UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 3

Título: Módulo para manejar el historial de las entidades y los trabajadores externos que pertenecen al Registro Central de la AGR.

Trabajo de Diploma para optar por el título de Ingeniero Informático

Autor (es): Nolvis Pérez Franco

Tutor: Ing. Raymond Weeden Gamboa

Tutor: Ing. Yasel Antonio Romero Piñeiro

La Habana, Junio de 2013

"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 Junio del año 2013.

Nolvis Pérez Franco

_____

Firma del Autor

Ing. Raymond Weeden Gamboa

_____

Firma del Tutor

Ing. Yasel Antonio Romero Piñeiro

_____

Firma del Tutor

"El mayor estímulo para esforzarnos en el estudio y en la vida es el placer del mismo trabajo, el placer de los resultados y la conciencia del valor de esos resultados para la comunidad."

Albert Einstein

Agradecimientos

A mis padres por confiar en mí en todo momento y sobre todo por su ejemplo y amor, sin ellos no hubiera sido posible seguir adelante siempre.

A mis tías Alicia, Marbellis y Oneida por su preocupación constante, por apoyarme y quererme como una hija más.

A mi hermano por estar siempre cuando lo necesito y por su confianza.

A mis abuelos, tíos y primos, a toda mi familia que siempre me ha apoyado los quiero.

A mi tutor Yasel por ser una persona especial, por comprenderme y enseñarme, por su paciencia y por estar siempre dispuesto a ayudarme.

A mi tutor Weeden por sus consejos y por toda la ayuda brindada. A ambos tutores muchas gracias, sin su ayuda no hubiera podido lograrlo.

A todo el equipo de desarrollo y amigos de Aduana nunca los olvidaré por ayudarme: Maurice, Adrián, Yordanis, Luis Manuel, Cusco, Idel, lázaro y Liliana.

A mis amigos por soportarme, entenderme, quererme y ayudarme siempre que lo necesité: Liz, Yamila, Yanet, Joceline, Yanerkis, Polanco, José Joel, Yosvany, Marcial y a dos de las personas más importantes para mi Helen y Jorge.

A todos mis compañeros de aula, a mi familia del secretariado de la FEU y de los sensuales.

A la Revolución Cubana y a Fidel, por darme la oportunidad de realizar mi sueño de graduarme.

A todos Muchas Gracias!!!

Dedicatoria

Dedico este trabajo a todas las personas que quiero, en especial mis padres, hermano, abuelos, tios, primos y demás familiares por ser ustedes una fuente de inspiración para mí.

A todos los amigos y personas que he conocido en el transcurso de estos cinco años de mi vida por su apoyo cuando lo he necesitado.

Nolvis Pérez Franco.

Resumen

Mantener un control sobre las operaciones que llevan a cabo cada una de las entidades y trabajadores de la Aduana General de la República de Cuba (AGR), es una tarea difícil para los funcionarios de la misma. La presente investigación tiene como objetivo fundamental concebir una solución informática que garantice la gestión de las operaciones realizadas por las entidades y trabajadores externos para facilitar la toma de decisiones en la AGR.

Para el desarrollo de la solución informática propuesta anteriormente se realizó un estudio previo de los sistemas que manejan historial de entidades y trabajadores, analizando sus principales características. Posteriormente se llevó a cabo un estudio de las metodologías y herramientas que fueron usadas para concebir la solución. Haciendo uso del modelo de desarrollo definido por el Centro de Informatización de la Gestión de Entidades (CEIGE) en las etapas de análisis, diseño e implementación, así como de validación, se obtuvo un módulo perteneciente al sistema de Gestión Integral de Aduanas (GINA)[1], que es capaz de mantener el registro de las acciones realizadas por los trabajadores y de centralizar el historial de las entidades, facilitando de esta forma su consulta en la AGR y garantizando la toma de decisiones oportuna en la misma.

Palabras claves: entidad, GINA, historial, operaciones, trabajador.

[1] **Gestión Integral de Aduanas. GINA**, sistema desarrollado por el Centro de Soluciones para la Aduana, del Centro para la Informatización y la Gestión de Entidades de la Universidad de las Ciencias Informáticas, como colaboración de la UCI y la Aduana General de la República.

Tabla de Contenidos

Tabla de Contenidos

Tabla de Contenidos

Introducción

La Aduana General de la República de Cuba (AGR) fue fundada el 5 de febrero de 1963. Es el órgano oficial que vela y regula el control aduanero aplicable a la entrada, el tránsito, el cabotaje, el trasbordo, el depósito y la salida del territorio nacional de mercancías, viajeros y sus equipajes, bienes y valores sujetos a regulaciones especiales, así como los medios en los que se transporten. Tiene como tareas principales garantizar la seguridad y protección de la sociedad socialista y de la economía nacional, así como la recaudación fiscal y la emisión de las estadísticas del comercio exterior, a través del cumplimiento de las políticas estatales de competencia aduanera para el tráfico internacional de viajeros, mercancías y medios de transporte. Constituye además, parte de la Administración Central del Estado y se subordina al Consejo de Ministros. (1)

La AGR tiene en funcionamiento diferentes sistemas informáticos que facilitan el control y la gestión de los procesos aduanales. Con el objetivo de integrar estas aplicaciones y generalizar los procesos de implementación y estandarización de las soluciones de desarrollo surge el sistema de Gestión Integral de Aduanas (GINA).

La Universidad de Ciencias Informáticas (UCI), cuenta con el Centro de Informatización para la Gestión de Entidades (CEIGE). En este centro se encuentra el departamento de Soluciones para la Aduana, en el cual se desarrolla el sistema GINA para automatizar los principales procesos que se llevan a cabo en la AGR. Este departamento desarrolla el Registro Central de Entidades (RCE) que en conjunto con los especialistas del Centro de Automatización para la Dirección y la Información (CADI), gestionan y controlan la participación de las entidades que son requeridas para la realización de los diferentes procesos aduaneros, como el comercio exterior e interior.

Actualmente el RCE emplea dos herramientas informáticas para su funcionamiento. Para generar los reportes hacen uso de DataEase 4.0, que implica incompatibilidades de software con las aplicaciones y herramientas modernas. Por otra parte se tiene el Sistema Único de Aduanas (SUA), que se encuentra en uso desde el año 2000; dicho sistema automatiza los procesos aduaneros: despacho comercial, no comercial, de viajeros, de medios de transporte y

de postal y envío. En el mismo son insertados los datos generados por el DataEase y este es capaz, dado que cuenta con tablas de control[2], de generar otros reportes necesarios en los procesos aduaneros; lo que trae consigo dependencia y redundancia en el proceso. La interfaz de usuario es poco intuitiva y poco amigable, dado que el DataEase se usa en interfaz de línea de comandos[3].

El sistema existente en la AGR no posee una forma de controlar las operaciones que realizan los trabajadores externos y entidades del RCE. Cuando la aduana decide dejar de trabajar con alguna entidad y luego de un tiempo necesita volver a trabajar con esta, los funcionarios tienen que introducir manualmente los datos de la entidad y de los trabajadores externos en el período que trabajó con la misma. Esta tarea aumenta los costos en tiempo y recursos humanos, además de que suele ser tediosa. El sistema SUA que se encuentra en uso en la AGR de Cuba, no permite obtener un reporte de las incidencias de una persona, ya que no se tiene un registro de las acciones que realizan los trabajadores de la aduana y los trabajadores externos a esta, durante un largo período de tiempo. Además de no tener registrado si estas incidencias son graves o no. Todo lo anterior afecta la toma de decisiones en la AGR al no tener una vía de consultar las operaciones realizadas por las entidades y trabajadores, así como del procesamiento de informes.

Una vez analizada la problemática anterior, se formula el siguiente **problema a resolver:** la falta de control sobre las operaciones realizadas por las entidades y trabajadores externos dificultan la toma de decisiones en la AGR.

Para dar solución al problema planteado anteriormente, se define como **objeto de estudio**: registro de historiales en sistemas informáticos de gestión aduanera.

Se determina como **campo de acción:** registro de historiales en los sistemas informáticos de la AGR.

---

[2] **Tablas de control:** dígase de los nomencladores que permiten el ingreso de datos seleccionando un elemento, minimizando con ello la probabilidad de que el usuario ingrese algún dato con errores.

[3] **Interfaz de Línea de Comandos.** CLI, Interfaz de modo texto para la ejecución de tareas mediante comandos en texto simple que el Sistema Operativo y las aplicaciones en uso son capaz de interpretar.

Introducción

Definiendo como **objetivo general:** desarrollar una solución informática que garantice la gestión de las operaciones realizadas por las entidades y trabajadores externos para facilitar la toma de decisiones en la AGR.

Se plantean como **objetivos específicos:**

- ✓ Elaborar el marco teórico relativo a las soluciones de software de gestión aduanera.
- ✓ Realizar el análisis y diseño de la solución.
- ✓ Implementar las funcionalidades identificadas en la solución teniendo en cuenta las herramientas definidas por el centro.
- ✓ Validar la solución obtenida haciendo uso de las técnicas de evaluación de software.

Para lograr el cumplimiento del objetivo planteado anteriormente, se definen como principales **tareas de la investigación:**

- ✓ Recopilación de la bibliografía referente al tema.
- ✓ Selección de la bibliografía.
- ✓ Análisis de la bibliografía.
- ✓ Valoración de las principales soluciones que manejen el problema en cuestión a nivel nacional e internacional.
- ✓ Especificación de los requisitos identificados.
- ✓ Definición de la solución teniendo en cuenta la arquitectura del sistema.
- ✓ Descripción formal de la solución de acuerdo a los procedimientos del centro.
- ✓ Definición de las pruebas de validación.
- ✓ Validación de la solución desarrollada.

Para el desarrollo de este trabajo de diploma se utilizarán algunos de los métodos científicos de la investigación, como los métodos teóricos que seguidamente se relacionan.

- ✓ **Analítico – sintético:** con el objetivo de realizar un estudio detallado de las teorías, tendencias y documentos relacionados con la necesidad de un componente que facilite el control de las acciones realizadas por las entidades y los trabajadores externos a lo largo del tiempo.
- ✓ **Histórico – lógico:** con el objetivo de hacer un estudio crítico de los trabajos anteriores relacionados con el tema a investigar, lo que todos conocen en el campo de la

investigación como el estudio del estado del arte. Dando la posibilidad de conocer acerca de la existencia y características de sistemas de este tipo que hayan sido desarrollados anteriormente en el mundo.

- ✓ **Modelación:** con el objetivo de desarrollar modelos estrechamente relacionados con el módulo, realizando abstracciones para dar una explicación de la realidad existente.

Conjuntamente a los métodos mencionados anteriormente y para completar la investigación se utilizará el siguiente método empírico:

- ✓ **Entrevista:** con el objetivo de obtener información sobre las necesidades reales que tiene la AGR, para que el sistema cumpla de forma completa con las mismas.

El presente trabajo de diploma contará con una estructura de tres capítulos, los cuales serán descritos a continuación:

Capítulo 1: en este capítulo se muestra la fundamentación teórica de la investigación. Se enuncian los principales conceptos que están asociados al dominio del problema. Se realiza el estudio crítico de las soluciones anteriores relacionadas con el problema planteado. Se relacionan las tendencias tecnológicas, herramientas, metodologías y lenguajes empleados en el desarrollo de la solución.

Capítulo 2: este capítulo abarca el análisis y el diseño del sistema, haciendo una descripción de los artefactos que son generados en el desarrollo de estas fases. Se describe los requerimientos del sistema, los diagramas de secuencia, el diseño de los prototipos de interfaz de usuario y de clases del negocio.

Capítulo 3: se lleva a cabo la implementación del sistema a partir de los artefactos concebidos en las etapas de análisis y diseño. Se realiza la validación de la solución propuesta mediante el uso de las distintas técnicas de evaluación de software, con el objetivo de determinar posibles no conformidades. Así como determinar si se cumplieron los objetivos propuestos inicialmente.

Capítulo 1: Fundamentación Teórica

1.1. Introducción

En el presente capítulo se describen los principales elementos que sirvieron de fundamentación teórica de la investigación. Se realiza un estudio profundo de las actuales soluciones informáticas a nivel nacional e internacional que están relacionadas con el registro de historiales de gestión aduanera. Se fundamenta acerca de aspectos y estilos arquitectónicos, tecnologías, herramientas, metodologías y lenguajes que son empleados en el desarrollo de la solución, para mejorar los procesos llevados a cabo por la AGR. Además de analizar y discutir los conceptos fundamentales relacionados con el tema a investigar, para proponer una definición propia de los mismos.

