación los resultados obtenidos luego de aplicar la prueba unitaria a dicha funcionalidad. Se tomó el hecho con identificador 1 (*$idHecho = 1*), la comparación de los resultados se muestran en la **tabla** 9.

No.	Descripción	Resultado esperado	Resultado obtenido
1	Tipo de resultado	array	array
2	Id Hecho	1	1
3	anno	2012	2012
4	fecha	01-06-2012	01-06-2012
5	hora	04:06:40	04:06:40
6	descripcion	Descripcion del hecho ocurrido asociado al mti	Descripcion del hecho ocurrido asociado al mti.
7	codigoSubsistema	11	11
8	codigoTipoHecho	1	1

Tabla 11. Resumen de resultados al aplicar las pruebas unitarias del *framework Symfony* al escenario de pruebas 1.1.

Las pruebas unitarias del *framework Symfony* para la funcionalidad *MostrarHechoHistorial* se definieron de la siguiente forma:

```
$idHecho = 1;
$result = MostrarHechoHistorial($idHecho);
$t->isa_ok($result, 'array');
$t->is($result[idHecho], 1);
$t->is($result[anno], 2012);
$t->is($result[fecha], 01-06-2012);
$t->is($result[hora], ’04:06:40);
$t->is($result[descripcion], ' Descripcion del hecho ocurrido por el mti ');
$t->is($result[codigoSubsistema], 11);
$t->is($result[codigoTipoHecho], 1);
```

El ejecutar estas pruebas el resultado arrojado es el siguiente:

```
www\GINA>symfony test: unit MostrarHechoHistorial
```

```
1..8
ok 1 - se obtuvo un objeto.
ok 2 - se obtuvo el identificador del hecho esperado.
ok 3 - se obtuvo el anno esperado.
ok 4 - se obtuvo la fecha exacta esperada.
ok 5 - se obtuvo la hora exacta esperada.
ok 6 - se obtuvo la descripción detallada esperada.
ok 7 - se obtuvo el susbsistema esperado.
ok 8 - se obtuvo el tipo de hecho esperado.
```

Luego de realizadas las pruebas unitarias a las dos funcionalidades mostradas se observa que fueron satisfactorias. Cada línea obtenida valida un único resultado.

3.3.2 Resumen de la validación.

En el proceso de validación de la solución obtenida en la presente investigación se realizaron pruebas unitarias a la aplicación, teniendo en cuenta que para cada funcionalidad se identificaron varios escenarios de pruebas y que cada escenario respondía a un requisito funcional en específico. Durante el **epígrafe 3.3** se describió detalladamente el proceso de pruebas realizado a la aplicación. Se aplicaron pruebas unitarias a las funcionalidades mas importantes como se describió anteriormente. Los resultados por escenarios de pruebas para cada una de las funcionalidades se detallan a continuación:

Funcionalidades en Consultar historial de MTI:

Registrar Hecho: 6 escenarios.

Mostrar árbol de hechos: 6 escenarios. 2 no conformidades.

Mostrar datos generales del MTI: 1 escenario.

Mostrar descripción avanzada del historial: 1 escenario.

Tras realizar las pruebas se obtuvo un total de 14 escenarios de prueba, detectándose un total de 2 no conformidades en la primera iteración (ver **figura 29**), esto permitió que se corrigieran de forma inmediata. Se hizo una segunda iteración logrando que el sistema estuviera un 100 % libre de errores en cuanto a unidad.

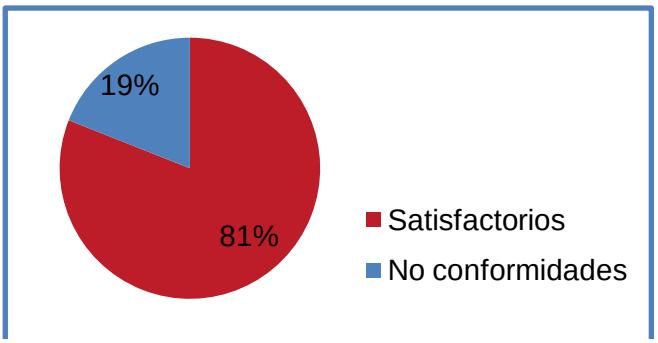

Figura 29. Resultados de las pruebas.