1.2. Historial de personas y entidades en registro central

Un historial es una reseña circunstanciada de los antecedentes de algo o de alguien, definición dada por la Real Academia Española. (2)

En la mayoría de los centros de trabajo se asocia el historial de las personas naturales con su currículum vitae[4]. En el RCE consiste en llevar un resumen de las operaciones (incidencias) de las personas, ya sean trabajadores externos o no a una entidad, así como las modificaciones que se le han realizado a una entidad.

Contar con el registro histórico de estos hechos sería uno de los aspectos más importantes dentro de la AGR puesto que serviría de referencia para gestionar y controlar a los trabajadores y entidades en las aduanas de Cuba.

1.2.1. Sistemas informáticos a nivel internacional

A continuación se muestran algunos de los principales Sistemas Integrales de Gestión de Aduanas que contienen herramientas para gestionar el historial de las entidades y trabajadores externos.

[4] **Currículum vitae**: relación de los títulos, honores, cargos, trabajos realizados, datos biográficos, etc., que califican a una persona.

SIDUNEA

El Sistema Aduanero Automatizado (SIDUNEA) es la herramienta informática para el control y administración de la gestión aduanera, este sistema es desarrollado por la Conferencia de las Naciones Unidas sobre el Comercio y el Desarrollo UNCTAD, y se encuentra protegido por derechos de propiedad intelectual. En la actualidad el sistema se encuentra operando o está en proceso de implementación en 93 países alrededor del mundo. (3) Es un sistema abierto que utiliza Java en su negocio. Los datos de transacciones y controles se almacenan en un sistema de gestión de bases de datos relacionales. El sistema SIDUNEA contiene ocho módulos; a continuación se describen algunas funcionalidades de estos que, de una forma u otra, persiguen objetivos similares a los procesos del RC.

MODSEL o Módulo de Selectividad**:** permite a la Aduana controlar la selección y el flujo de las declaraciones a través del sistema de procesamiento de la declaración de aduana. Contiene los controles para bloquear la liquidación de las declaraciones seleccionadas y tiene muchas funciones para hacer consultas y reportes.

MODCBR**:** es el módulo central del sistema, la base de los procedimientos de presentación, control, cobro y liquidación de los impuestos. Trabaja principalmente con declaraciones de mercancías, su ingreso al sistema, su verificación local y remota, registro, aplicación del resultado de selectividad y validación.

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

- ✓ Evita la imposición de patrones organizacionales en la administración de la aduana, ya que flexibiliza el proceso de administración en la aduana evitando llevar una estricta norma organizativa.
- ✓ El SIDUNEA actualiza permanentemente las tablas, acuerdos y normas por lo que el usuario realizará declaraciones de acuerdo a las legislaciones vigentes.
- ✓ El usuario podrá ver detalles de sus declaraciones almacenadas localmente en su estación de trabajo y simplemente actualizar los datos.
- ✓ Posee un módulo de selectividad que permite a la aduana acelerar el proceso de despacho y mejorar su capacidad de control. Selecciona declaraciones para control empleando criterios aleatorios de selección local.

- ✓ Permite una auditoria completa de los archivos que contienen el registro de una transacción comercial (importación/exportación).

SIDUNEA se puede configurar de acuerdo a las características nacionales de cada régimen aduanero, al arancel nacional y a la legislación de cada país, además de implementar los estándares internacionales para procesar los datos de comercio exterior ya acordados por la Organización Mundial de Aduanas (WCO) y por la Organización Internacional para la Estandarización (ISO).(4)

Entre sus funciones principales se encuentran la auditoría y gestión del historial de las personas que intervienen en los distintos procesos aduaneros. Cuando se gestiona el historial de las personas este sistema se basa solo en los trabajadores propios de la aduana, para acceder al registro de los externos lo hace mediante los servicios que brinda la cámara de comercio exterior del país donde se encuentre funcionando el sistema. (4)

El mismo fue el primer sistema computarizado que se instaló en la AGR pero no todos los módulos se pudieron implantar por incongruencias en su concepción con las características específicas de la misma. (1) Además como es un software con carácter privativo sus licencias se acercan a los ocho millones de dólares al año (4), factor que dificulta el acceso de los países subdesarrollados.

Korea Custom Service (KCS)

El Servicio de Aduanas de Corea (KCS) opera actualmente un total de 58 sistemas de información relacionados con la administración de aduanas y trabaja sobre la base de los sistemas EDI (Intercambio Electrónico de Datos), logrando la transferencia por medios electrónicos de información comercial o de negocios.

En el mundo esta solución se encuentra entre los primeros lugares por sus características. El KCS desarrolla un sistema de información, investigación y vigilancia para el enfrentamiento a acciones delictivas que se cometan en las aduanas. Además de estar compuesto por cinco subsistemas: Información General, Actividades de Investigación sobre Infracciones, Área de Investigación de la Empresa, Área de Vigilancia de Pasajeros, Vigilancia de los Buques y la Tripulación. Todas estas áreas en conjunto se encargan principalmente de la gestión de ex-convictos y sospechosos de violaciones en las aduanas, de la notificación a las autoridades en

caso violaciones así como la gestión de la información sobre los pasajeros de alto riesgo y de viajeros frecuentes. (5)

El KCS posee varias limitaciones realizando un análisis desde el punto de vista de un futuro despliegue en Cuba ya que es muy difícil adaptarlo a los procesos aduaneros propios de Cuba. Este sistema obtiene la información de los trabajadores y entidades a través de los servicios que brinda la cámara de comercio exterior de Corea. Mientras que la AGR de Cuba al tener una estructura diferente al resto de las aduanas del mundo gestiona los trabajadores y entidades con el sistema de RCE propio de la aduana y no hace uso de los servicios de la cámara de comercio exterior. El carácter privativo del sistema KCS va en contra de la tendencia cubana a la migración hacia el uso del software libre. Además de que el registro de las incidencias se basa sólo en los pasajeros y no toma en cuenta las incidencias cometidas por los trabajadores de la aduana.

1.2.2. Sistema informático en Cuba

SUA

La AGR presenta su propia solución informática denominada Sistema Único de Aduanas (SUA). La misma automatiza los cinco procesos aduaneros (despacho comercial, no comercial, de viajeros, de medios de transporte y de postal y envío), posee una base de datos única y centralizada, a la que acceden en línea todas las aduanas del país. En la actualidad especialistas del CADI, estudiantes y trabajadores de la UCI se han centrado en la realización de la reingeniería del mismo con el objetivo de crear un sistema con herramientas más modernas y potentes. Este posee módulos que validan y controlan las entradas de datos con nomencladores y clasificadores (aproximadamente 100) los cuales facilitan la flexibilidad del sistema, además de tener organizada la información y así asegurar la consistencia de los datos.

Actualmente este sistema no cumple con las necesidades y requerimientos que surgen en la AGR de poseer un registro de las operaciones que realizan los trabajadores externos y entidades, así tener una valoración de estas incidencias para brindar un mayor control sobre los mismos, elementos que son expuestos en la problemática del presente trabajo, de ahí la necesidad de desarrollar una solución que permita la gestión de un historial para las entidades y trabajadores externos pertenecientes al RCE de la AGR.

1.3. Descripción de la solución a desarrollar

Los especialistas del sistema GINA, en conjunto con los del CADI desde el año 2004 están interesados en el desarrollo de un módulo que se encargue de la gestión de las entidades y operaciones fundamentales de las personas en el sistema GINA.

Dicho módulo servirá para centralizar el historial de entidades dentro del sistema GINA haciendo que este pueda consultarse más rápidamente y de forma organizada permitiendo tener un resumen de las incidencias cometidas por las personas, sean trabajadores externos o no a una entidad en los distintos procesos que se llevan a cabo en la aduana y de los diferentes estados por los que ha transitado una entidad por rango de fechas.

1.4. Diseño del software

En el diseño se modela el sistema y se encuentra la forma para que soporte todos los requerimientos, incluyendo los requisitos no funcionales y otras restricciones, se asienta en el núcleo técnico del proceso de ingeniería del software y se aplica independientemente del paradigma de desarrollo utilizado. En la elaboración de la solución es preciso tener presente ciertos estándares, estilos y patrones que son muy útiles en la obtención de un diseño de software robusto.

1.4.1. Arquitectura de software

La arquitectura de software en el desarrollo de aplicaciones informáticas es un término muy frecuente y de gran referencia debido a que sirve para guiar la implementación y el desarrollo del sistema desde etapas tempranas del diseño. Existen diferentes definiciones acerca del tema, y aunque todas son consideraciones especializadas de distintos autores, coinciden en que a grandes rasgos se refiere a la estructura del sistema, constituida por componentes de software y relaciones entre estos. (6)

Algunas de estas definiciones han sido expuestas, por ejemplo:

“La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el

más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones". (7)

"La arquitectura del software de un programa o sistema de computación es la estructura o estructuras del sistema que comprende los elementos del software, las propiedades externamente visibles de esos elementos, y las relaciones entre ellos". (6)

Esta definición es un poco amplia, por lo que una de las más aplicadas es la establecida por la IEEE STD 1471-2000 (*Software Engineering Standards Committee of the IEEE Computer Society, 2000*): *"La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución."*

1.4.2. Estilos arquitectónicos

Los estilos arquitectónicos no son más que patrones pertenecientes a un tipo de vista concreto, que definen una serie de restricciones a los tipos de elementos y relaciones de la vista arquitectónica. Algunos estilos son universales, mientras que otros definen un tipo particular de software. En cualquier caso, la arquitectura de un sistema está compuesta por vistas pertenecientes a múltiples estilos. (8)

Estos sirven para sintetizar estructuras de soluciones, definen los posibles patrones de las aplicaciones y permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales. Además de que conjugan elementos, conectores, configuraciones y restricciones. Al estipular los conectores como elemento de juicio de primera clase, el concepto de estilo, incidentalmente, se sitúa en un orden de discurso y de método que el modelado orientado a objetos en general y UML (del inglés, Unified Modeling Language) en particular no cubren satisfactoriamente. La descripción de un estilo se puede formular en lenguaje natural o en diagramas, pero lo mejor es hacerlo en un lenguaje de descripción arquitectónica o en lenguajes formales de especificación.

A diferencia de los patrones de diseño, que son centenares, los estilos se ordenan en seis o siete clases fundamentales y unos veinte ejemplares, como máximo. Es digno de señalarse el empeño por incluir todas las formas existentes de aplicaciones en un conjunto de dimensiones

tan modestas. Las arquitecturas complejas o compuestas resultan del agregado o la composición de estilos más básicos.

Estilos de llamada y retorno

- ✓ *Model-View-Controller* (MVC).
- ✓ Arquitecturas en Capas.
- ✓ Arquitecturas Orientadas a Objetos.
- ✓ Arquitecturas Basadas en Componentes.

1.4.3. Patrones

Cada patrón describe un problema que ocurre y una y otra vez en un entorno determinado y describe también el núcleo de la solución del problema, de forma que puede utilizarse un millón de veces sin tener que hacer dos veces lo mismo. (9)

Dentro de las definiciones de patrones se encuentra la propuesta por Craig Larman: *"Un patrón es un par problema/solución con nombre que se puede aplicar en nuevos contextos, con consejos sobre cómo aplicarlo en nuevas situaciones, o sea, un patrón es una descripción de un problema bien conocido que suele incluir: descripción, escenario de uso, solución concreta, consecuencias de utilizar el patrón, ejemplos de implementación y lista de patrones relacionados."* (10)

Patrones de arquitectura

Los patrones de arquitectura están relacionados fundamentalmente con la estructura de un sistema de software. Especifican un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes. Describen un problema particular y recurrente del diseño, que surge en un contexto específico, y presenta un esquema genérico y probado de su solución. (11)

Patrón Modelo Vista Controlador (MVC): Es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres capas o componentes distintos. El patrón MVC se utiliza frecuentemente en aplicaciones web, donde la vista es la página HTML (*del inglés, Hypertext Markup Language*) que se visualiza en el navegador y el código que proporciona de datos dinámicos a la página, el modelo es el Sistema

de Gestión de Base de Datos y la lógica de negocio y el controlador es el responsable de recibir los eventos de entrada desde la vista, enviarlos al modelo y manejar también las respuestas del modelo y transmitirlas hacia la vista. (12)

Modelo: El modelo es un conjunto de clases que representan la información del mundo real que el sistema debe procesar. El modelo desconoce la existencia de las vistas y del controlador. Ese enfoque suena interesante, pero en la práctica no es aplicable pues deben existir interfaces que permitan a los módulos comunicarse entre sí. Esta es la recepción específica del dominio de la información sobre la cual funciona la aplicación. El modelo es una forma de llamar la capa de dominio. La lógica de dominio añade significados a los datos.

Vista**:** Las vistas son el conjunto de clases que se encargan de mostrar al usuario la información contenida en el modelo. Una vista está asociada a un modelo, pudiendo existir varias vistas asociadas al mismo modelo. Una vista obtiene del modelo solamente la información que necesita para desplegar y se actualiza cada vez que el modelo del dominio cambia por medio de notificaciones generadas por el modelo de la aplicación.

Controlador**:** El controlador es un objeto que se encarga de dirigir el flujo del control de la aplicación debido a mensajes externos, como datos introducidos por el usuario u opciones del menú seleccionadas por él. A partir de estos mensajes, el controlador se encarga de modificar el modelo o de abrir y cerrar vistas. El controlador tiene acceso al modelo y a las vistas, pero las vistas y el modelo no conocen de la existencia del controlador.

Patrones de diseño

Los Patrones de Diseño no son más que soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan, son descripciones de clases cuyas instancias colaboran entre sí y brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares.

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.

- ✓ **Fachada:** Patrón estructural que permite proveer una interfaz unificada y sencilla como intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas.
- ✓ **Mediador**: Patrón de comportamiento que coordina las relaciones entre sus asociados. Permite la interacción de varios objetos, sin generar acoples fuertes en esas relaciones.
- ✓ **Cadena de Responsabilidad**: La cadena de responsabilidad se encarga de evitar el acoplamiento del remitente de una petición a su receptor, dando a más de un objeto la posibilidad de manejar la petición. (13)
- ✓ **Decorator:** Responde a la necesidad de añadir dinámicamente funcionalidad a un objeto. Esto permite no tener que crear sucesivas clases que hereden de la primera incorporando la nueva funcionalidad, sino otras que la implementan y se asocian a la primera. (14)

GRASP

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Tienen como motivación:

- ✓ La asignación de responsabilidades como la habilidad más importante en el análisis y diseño orientado por objetos.
- ✓ Respetar los principios fundamentales como uno de los factores críticos, para obtener diseños reutilizables, mantenibles y extendibles.

1.5. Metodología, lenguajes y herramientas empleadas en el desarrollo

En la actualidad se hace necesario el uso de procedimientos, técnicas y herramientas que posibiliten la creación de un producto de software fiable y fácilmente mantenible. Los sistemas de software representan un elemento que ha contribuido considerablemente al desarrollo de numerosas empresas. Para poder realizar estos sistemas con la mejor calidad posible se hace necesaria la utilización de ciertas estrategias que garanticen un buen resultado o mejoren las características del producto desarrollado.

A continuación se muestra una descripción de algunas de las tecnologías definidas para el desarrollo del sistema GINA haciendo énfasis en sus principales características.

1.5.1. Metodología de desarrollo

Las metodologías de desarrollo de software describen los pasos que se deben seguir para la producción de un determinado producto informático. Su principal objetivo es crear software de calidad e ir mejorando siempre lo que ya este hecho. Además de guiar a los desarrolladores al crear un nuevo software, pero los requisitos de un software a otro son tan variados y cambiantes, que ha dado lugar a que exista una gran variedad de metodologías para la creación del software.

En CEIGE al cual pertenece el Departamento de Soluciones para la Aduana, cuenta con un modelo estandarizado, que establece las distintas fases por las que se debe transitar y el conjunto de artefactos a generar en cada una de ellas. Este modelo de desarrollo de software detalla el ciclo de vida de sus proyectos con la incorporación de los distintos subprocesos dictados por el Nivel II de CMMI (Capability Maturity Model Integration), certificación obtenida por el centro en julio de 2011 y reconocida por el SEI (Software Engineering Institute) como aval de la calidad de su proceso de desarrollo de software. (15)

El modelo de desarrollo tiene en cuenta en el ciclo de vida de los proyectos las fases y actividades por áreas de procesos que plantea el nivel dos de CMMI. En la siguiente figura *(Figura 1.1)* se puede apreciar las siete fases por las que deben transitar.

Figura 1.1 Ciclo de vida de proyectos del CEIGE. Tomado de (15)

- ✓ Estudio preliminar: se realizan las actividades de planeación y un estudio inicial de las necesidades del cliente que permite obtener información fundamental acerca del alcance del proyecto y realizar estimaciones de tiempo, esfuerzo y costo.
- ✓ Modelado del negocio: es donde se comprende los procesos de negocio de la organización y el funcionamiento del negocio que se desea automatizar para tener garantías de que el software desarrollado va a cumplir su propósito.
- ✓ Requisitos: es donde se desarrolla un modelo del sistema que se va a construir. Incluye los artefactos que describen todas las interacciones que tendrán los usuarios con el software y que responden a los requisitos funcionales del sistema. Se especifican los requisitos funcionales y no funcionales.
- ✓ Análisis y diseño: se modela el sistema para que soporte todos los requisitos que contribuye a una arquitectura sólida y estable.
- ✓ Implementación: se lleva a cabo la implementación del sistema en términos de componentes, es decir, ficheros de código fuente, scripts, ejecutables y similares.
- ✓ Pruebas internas: se desarrollan las pruebas del grupo de calidad del centro verificando el resultado de la implementación. Permite identificar posibles errores en la documentación y el software.
- ✓ Pruebas de liberación: Se aplican pruebas diseñadas e implementadas por el Laboratorio Industrial de Pruebas de Software a todos los entregables de los proyectos antes de ser entregados al cliente para su aceptación.

De las fases expuestas anteriormente, se emplearon en el desarrollo de la solución: realizar un estudio preliminar, el modelado del negocio, requisitos, el análisis y diseño, la implementación y la realización de las pruebas internas.

1.5.2. Lenguaje de modelado

La estandarización de un lenguaje de modelado es de suma importancia, debido a que es la parte principal del proceso de comunicación que requieren todos los involucrados en un proyecto informático de desarrollo de software. Razón por la cual las organizaciones usan el lenguaje de modelado en combinación con una metodología de desarrollo de software para

avanzar de una especificación inicial a un plan de implementación y para comunicar dicho plan a todo un equipo de desarrolladores.

El Lenguaje Unificado de Modelado (UML) es el lenguaje de modelado de sistemas software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). El UML es la notación (principalmente gráfica) que usan los métodos para expresar un diseño. El proceso indica los pasos que se deben seguir para llegar a un diseño. Además, ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

El UML es el lenguaje para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. (16)

1.5.3. Herramienta CASE para el modelado

Las herramientas CASE[5] actualmente brindan una gran cantidad de componentes que incluyen la mayoría de los requisitos necesarios para el desarrollo de los sistemas, han sido creadas con una gran exactitud en cuanto a las necesidades de los desarrolladores de software para la automatización de procesos incluyendo el análisis, diseño e implantación.

En la actualidad muchas empresas se han extendido a la adquisición de herramientas CASE, con el fin de automatizar sus procedimientos administrativos. Dentro de estas herramientas se encuentra Visual Paradigm for UML.

Visual Paradigm for UML

Visual Paradigm for UML (VP-UML) es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción,

[5] **CASE:** siglas en ingles que se utilizan para referirse a Ingeniería de Software Asistida por Computadora

pruebas y despliegue. Permite dibujar todos los tipos de diagramas (diagramas de casos de uso, diagramas de requerimiento y diseño de bases de datos relacionales), código inverso, generar código desde diagramas y generar documentación. (17) También proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Presenta licencia gratuita y comercial. Es fácil de instalar y actualizar y compatible entre ediciones.

Esta herramienta es multiplataforma y se integra con algunas herramientas realizadas en Java entre las cuales se puede citar el entorno de desarrollo integrado NetBeans. Con VP-UML, el equipo de desarrollo de software puede realizar el análisis y diseño de sistemas con eficacia.

Debido a las características y facilidades planteadas anteriormente se seleccionó la herramienta Visual Paradigm for UML en su versión 3.4 para la realización del diseño de la solución.

1.5.4. Lenguajes de programación

Un lenguaje de programación es un lenguaje diseñado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Este permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que este pueda comunicarse con los dispositivos hardware y software existentes. Por lo tanto, un lenguaje de programación es un modo práctico para que los seres humanos puedan dar instrucciones a un ordenador o computadora. (18)

En la actualidad para el desarrollo de las diferentes soluciones informáticas existen varios lenguajes de programación, partiendo desde aquellos privativos por los cuales es necesario pagar una patente y llegando hasta los lenguajes de carácter libre cuyo soporte es dado por la comunidad de usuarios que deseen colaborar.

Lenguajes del lado del cliente a emplear

Cuando se programa una página web, en la mayoría de los casos se utiliza los que se conocen como “lenguajes del lado del cliente”. Es decir, que el servidor no interviene en el proceso de crear la página web solicitada por el usuario. Para el desarrollo de la solución se eligieron para realizar la programación en el lado del cliente los siguientes lenguajes:

XHTML:

El lenguaje HTML es un estándar reconocido en todo el mundo y cuyas normas define un organismo sin ánimo de lucro llamado W3C (World Wide Web Consortium). Como se trata de un estándar reconocido por todas las empresas relacionadas con el mundo de Internet, una misma página HTML se visualiza de la misma manera en cualquier navegador de cualquier sistema operativo. (18) 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.

El lenguaje XHTML es muy similar al lenguaje HTML. De hecho, XHTML no es más que una adaptación de HTML al lenguaje XML (Ver Figura 2). Técnicamente, HTML es descendiente directo del lenguaje SGML4 (Standard Generalized Markup Language), mientras que XHTML lo es del XML (que a su vez, también es descendiente de SGML[6]) (Ver **Figura 1. 2)** (19)

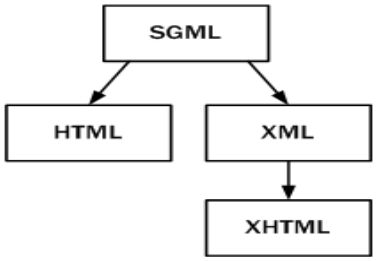


Figura 1. 2 Esquema de la evolución de HTML y XHTML. Tomado de (18)

CSS:

El lenguaje CSS, hojas de estilo en cascada que viene del inglés *Cascading Style Sheets*, del que toma sus siglas. Es usado para definir la presentación de un documento estructurado escrito en HTML y XHTML con el fin de separar la estructura de la presentación y es imprescindible para la creación de páginas web complejas. La separación de los contenidos y su presentación posee numerosas ventajas, ya que obliga a crear documentos HTML/XHTML bien definidos y con significado completo (también llamados “documentos semánticos”).

[6] **SGML**: Consiste en un sistema para la organización y etiquetado de documentos.

Además, mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes. (20)

En el sistema GINA es empleado para decorar el aspecto de la capa de presentación. Describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura. Ofrece a los desarrolladores el control total sobre estilo y formato de sus documentos.

JavaScript:

Es un lenguaje de programación que realiza acciones dentro del ámbito de una página web. Su compatibilidad con la mayoría de los navegadores modernos, lo posiciona como el lenguaje de programación del lado del cliente más utilizado. Con JavaScript se puede crear efectos especiales en las páginas y definir interacción con el usuario. Se pueden crear páginas interactivas y técnicamente, es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. Entre sus características se encuentran la de ser un lenguaje orientado a objetos, con funciones, su código se integra en las páginas XHTML, incluido en las propias páginas. Además de ser simple ya que no hace falta tener conocimientos avanzados de programación para poder hacer un programa en JavaScript. (21)

En el sistema GINA se utiliza, principalmente, para manejar objetos dentro de las páginas web. Dichos objetos facilitan la programación de páginas interactivas, a la vez que se evita la posibilidad de ejecutar comandos que puedan ser peligrosos para la máquina del usuario, tales como formateo de unidades y modificación de archivos.