3.4 Conclusiones parciales.

La utilización de métricas para validar el diseño en cuanto a tamaño operacional de clases y relaciones entre clases permitió obtener buenos valores para los atributos de calidad predominando una baja responsabilidad, baja complejidad, bajo acoplamiento y un alto nivel de reutilización entre las clases.

La aplicación de Pruebas Unitarias a la solución desarrollada permitió encontrar errores que afectaban el funcionamiento del sistema, posibilitando que estos sean corregidos de inmediato permitiendo que finalmente se pueda utilizar el componente quedando libre de errores que afecten el sistema y cumpliendo con los objetivos trazados inicialmente.

Conclusiones.

Para la realización de la presente investigación primeramente se analizaron las soluciones existentes que gestionan los procesos aduanales, comprobándose que las analizadas no eran factibles por ser privativas, costosas y no adecuarse a las legislaciones de la AGR.

La utilización de patrones de diseño y la generación de los artefactos definidos por el Departamento de Soluciones para la Aduana, permitió generar el modelo de la implementación del sistema deseado.

El diseño y la implementación de las interfaces y las clases del negocio de la solución, permitieron obtener una aplicación que respondiera a los RF definidos durante la etapa de análisis.

La aplicación de Pruebas Unitarias facilitó la detección de no conformidades en el sistema obtenido permitiendo su solución inmediatamente.

Recomendaciones.

Aún con el cumplimiento de los objetivos, el diseño e implementación del sistema para la informatización de los procesos involucrados con el historial de MTI de la Aduana General de la República, se necesita incorporar recomendaciones que se consideran fundamentales:

- ✓ Incorporar el componente obtenido a la gestión del historial de MTI en los futuros subsistemas del sistema GINA que lo requieran.
- ✓ Continuar con el seguimiento de las actualizaciones de las herramientas y tecnologías informáticas empleadas en la solución para garantizar mejoras en futuras versiones del sistema.

Referencias bibliográficas.

1. **Naranjo Garcia, Adrian.** *Sistema de Autenticación y Control de Acceso para aplicaciones del Departamento de Soluciones para la Aduana.* Ciudad de la Habana: Universidad de las Ciencias infórmaticas. : s.n., 2010.

2. **Navasa Martínez, Amparo.** *Marco de trabajo para el desarrollo de arquitecturas software orientado a aspectos.* Cáceres : Universidad de Extremadura Servicio de Publicaciones : s.n., 2008.

3. **Fuentes, Lidia, Troya, José M. and Vallecillo, Antonio.** *Desarrollo de Software Basado en Componentes.* Málaga, España : ETSI Informatica : Campus Teatinos : s.n., 2012. S/N.

4. **Montilva C., Jonás A., Arapé, Nelson and Colmenares, Juan Andrés.** *Desarrollo de Software Basado en Componentes.* Venezuela : s.n., 2011. S/N.

5. **Ariza Rojas, Maribel and Molina García, Juan Carlos.** *INTRODUCCIÓN Y PRINCIPIOS BÁSICOS DEL DESARROLLO DE SOFTWARE BASADO EN COMPONENTES.* 2004.

6. Sistema SOFIA. [En línea] 2012. http://www.aduana.gov.py/49-4-sistema-sofia.html.

7. **Febres Arellano , Ana Alejandra.** Conociendo al SIDUNEA - Sistema Aduanero Automatizado. [En línea] 2012. http://www.gestiopolis.com/. UNCTAD-Venezuela Ana Febres RRPP/ UNCTAD-SIDUNEA.

8. **Pressman, Roger S.** *Ingenieria de software, Un enfoque Practico.* s.l. : Sexta Edicion.