Lenguajes del lado del servidor a emplear

Un lenguaje del lado del servidor es aquel que se ejecuta en el servidor web, justo antes de que se envíe la página a través de Internet al cliente. Las páginas que se ejecutan en el servidor pueden realizar accesos a bases de datos, conexiones en red, y otras tareas para crear la página final que verá el cliente. Los lenguajes más utilizados para el desarrollo de páginas dinámicas son el ASP, JSP, PERL y PHP. (22) El sistema GINA se ha implementado sobre la base del PHP como lenguaje de programación del lado del servidor.

PHP es un lenguaje de programación generalmente usado en la programación de sitios web dinámicos y actualmente es casi el lenguaje de desarrollo de sitios más usado en todo el

mundo. Originalmente fue desarrollado en Perl[7]. PHP al principio significaba Personal Home Page pero con el tiempo como ya es desarrollado por otro grupo se llama PHP Hypertext Preprocesor. (23)

Para el desarrollo de la solución se empleó PHP en su versión 5.3.8. Debido a que este al ser un lenguaje interpretado, solo se necesita un navegador web para ejecutarlo. Los scripts se ejecutan remotamente y el resultado aparece en la máquina cliente (local). Posee soporte para muchos tipos de bases de datos, entre los principales están MySQL, PostgreSQL, Oracle, entre otras. Es embebido en código HTML y orientado a objetos.

1.5.5. Marco de trabajo

Un marco de trabajo (*framework)*, en el desarrollo de software es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Además contiene un conjunto de conceptos, prácticas y criterios estandarizados, que nos ayuda a enfocar un tipo de problema en particular, se diseñan para facilitar el desarrollo de un software, permitiendo ahorrar tiempo y eliminando código innecesario. Se pueden considerar como el esqueleto del programa. (24)

Ext JS *Framework*.

Es un *framework* diseñado para la creación de páginas web dinámicas del lado del cliente basado en lenguaje JavaScript, permite una gran reutilización de componentes, personalización de los mismos, haciendo uso del manejo de tecnologías como AJAX (*del inglés, Asynchronous JavaScript And XML*) para el intercambio asincrónico de los datos entre el cliente y el servidor, además de lo ventajoso que puede representar la similitud de su aspecto con el de una aplicación de escritorio. Sus características principales son: gran desempeño, componentes de interfaz de usuario personalizables, con buen diseño y documentación. (25)

[7] **Perl:** lenguaje de programación que soporta estructuras de datos complejas, es un modelo de programación orientada a objetos.

Para el desarrollo del sistema GINA se utiliza este *framework* en su versión 3.0, para validar los datos antes de ser enviados al servidor y para proveer al sistema de una buena apariencia y usabilidad de forma tal que se simule una aplicación de escritorio en el navegador web.

Symfony Framework.

Es un completo marco de trabajo diseñado para optimizar el desarrollo de las aplicaciones web mediante algunas de sus características principales. Este separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación.

Está desarrollado completamente en PHP 5. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Este *framework* es compatible con la mayoría de los gestores de bases de datos, como MySQL, PostgreSQL, Oracle y Microsoft SQL Server. Además de ser un framework multiplataforma. (26)

Este *framework* de desarrollo, en su versión 1.2.8, es la herramienta base utilizada para llevar a cabo la implementación de la lógica de negocio del sistema GINA, debido a que brinda facilidades para el desarrollo con el lenguaje PHP. Utiliza metodologías y patrones de diseño que brindan la posibilidad de reutilizar el código. Presenta buena seguridad, es multiplataforma y sencillo de configurar. Permite por medio de Propel[8] el mapeo de las base de datos, transformando las tablas en clases del negocio. Crea formularios seguros para la entrada de información, soporta internacionalización.

1.5.6. Entorno de Desarrollo Integrado (IDE)

Un entorno de desarrollo integrado, no es más que un programa informático compuesto por una serie de herramientas de programación utilizadas por los programadores para desarrollar código, consisten en un editor de código, un compilador, un depurador y un constructor de

[8] **Propel:** es un kit de mapeo objeto-relacional (ORM) de código abierto escrito en PHP. Es además una parte integral del *framework Symfony*.

interfaz gráfica. Este puede dedicarse a un solo lenguaje de programación o bien puede utilizarse para varios.

NetBeans IDE

NetBeans IDE es una herramienta libre y gratuita sin restricciones de uso, pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extenderlo. Posee un amplio soporte para el lenguaje PHP así como para los *framework* de trabajo *Symfony* y Ext JS.

Para el desarrollo de la solución se ha elegido NetBeans IDE, en su versión 7.2. Este presenta varias características integrado a PHP como la creación de proyectos en PHP así como depuración, provee integración con el *framework Symfony*, con el SGBD Oracle y con el sistema de control de versiones *Subversion.*

1.5.7. Gestor de Base de Datos

Un Sistema Gestor de Bases 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. (27)

Se considera a Oracle como uno de los sistemas de bases de datos más completos, destacando:

- ✓ Soporte de transacciones.
- ✓ Estabilidad.
- ✓ Escalabilidad.
- ✓ Soporte multiplataforma.

Su dominio en el mercado de servidores empresariales ha sido casi total hasta hace poco, recientemente sufre la competencia del Microsoft SQL Server de Microsoft y de la oferta de otros RDBMS con licencia libre como PostgreSQL, MySQL o Firebird. Las últimas versiones de Oracle han sido certificadas para poder trabajar bajo GNU/Linux.

Las principales funcionalidades aportadas por todo el SGBD Oracle son:

- ✓ Soporte y tratamiento de una gran cantidad de datos (Gbytes).
- ✓ Soporte de una gran cantidad de usuarios accediendo concurrentemente a los datos.
- ✓ Seguridad de acceso restringiendo dicho acceso a las necesidades de cada usuario.
- ✓ Integridad referencial en su estructura de base de datos.
- ✓ Conectividad entre las aplicaciones de los clientes en su puesto de trabajo y el servidor de base de datos Oracle (estructura cliente/servidor).
- ✓ Conectividad entre base de datos remotas (estructura de bases de datos distribuidas).
- ✓ Portabilidad.
- ✓ Compatibilidad.

En las soluciones implementadas para la AGR el gestor de base de datos que se utiliza es Oracle en su versión 11g.

1.6. Conclusiones parciales

En el capítulo se demuestra la necesidad e importancia que tiene para la Aduana contar con un sistema informático que gestione el historial de los trabajadores externos y entidades, se realiza un análisis de los sistemas extranjeros que son utilizados para tales fines, los cuales son sistemas privativos que imposibilita el acceso de los países subdesarrollados así como la imposibilidad de adaptación a los procesos aduaneros propios de la AGR. Se analizaron diferentes temáticas importantes en el desarrollo de un software como son: el diseño y arquitectura de software, los patrones de arquitectura y diseño, metodologías, lenguajes y tecnologías a utilizar.

Como metodología de desarrollo de software se utilizó el modelo de desarrollo definido por el centro CEIGE. Se determinó emplear Ext JS en su versión 3.0 como *framework* de trabajo en el lado del cliente, Symfony en su versión 1.2.8 como *framework* de trabajo en el lado del servidor y como SGBD se empleó Oracle en su versión 11g, ya que son las tecnologías y herramientas que define el proyecto.

Capítulo 2: Análisis y Diseño

2.1. Introducción

En el capítulo se describen las principales características del negocio, se exponen los requisitos funcionales definidos para el sistema, enumerando en este sentido las técnicas de captura y validación de requerimientos. Además se realiza la modelación del sistema mediante el diseño del mismo. Estos factores permiten contribuir a una arquitectura estable, sólida y que soporte los requisitos definidos, siendo un punto de partida en las actividades de la implementación. Se muestra los patrones empleados, así como una serie de artefactos dentro de los que se encuentran los diagramas de secuencia, el modelo conceptual y el modelo entidad-relación.

2.3. Modelo conceptual

A continuación se observa *(Figura 2.1)* el modelo conceptual del negocio que presenta las entidades importantes y sus relaciones. Este sirve como una primera aproximación al diseño definitivo del modelo de datos.

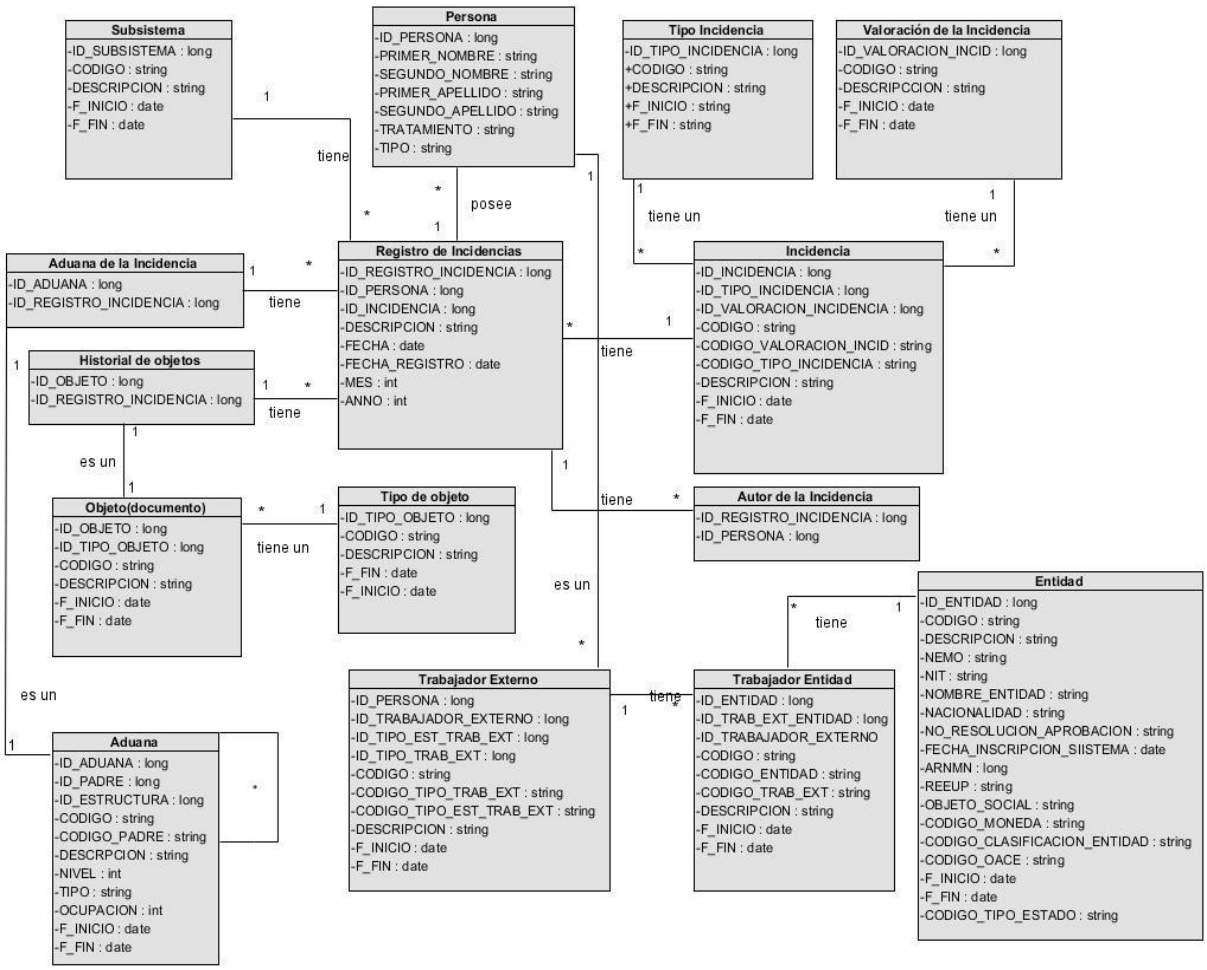


Figura 2.1 Modelo Conceptual de Historial de Entidades y Trabajadores Externos.

2.4. Especificación de los requerimientos del sistema

Especificar los requerimientos del sistema sirve como base para las actividades de ingeniería de software, describe la función y las restricciones que regirán su desarrollo, por lo cual deben tenerse en cuenta las técnicas que son más factibles utilizar para la captura y la validación de estos requisitos; a continuación se exponen las técnicas empleadas además de un listado de los mismos.

2.4.1 Técnicas para la captura de requerimientos

Siguiendo el procedimiento para la Ingeniería de Requisitos en el Departamento de Desarrollo de Soluciones para la Aduana del centro CEIGE y teniendo en cuenta las características del cliente, se decidió utilizar como técnicas que permitirán a los analistas la recopilación y

obtención de la información necesaria para la elicitación de requisitos la entrevista, la observación, la tormenta de ideas y los talleres. (28)

Se combinaron las siguientes técnicas: entrevista, observación y tormenta de ideas. Se **entrevistaron** a varios especialistas y funcionarios de la aduana del RCE, en visitas realizadas a las instalaciones, ya que las entrevistas permiten un intercambio más abierto con los especialistas, mediante preguntas que esclarecen con precisión el funcionamiento de todo el trabajo en el RCE de Aduana. La **observación** se llevó a cabo para percibir cómo se desarrollan las operaciones de registro de los procesos que realizan los trabajadores externos y entidades, pudiendo captar directamente las particularidades de estos procesos en las visitas realizadas; y la **tormenta de ideas** donde el equipo de desarrollo en conjunto con la especialista de la aduana acumularon ideas para tener una perspectiva general de las necesidades del sistema.

2.4.2. Requerimientos funcionales

Los requerimientos funcionales (RF) son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requerimientos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer. (29)

Los RF no son más que las condiciones o capacidades que el sistema debe cumplir, debe constituir el punto de partida para identificar qué debe hacer el sistema. Debe comprenderlo tanto los desarrolladores como los usuarios.

A continuación se relacionan los requerimientos definidos para el sistema modelado:

RF1 Gestionar una incidencia de un trabajador externo: el sistema debe permitir la gestión de las incidencias de un trabajador, dando la posibilidad de registrarlas, modificarlas y de eliminarlas.

RF1.1 Insertar una incidencia.

RF1.2 Modificar una incidencia.

RF1.3 Eliminar una incidencia.

RF2 Obtener historial de Incidencias de un trabajador externo: el sistema debe permitir que dado un trabajador externo se obtenga el listado de las incidencias que posee el mismo y sus datos generales.

RF3 Obtener historial de estados de un trabajador externo: el sistema debe permitir obtener dado un trabajador externo el registro de los estados por los que ha transitado por rango de fechas, así como sus datos en ese estado.

RF4 Obtener historial de los estados de una entidad: el sistema debe permitir obtener dada una entidad el registro de los estados por los que ha transitado por rango de fechas, así como sus datos en ese estado.

RF5 Obtener dado un tipo de incidencia los trabajadores que han incurrido en incidencias de ese tipo: el sistema debe permitir obtener un listado de todos los trabajadores que posean una incidencia del tipo dado.

RF6 Obtener dado un subsistema los trabajadores que tienen incidencias registradas: el sistema debe permitir obtener un listado de los trabajadores que posean al menos una incidencia registrada por el subsistema dado.

RF7 Obtener los trabajadores que han incurrido en incidencias de un tipo dado y que han sido registradas por un subsistema dado: el sistema debe permitir obtener un listado de los trabajadores que posean al menos una incidencia registrada por el subsistema dado.

RF8 Obtener los trabajadores externos que han incidido en un tipo de objeto dado: el sistema debe permitir obtener un listado de los trabajadores que posean incidencias relacionadas con el tipo de objeto dado.

RF9 Obtener los trabajadores según un tipo de objeto, un tipo de incidencia y un subsistema: el sistema debe permitir obtener un listado de los trabajadores que posean incidencias relacionadas con el tipo de objeto, el tipo y el subsistema dado.

RF10 Obtener los trabajadores que tengan incidencia registradas por un subsistema y que han incidido en un tipo de objeto dado.

RF11 Obtener dada una entidad los trabajadores que tienen incidencias: el sistema debe permitir obtener un listado de los trabajadores de la entidad dada que posean al menos una incidencia.

RF12 Obtener dada una entidad y fecha los trabajadores que tienen incidencias: el sistema debe permitir obtener todos los trabajadores de la entidad dada que tengan al menos una incidencia en la fecha.

RF13 Obtener dada una fecha los trabajadores que tienen incidencias: el sistema debe permitir obtener todos los trabajadores que tengan al menos una incidencia en la fecha.

RF14 Obtener los trabajadores que tuvieron alguna incidencia en un rango de fecha dado.

RF15 Obtener los trabajadores que tengan incidencia registradas por un subsistema, de un tipo dado, que han incidido en un tipo de objeto determinado, que pertenezcan a una entidad y que tenga una fecha dada o un rango de fecha.

Descripción del requisito Obtener historial de Incidencias de un Trabajador Externo:

Precondiciones	El especialista de RCE se ha autenticado en el sistema y debe tener los permisos para obtener el registro de las incidencias. El trabajador debe estar registrado en el sistema.
Flujo de eventos	
Flujo básico Obtener historial de Incidencias de un Trabajador Externo	
1	Se selecciona en el menú principal de Registro Central de Entidades la opción “Gestión de trabajadores”.
2	Se muestra el listado con todos los trabajadores registrados en el sistema.
3	Se selecciona un trabajador y se da clic en la opción “Historial Trabajador”.
4	Se selecciona la opción “Ver historial de incidencias trabajador”.
5	Se muestra la pantalla con los datos del trabajador y sus incidencias.
6	Concluye el requisito
Pos-condiciones	
1	Se muestra el historial del trabajador.
Validaciones	
1	N/A
Requisitos especiales	N/A
Asuntos pendientes	N/A

En el **anexo #1** se puede obtener la descripción del resto de los requisitos, así como sus prototipos de interfaz.

2.4.3 Técnicas para la validación de requerimientos

Siguiendo el procedimiento para la Ingeniería de Requisitos en el Departamento de Desarrollo de Soluciones para la Aduana del CEIGE, el cual plantea sobre las técnicas para la validación de los requerimientos lo siguiente:

En el procedimiento propuesto se recomienda el uso de varias técnicas para la correcta validación de los requisitos en el Departamento de Soluciones para la Aduana las cuales son: revisiones del documento de requerimientos, construcción de prototipos, matrices de trazabilidad y generación de casos de pruebas. (28)

La validación de los requerimientos se realizó con la combinación de las siguientes técnicas: revisiones del documento de requerimientos y construcción de prototipos. Se realizaron diversas **revisiones al documento de requerimientos** con la participación de los analistas del proyecto, jefe del subsistema, jefe del proyecto y especialistas del CADI para logar una correcta interpretación de la información transmitida, los señalamientos planteados fueron recogidos y aplicados posteriormente. Además se efectuó la **construcción de prototipos**, los cuales fueron presentados por cada requerimiento de software, aclarando con la especialista principal de Sistemas Automatizados del CADI, si las necesidades del RCE de la aduana fueron cubiertas por el sistema.

2.5. Patrones utilizados

Con la utilización del marco de trabajo Symfony, el sistema a desarrollar implementa varios de los patrones de diseño y arquitectónicos utilizados en la actualidad, ofreciendo a los desarrolladores la utilización de buenas prácticas en la implementación y la no preocupación de la aplicación de los mismos, brindándole al equipo de desarrollo y al producto cuantiosas ventajas.

Patrón Modelo Vista Controlador (MVC)

Symfony toma lo mejor de la arquitectura MVC y la realiza de modo que el desarrollo de aplicaciones sea rápido y sencillo. El controlador frontal es un componente que sólo tiene código relativo al MVC, por lo que no es necesario crear uno, debido a que el f*ramework Symfony* lo genera de forma automática. (26)

En la capa del Modelo se encuentran las clases, que son generadas de forma automática según la estructura de la base de datos. La librería Propel se encarga de esta generación automática, ya que crea el *esqueleto* o estructura básica de las clases y genera automáticamente el código necesario. En Symfony, el acceso y la modificación de los datos que se almacenan en la base de datos, se realiza mediante objetos. (26) A medida que el desarrollo de un proyecto va avanzando, puede ser necesario agregar métodos y propiedades personalizadas en los objetos del modelo, esto trae consigo que se aumenten las tablas o columnas. Asimismo, cada vez que se modifica se deben regenerar las clases del modelo de objetos. Si se añaden los métodos personalizados en las clases que se generan, cada vez que se vuelvan a generar esas clases estos métodos se borrarían.

La vista es la encargada de originar las páginas que son mostradas como resultado de las acciones, donde se encuentra el layout, que es común para todas las páginas de la aplicación. Además de que su lógica se puede transformar en un archivo de configuración sencillo, sin necesidad de programarla. (26)

Patrones GRASP[9]

Experto**:** es uno de los patrones que más se utiliza cuando se trabaja con Symfony, con la inclusión de la librería Propel para mapear la Base de Datos. El f*ramework Symfony* utiliza esta librería para realizar su capa de abstracción en el modelo, encapsular toda la lógica de los datos y generar las clases con todas las funcionalidades comunes de las entidades, las clases de abstracción de datos (Peer del Modelo) poseen un grupo de funcionalidades que están relacionadas directamente con la entidad que representan y contienen la información necesaria de la tabla que representan. (26) Un ejemplo del uso de este patrón se puede evidenciar

[9] **GRASP:** es un acrónimo que significa General Responsibility Assignment Software Patterns (*en español, patrones generales de software para asignar responsabilidades*).

cuando se generan las clases del modelo de la tabla *HP_REGISTRO_INCIDENCIA* se crean las siguientes clases:

- ✓ *HpRegistroIncidencia.php*
- ✓ *BaseHpRegistroIncidencia.php*
- ✓ *HpRegistroIncidenciaPeer.php*
- ✓ *BaseHpRegistroIncidenciaPeer.php*
- ✓ *HpRegistroIncidenciaForm.class.php*
- ✓ *Base HpRegistroIncidenciaForm.class.php*
- ✓ *HpRegistroIncidenciaMapBuilder.php*

Estas clases contienen toda la informacion necesaria para representar la tabla *HP_REGISTRO_INCIDENCIA,* así como varias funcionalidades relacionadas con la misma.

Creador**:** este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. Las clases *Peer* crean los objetos para guardar los formularios. A continuación se muestra un ejemplo en el que la clase **HpRegistroIncidenciaPeer()** crea una instancia de la clase **HpRegistroIncidenciaForm()** con la finalidad de guardar el registro de una incidencia.

```
// lib/model/rc/historial/model/HpRegistroIncidenciaPeer.php
class HpRegistroIncidenciaPeer extends BaseHpRegistroIncidenciaPeer {
  static function registrarIncidencia($datosIncidencia) {
   //se crea el objeto registro incidencia
   $objRegIncideciaForm = new HpRegistroIncidenciaForm();
   }
}
```

Alta Cohesión**:** cada elemento dentro del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos. El f*ramework Symfony* permite la organización del trabajo en cuanto a la estructura del proyecto y la asignación de responsabilidades con una alta cohesión. (26) La clase historial/Actions.class.php es un ejemplo de ello, está formada por varias funcionalidades que están estrechamente relacionadas, siendo responsable de definir las acciones para las plantillas y colaborar con otras para realizar diferentes operaciones e instanciar objetos.

Bajo Acoplamiento**:** este patrón 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. (26) La clase historial/Actions.class.php hereda únicamente de sfActions para alcanzar un bajo acoplamiento de clases. Las clases que implementan la lógica del negocio y de acceso a datos se encuentran en el modelo, las cuales no tienen asociaciones con las de la vista o el controlador, lo que proporciona que la dependencia en este caso sea baja.

Controlador**:** se emplea la página historial_dev.php, 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.

Patrones GOF

Instancia única (*singleton*): admite exactamente 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. (26) 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.

Registro (*registry)*: este patrón es muy útil para los desarrolladores en la Programación Orientada a Objetos, siendo un medio sencillo y eficiente de compartir datos y objetos en la aplicación sin la necesidad de preocuparse por conservar numerosos parámetros o hacer uso de variables globales. Este patrón se aplica en la clase sfConfig, que es la encargada de acumular todas las variables de uso global en el sistema.

Decorador: la clase abstracta *sfView*, padre de todas las vistas, contienen un decorador para permitir agregar funcionalidades dinámicamente. El archivo *layout.php*, conocido como plantilla

global, contiene el código HTML que es tradicional en todas las páginas del sistema, para no tener que repetirlo.

2.6. Modelo del diseño

El modelo del diseño muestra las clases del sistema y los diferentes tipos de relaciones entre ellas. Además de servir como abstracción de la implementación y es, de ese modo, utilizado como una entrada fundamental de las actividades de implementación.