9. **Cobo Rodriguez, Jose Antonio.** *Línea Base Arquitectónica para el Polo Sistemas Tributarios y de Aduanas.* Ciudad de la Habana: Universidad de las Ciencias infórmaticas. : s.n., 2008. S/N.

10. **Larman, Craig.** *UML y Patrones.* 1999. ISBN 970-1 7-0261-1..

11. **Duran Rivas, Yaksel.** *Diseño e implementación del módulo Importación de los Depósitos Temporales de Frontera.* Ciudad de la Habana: Universidad de las Ciencias infórmaticas. : s.n., 2011.

12. **CEIGE: Departamento Soluciones para la Aduana.** *MODELO DE DESARROLLO DE SOFTWARE.* Ciudad de la Habana: Universidad de las Ciencias infórmaticas. : s.n., 2012.

13. **Schmuller, Joseph.** *Aprendiendo UML en 24 horas.* PEARSON EDUCACION, Mexico : Pearson Educacion de Mexico, S.A. de C.V., : s.n., 2000. 968-444-463-X.

14. Visual Paradigm International. UML, BPMN and Database Tool for Software Development. [En línea] http://www.visual-paradigm.com.

15. PHP.net. PHP: Hypertext Preprocessor. PHP: General Information. [En línea] http://www.php.net/manual/en/faq.general.php.

16. **Pérez, Javier Eguíluz.** *Introducción a XHTML.* 2008.

17. —. *Introducción a CSS.*

18. —. *Introducción a JavaScript.* 2007.

19. EcuRed. [En línea] http://www.ecured.cu/index.php/Framework.

20. Potencier, Fabien and Zaninotto, François. Symfony la guía definitiva. . [En línea] 2012. www.librosweb.es..

21. http://extjs.com. [En línea] 2012. http://extjs.com.

22. **Velasco, Roberto Hernando.** El SGBDR Oracle. [En línea] http://www.rhernando.net/modules/tutorials/doc/bd/oracle.html..

23. Ecured. [En línea] http://www.ecured.cu/index.php/Patrones_en_Symfony.

24. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** Proceso Unificado de Desarrollo de Software. Wesley, Addison : 0-201-57169-2, 2000.

ANEXOS.

Anexo #1. Descripción del Modelo de Datos.

Nombre de la entidad:	**MTI_HECHO**	
Propósito de la entidad:	Almacenar los hechos de un MTI ocurridos en determinado momento.	
Nombre Campo	**Tipo Dato**	**Propósito**
ID_HECHO	NUMERIC	Llave primaria de la tabla
ID_TIPO_HECHO	NUMERIC	Relación con la tabla TC_TIPO_HECHO. Tabla de control Tipo de hechos.
ID_SUBSISTEMA	NUMERIC	Relación con la tabla TC_SUBSISTEMA. Tabla de control Subsistema.
ID_MTI	NUMERIC	Relación con la tabla MTI.
FECHA	DATE	Fecha de creación de un hecho.
ANNO	NUMERIC	Año de creación de un hecho.

Nombre de la entidad:	**TC_TIPO_HECHO**	
Propósito de la entidad:	Almacenar los tipos de hechos que corresponden a los distintos MTI.	
Nombre Campo	**Tipo Dato**	**Propósito**
ID_TIPO_HECHO	NUMERIC	Llave primaria de la tabla.

Nombre de la entidad:	**TCR_NIVEL_HISTORICO**
Propósito de la entidad:	Almacenar los campos de los cuales se relacionen un tipo de hecho con un rol de la aduana.

Nombre Campo	Tipo Dato	Propósito
ID_NIVEL_HISTORICO	NUMERIC	Llave primaria de la relación de las tablas
ID_TIPO_HECHO	NUMERIC	Relación con la tabla TC_TIPO_HECHO. Tipo de hechos.
ID_ROL_ADUANA	NUMERIC	Relación con la tabla TC_ROL_ADUANA. Rol de la aduana (tabla de Administración).

Anexo #2. Diagrama de secuencia.