El modelo del diseño de la presente investigación se encuentra compuesto por el diagrama de clases del diseño, el modelo de datos y por los diagramas de secuencia.

2.6.1. Diseño del modelo de datos

El modelado y diseño de la base de datos es un factor importante para el desarrollo de un sistema. El objetivo del diseño es generar un conjunto de esquemas de relaciones que permitan almacenar la información con un mínimo de redundancia, pero que a la vez faciliten la recuperación de la información. Este modelo describe los datos como entidades, relaciones (vínculos) y atributos, y permite representar el esquema conceptual de una base de datos de forma gráfica mediante los diagramas Entidad-Relación (ER).

En la siguiente figura (**Figura 2.2**) se muestra el diagrama ER de la solución, el cual en su totalidad está compuesto por 23 tablas, divididas en diferentes colores para mejor entendimiento del mismo. Las tablas de color gris son tablas de nomencladores ubicadas en el esquema del subsistema de Tablas de Control; las tablas de color verde son las ubicadas en el esquema de Historial Persona, las de color naranja las ubicadas en el subsistema de RC y el color azul representa a la tabla Persona, encargada del almacenamiento de las personas naturales dentro del sistema.

En el **anexo #2** se puede obtener una descripción de cada tabla ubicada en el esquema Historial Persona y de las tablas TC_INCIDENCIA, TC_TIPO_INCIDENCIA y TC_VALORACION_INCIDENCIA que se incluyeron en el esquema del subsistema de Tablas de Control, profundizando en cada uno de sus atributos y relaciones.

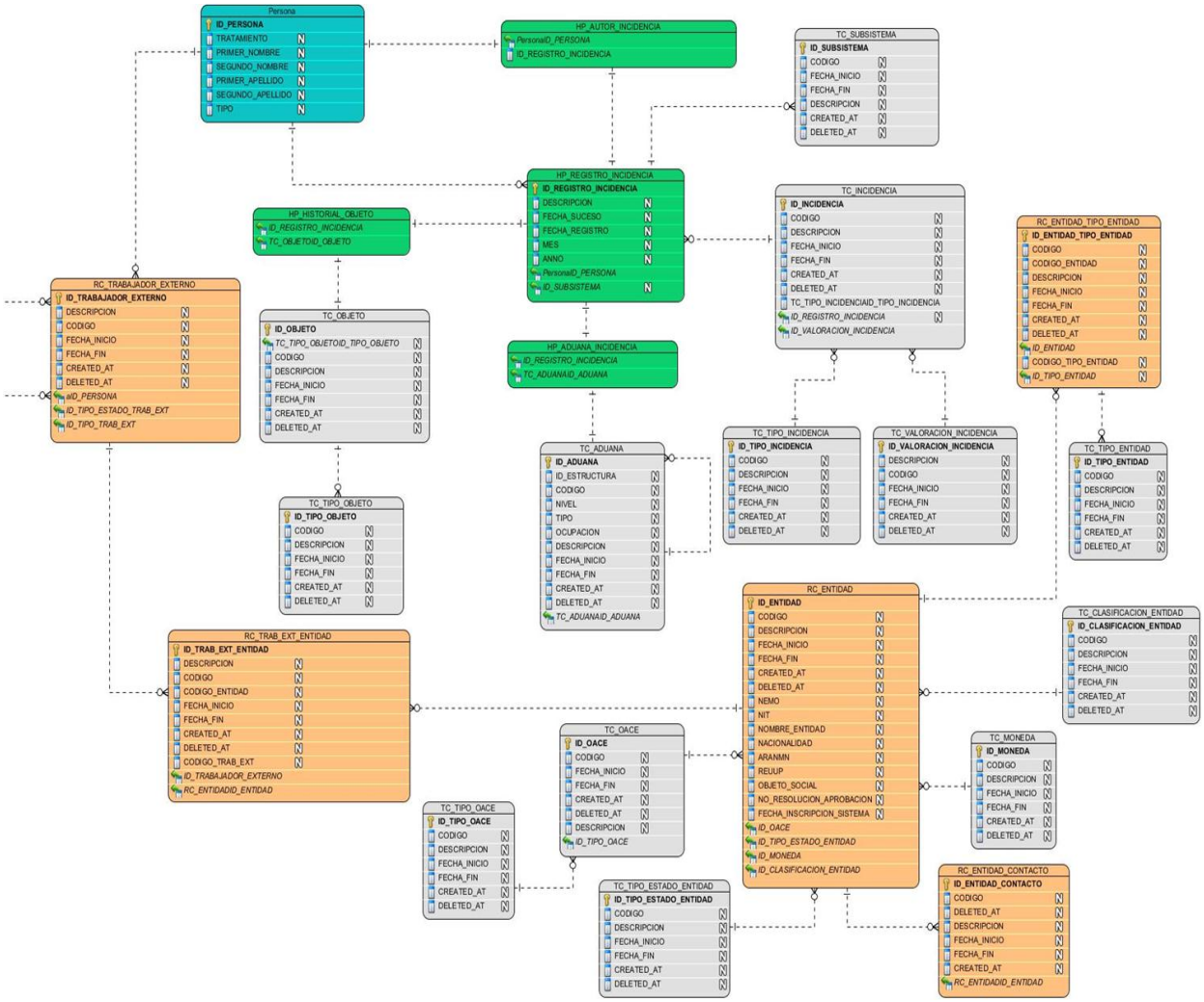


Figura 2.2 Diagrama ER.

2.6.2. Diagrama de clases del diseño

Con la utilización del *framework Symfony* y las librerías Ext JS el diseño se puede desarrollar con una estructura similar para varios requerimientos. La página servidora *Actions.class* se encargará de las peticiones que realiza el controlador frontal; luego estas se redireccionan al Layout. Posteriormente se construye la página cliente que importa todas las interfaces de usuario del subsistema. Estas interfaces están definidas en el fichero de configuración view.yml y se encuentran representadas dentro del paquete Interfaces JS.

A continuación se muestra el diagrama de clases del diseño (**Figura 2.3**) que describe el proceso de obtención del registro de las incidencias de una persona.

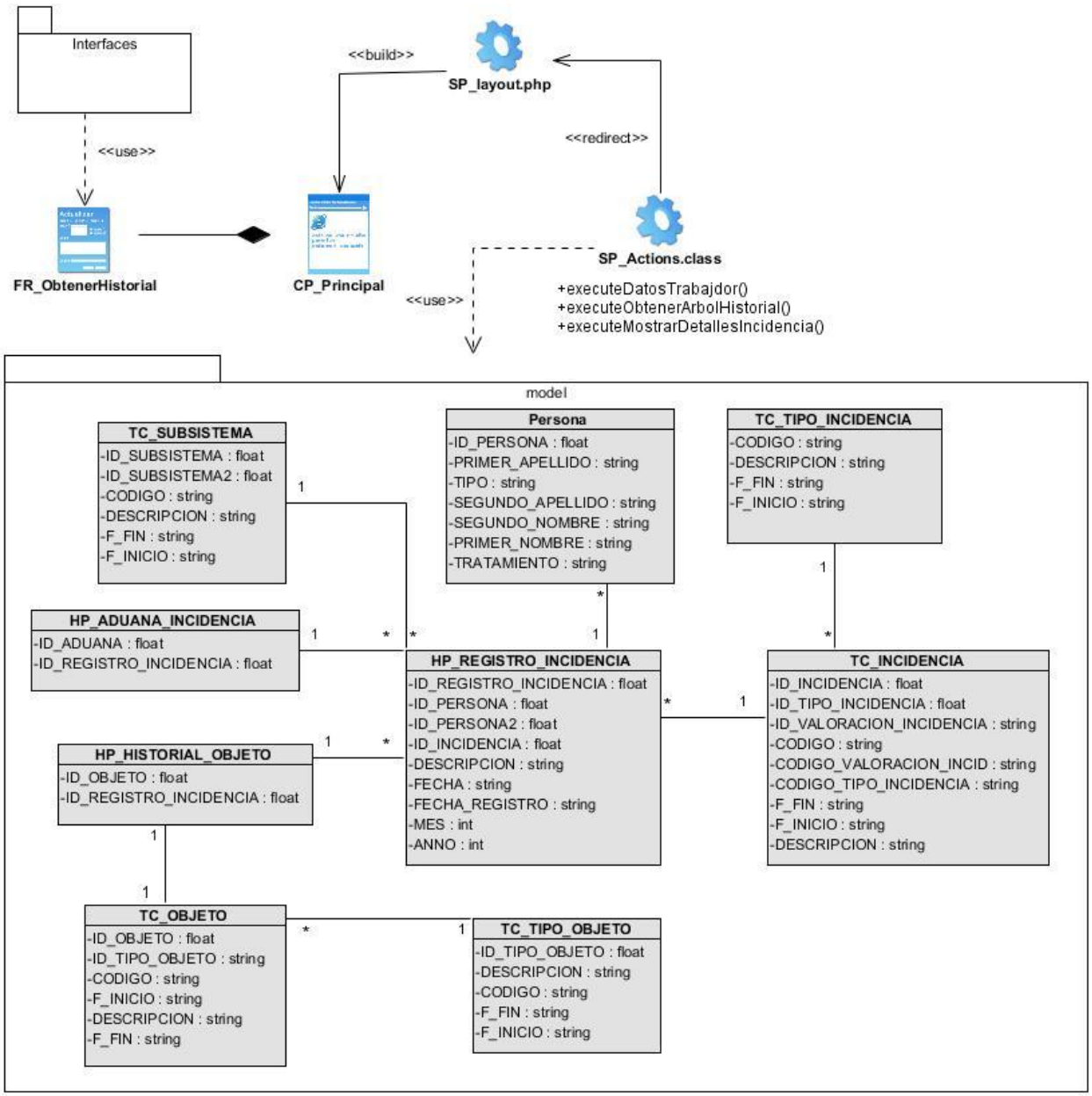


Figura 2.3 Diagrama de clases del diseño con estereotipos web del requisito Obtener historial de Incidencias de un Trabajador Externo.

2.6.3. Diagrama de secuencia

Un diagrama de secuencia muestra una interacción ordenada según la secuencia temporal de eventos, los objetos como líneas de vida a lo largo de la página y con sus interacciones en el tiempo representadas como mensajes. Tiene como objetivo describir el comportamiento dinámico del sistema de información, haciendo énfasis en la secuencia de los mensajes

intercambiados por los objetos. Se pueden colocar etiquetas que pueden ser descripciones de acciones, restricciones de tiempo, entre otros. (29)

A continuación se muestra (**Figura 2.4**) el diagrama de secuencia para el requisito obtener registro de las incidencias de una persona. En el que se describe el proceso que permite al usuario ver el historial o registro de las incidencias de la persona seleccionada; este inicia cuando el usuario (especialista del RCE) selecciona en la interfaz principal de RC la opción 'Gestión de Trabajadores', luego selecciona el trabajador al cual desea obtener su historial de incidencias y da clic en la opción 'Ver Historial'. Esta solicitud hace link a la clase Actions.class que primeramente ejecuta la funcionalidad que permite obtener los datos del trabajador, luego construye la interfaz CP_ConsultarHistorial que muestra inicialmente los datos del trabajador y después el usuario selecciona el año, el mes y el tipo de incidencia que desee obtener más detalles. Estos datos son enviados al Actions.class que a su vez hace llamadas a las clases HpRegistroIncidenciaPeer y TcTipoIncidenciaPeer. Al hacer clic en la incidencia se envía el id de la misma y se ejecuta la funcionalidad que muestra sus detalles.

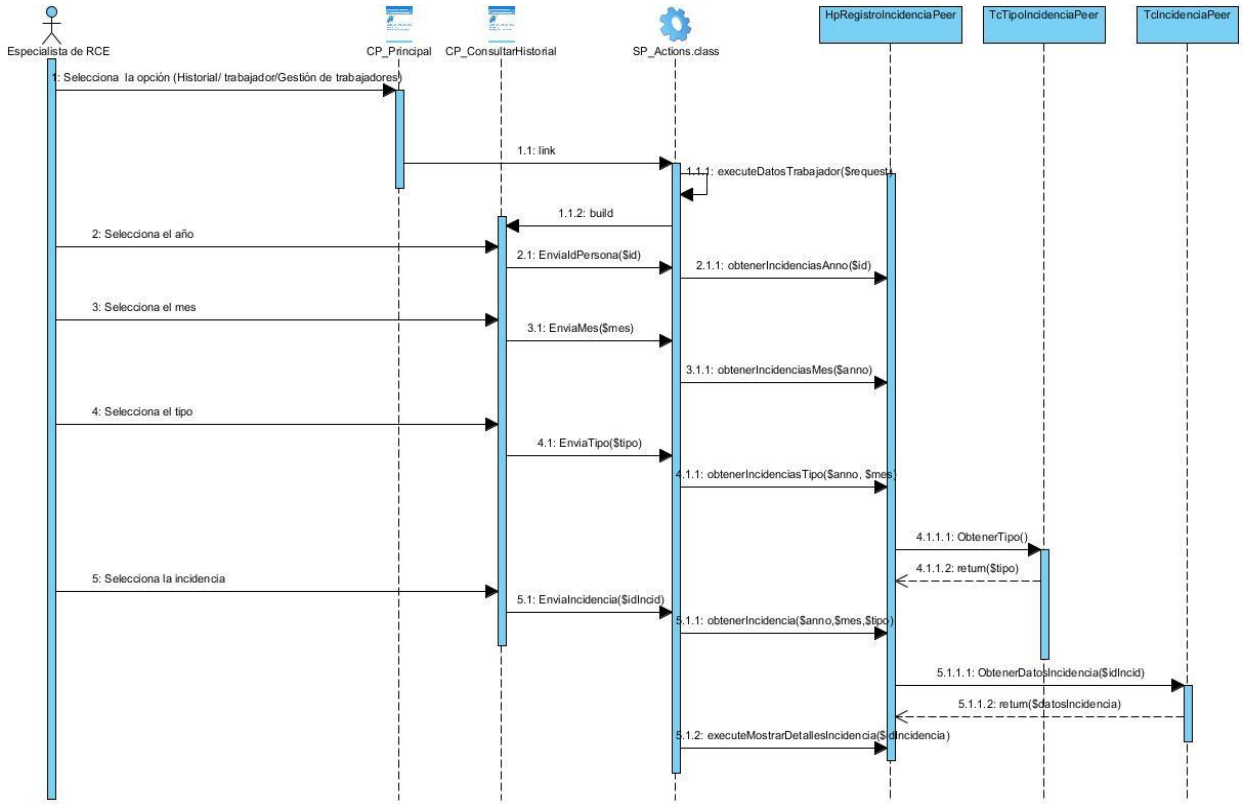


Figura 2.4 Diagrama de secuencia del requisito Obtener historial de Incidencias de un Trabajador Externo.

2.7. Validación del diseño

La clase es la unidad principal de todo sistema orientado a objetos. Esto implica que las medidas y métricas para una clase individual, la jerarquía y las colaboraciones sean sumamente valiosas para un ingeniero de software que tenga que estimar la calidad de un diseño. La métrica es una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado, los objetivos principales de las métricas son: comprender mejor la calidad del producto, estimar la efectividad del proceso y mejorar la calidad del trabajo realizado a nivel del proyecto. (29)

A continuación se aplica una de las métricas de diseño orientado a objeto y se analizan los resultados para determinar la calidad del diseño planteado.

Métrica de Tamaño Operacional de Clase (TOC)

Esta métrica es propuesta por Lorenz y Kidd los cuales dividen las métricas basadas en clases en cuatro categorías: tamaño, herencia, valores internos y valores externos. Las métricas orientadas a tamaños para una clase se centran en cálculos de atributos y de operaciones para una clase individual, y promedian los valores para el sistema en su totalidad. Las métricas basadas en herencia se centran en la forma en que se reutilizan las operaciones en la jerarquía de clases. Las métricas para valores internos de clase examinan la cohesión y asuntos relacionados con el código, y las métricas orientadas a valores externos examinan el acoplamiento y la reutilización. (30)

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:

Atributos	Afectación
Responsabilidad	El aumento del TOC implica el aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	El aumento del TOC implica el aumento de la complejidad de implementación de la clase.
Reutilización	El aumento del TOC implica la disminución del grado de reutilización de la clase.

Tabla 2.1 Afectaciones en el diseño según la métrica TOC.

Para evaluar las métricas son necesarios los valores de los umbrales para los parámetros de calidad. Algunos especialistas plantean umbrales para esta métrica basándose en el promedio de operaciones por clases obtenidos, estos valores fueron los aplicados en el diseño de este sistema. (**Tabla 2.2**).

	Criterio	Categoría	
Responsabilidad	Baja	<=Promedio	<= 8,17
	Media	>Promedio y <= 2*Promedio	>8,17<=16,34
	Alta	>2*Promedio	>16,34
Complejidad Implementación	Baja	<=Promedio	<=8,17
	Media	>Promedio y <= 2*Promedio	>8,17<=16,34
	Alta	>2*Promedio	>16,34
Reutilización	Baja	>2*Promedio	>16,34
	Media	>Promedio y <= 2*Promedio	>8,17<=16,34
	Alta	<=Promedio	<= 8,17

Tabla 2.2 Valores de los umbrales para la métrica TOC.

A continuación se muestran los resultados de la evaluación de la métrica.

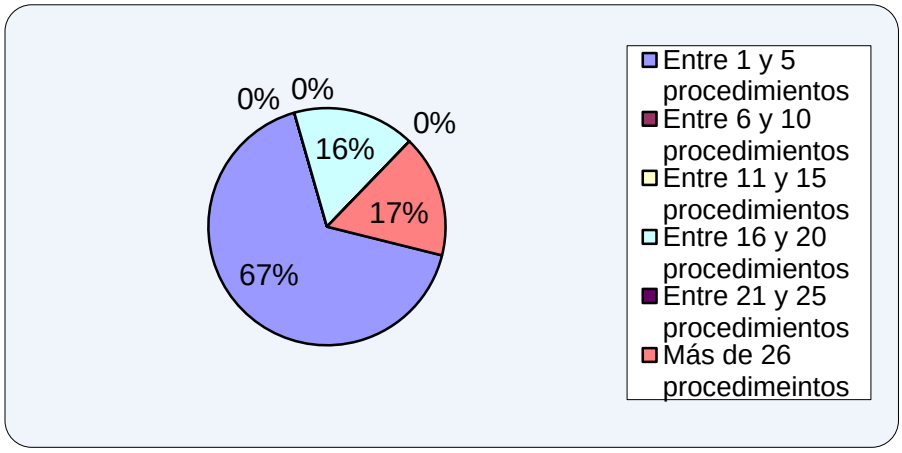


Figura 2.5 Representación del tamaño de las clases del sistema Historial Persona.

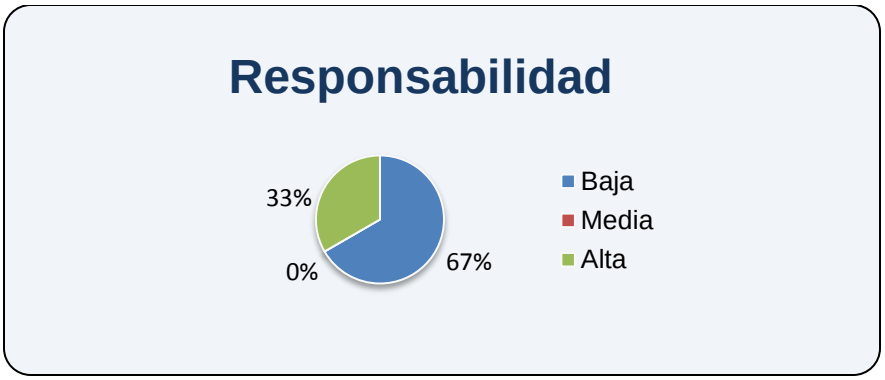


Figura 2.6 Representación de las clases según la responsabilidad.

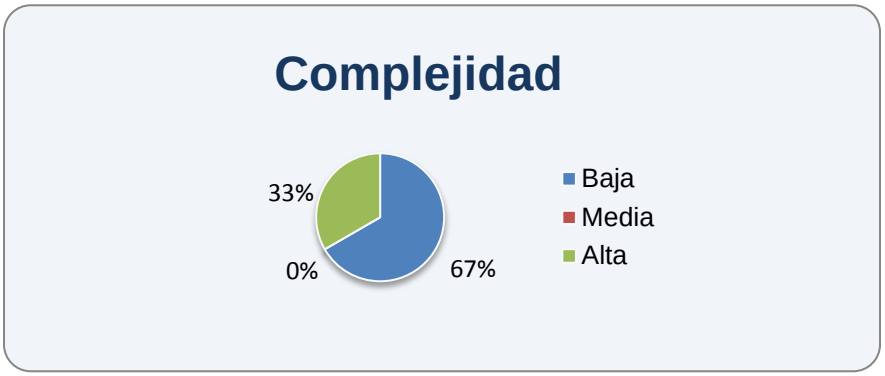


Figura 2.7 Representación de las clases según la complejidad de implementación.

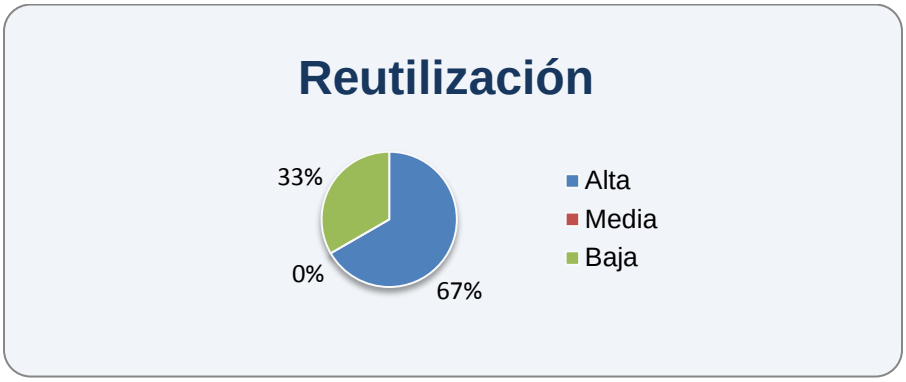


Figura 2.8 Representación de las clases según la reutilización.

Tras un análisis de los resultados arrojados por la evaluación bajo los instrumentos de medición de la métrica TOC, se demuestra que se alcanzaron buenos valores para cada uno de los atributos de calidad evaluados, puesto que, como se puede observar, el 67% de las clases del software contienen un número menor que el promedio de procedimientos por clases, lo cual influye positivamente en el hecho de que predomine una responsabilidad baja de las clases en

un 67%, y hace que carezcan de mucha complejidad, por lo que se tornan reutilizables. Estos valores demuestran que los indicadores de reutilización, complejidad y responsabilidad no se ven afectados.

2.8. Conclusiones parciales

En el capítulo se definieron los requisitos funcionales del sistema y las diferentes técnicas empleadas para la captura y validación de los mismos. Se realizó el diseño de la solución, obteniendo como resultado una serie de artefactos, en los que se exponen varios diagramas tales como: el diagrama de secuencia, el de clases del diseño con estereotipos web y el diseño de la base de datos. Además se presentaron los resultados de su validación, evidenciándose a través de la utilización de la métrica TOC, buenos valores para los atributos de calidad, predominando una baja responsabilidad, complejidad y un alto por ciento de reutilización.

Capítulo 3: Implementación y Validación

3.1. Introducción

En este capítulo se lleva a cabo la implementación del sistema, describiendo los estándares de codificación utilizados para el desarrollo del mismo y se mostrará el código obtenido. Además se realizará la validación del sistema implementado mediante pruebas de software, con el objetivo de medir el grado en que este cumple con los requerimientos planteados por el cliente.

3.2. Diagrama de componentes

Un diagrama de componentes muestra el sistema dividido por componentes, así como la relación entre estos. Un componente es una parte física de un sistema (módulo, base de datos, programa o ejecutable). (29)

En la *Figura 3.1* se muestra el diagrama de componentes obtenido durante el diseño de la solución, el cual se encuentra compuesto en su totalidad por diez componentes.

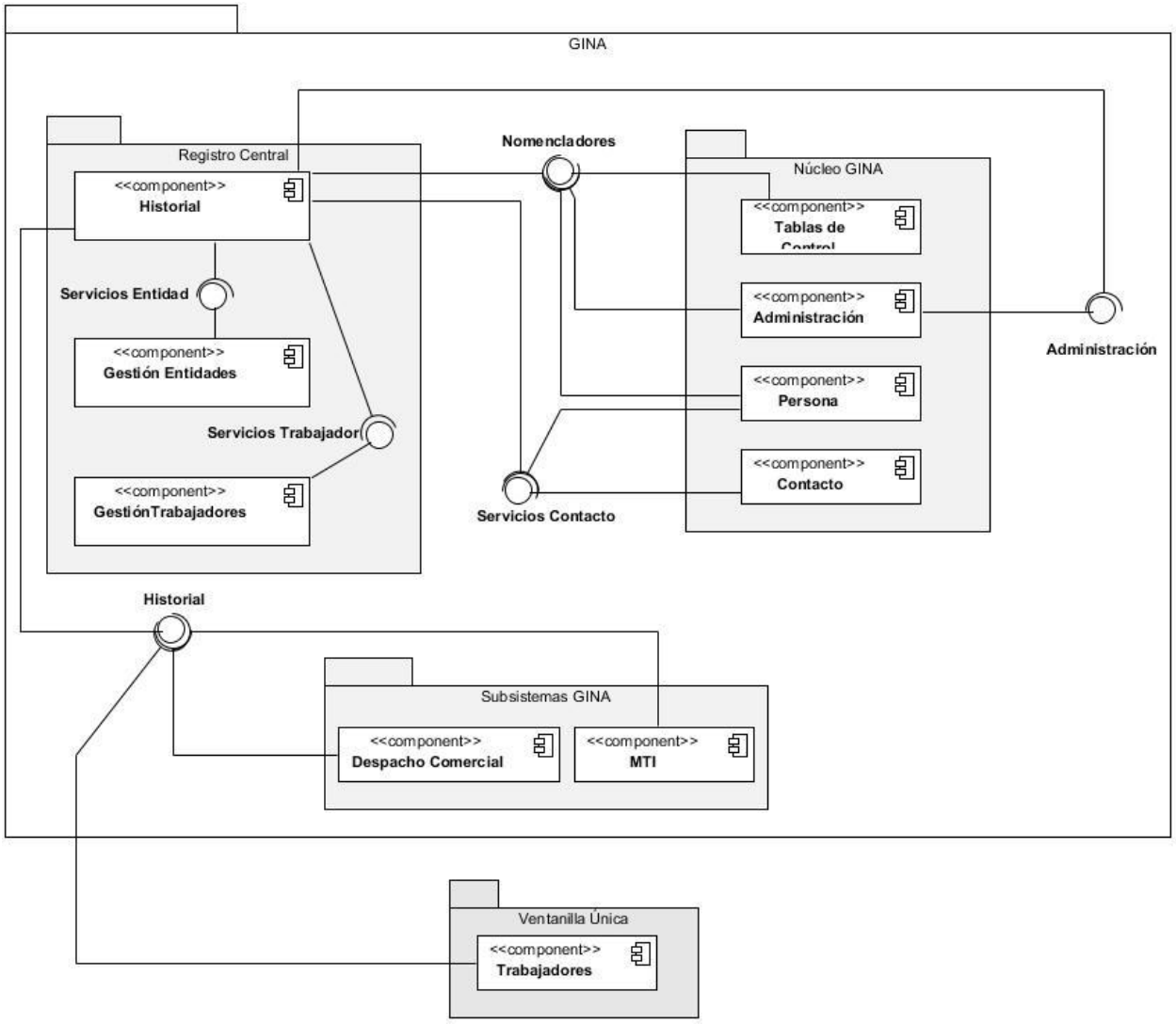


Figura 3.1 Diagrama de Componentes.

El componente **Historial** es el encargado de mantener el registro de las operaciones que realizan los trabajadores y entidades de la AGR, así como los diferentes estados por los que han transitado. Este consume los servicios que prestan los componentes **Gestión Trabajadores** y **Gestión Entidades** para poder mantener un histórico de ambos elementos. Utiliza los componentes **Tablas de Control** para gestionar los nomencladores empleados por el historial y **Administración** para establecer los niveles de acceso. Emplea además el componente **Contacto** para obtener los servicios de contacto a las personas registradas en el sistema, permitiendo la gestión de los mismos así como el componente **Persona** para acceder al registro de los datos de todas las personas naturales dentro del sistema GINA. El historial

brinda servicios a los componentes **Despacho Comercial** y **MTI** del subsistema GINA, además del componente **Trabajadores** de Ventanilla Única.

3.3. Estándares de codificación

En la actualidad existen diferentes estándares para cada uno de los lenguajes, su utilización permite una comunicación fluida y directa entre los programadores de manera que se favorece la reutilización y mantenimiento de los sistemas.

En el Departamento de Soluciones para la Aduana, el estándar de codificación fue definido por la dirección del proyecto en sus inicios, debido a que la adopción de un estándar de codificación sólo es viable si se sigue desde el principio hasta el final del proyecto de software y este se encuentra registrado en el documento "Propuesta de un estándar de codificación". (31)

A continuación se citan algunas de las principales reglas de codificación utilizadas en el desarrollo de la solución:

Acciones (métodos o funcionalidades de la clase *nombreDelModuloAction.class.php*):

- ✓ Dentro de las especificaciones del *framework* está que cada una de estas acciones debe comenzar con la palabra ***execute***.
- ✓ Todos los nombres de acciones deben estar en la nomenclatura "***CamelCase***"[10] comenzando por la palabra ***execute***.
- ✓ Los nombres de las acciones deben especificar, con la menor cantidad de palabras, cual es el objetivo de la acción.

El nombre de las clases debe estar expresado en notación ***UpperCamelCase***[11], además del uso "**Peer**" como sufijo. No se deben utilizar guiones bajos en su nombre "_" y expresar con claridad cuál es el alcance y la responsabilidad de la misma.

Las funciones definidas deben seguir la nomenclatura ***UpperCamelCase***, su nombre debe dejar reflejado claramente cuál es la acción que realiza el mismo. Debe apoyarse en la

[10] **CamelCase**: consiste en escribir frases o palabras compuestas eliminando los espacios intermedios y poniendo en minúscula la primera letra y en mayúscula las demás primeras letras de cada palabra contigua.

[11] **UpperCamelCase**: Consiste en escribir frases o palabras compuestas eliminando los espacios intermedios y poniendo en mayúscula la primera letra de cada palabra incluyendo la primera letra de la frase.

utilización de sufijos que ayuden a identificar el resultado final de la ejecución de un método y de los prefijos para expresar la acción que realiza sobre un elemento determinado.

Los nombres de las variables deben expresar claramente el contenido de la misma, pueden estar referidas en singular o plural y se definen al principio de las estructuras donde son utilizadas. Además de que 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. De esta nomenclatura se exceptúan las variables generadas por el *framework*.

3.4. Tratamientos de errores

Entre los aspectos más importantes a tener en cuenta durante el desarrollo de un software se encuentra el tratamiento de errores, debido a que los usuarios cometen errores a la hora de ejecutar cualquier acción sobre la aplicación. En el sistema propuesto se tratan estos errores de forma tal que las interacciones con la base de datos (inserción, eliminación, modificación y obtención) se realicen de forma correcta.

En el lado del cliente se buscó la forma en que el usuario introduzca la menor cantidad posible de datos, evitando incoherencias e incorrecciones en los mismos. Cuando se realiza una entrada de datos por parte del usuario se implementaron funciones que validaron dicha entrada. Ante la ocurrencia de errores se toma como estrategia notificar al usuario, garantizando que los mensajes sean amigables, precisos y brinden solo la información que necesite saber.

Otro de los aspectos importantes utilizado en el tratamiento de errores es el de los validadores de formularios que provee el *framework Symfony*. Estos formularios son utilizados para las inserciones o actualizaciones, ya que garantizan la limpieza de los datos obtenidos de la capa de la vista.

3.5. Comunicación entre las capas del Modelo-Vista-Controlador

El *framework* Symfony interactúa con las capas del negocio al hacer uso del patrón arquitectónico MVC, logrando un mayor rendimiento, escalabilidad y organización de las diferentes estructuras del sistema. A continuación se muestra un ejemplo de comunicación entre capas para ayudar a comprender el funcionamiento de la solución.

3.5.1. Ejemplo de comunicación entre capas

En el presente epígrafe se explica el funcionamiento de la comunicación entre capas del sistema durante la ejecución del requisito funcional registro de las incidencias de un trabajador.

Este se inicia cuando el usuario a cargo del sistema desea consultar el historial de un trabajador de los que se encuentran registrados en el subsistema de RC, en la interfaz de gestión de trabajadores.

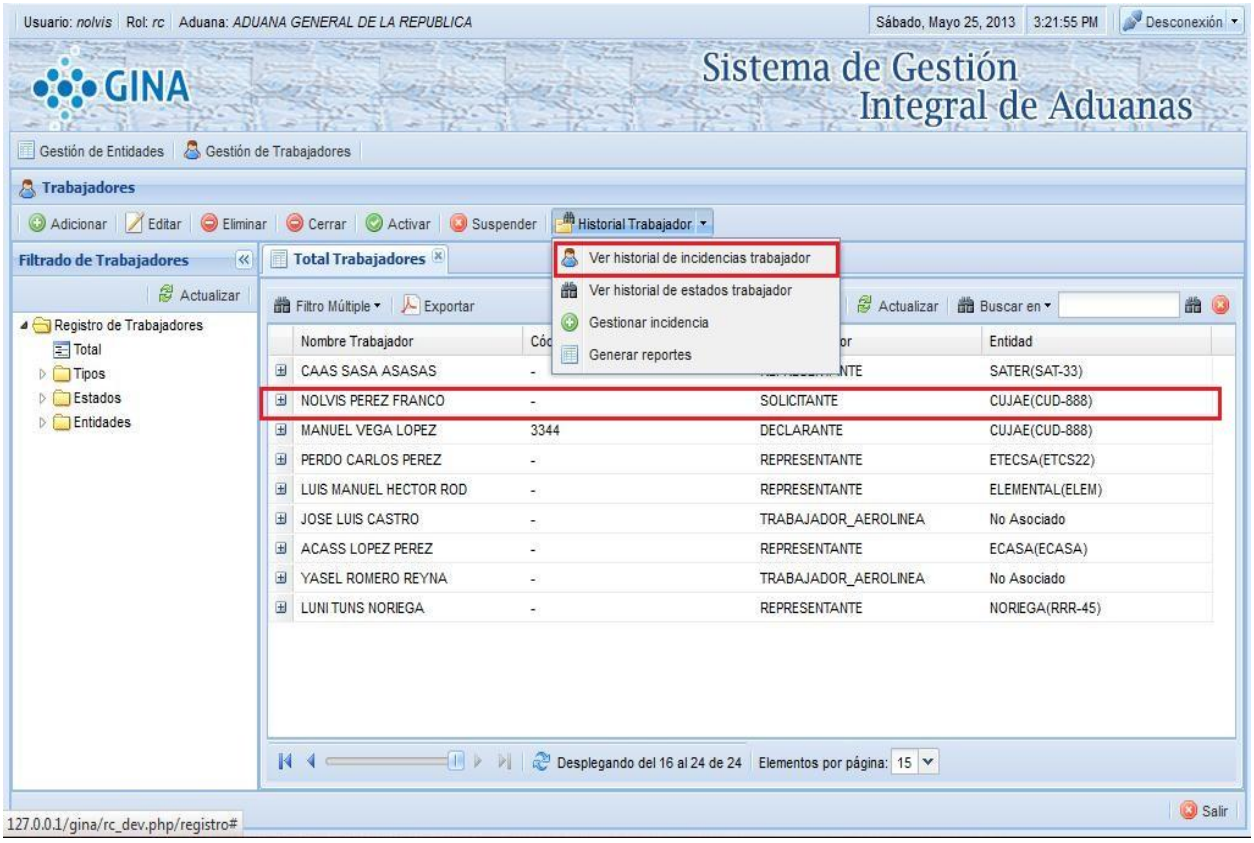


Figura 3.2 Interfaz de gestión de trabajadores.

Para acceder al historial de uno de los trabajadores del listado anterior primeramente se selecciona al trabajador y posteriormente se ejecuta el botón señalado en rojo ***Ver historial de incidencias trabajador*** cuyo código se muestra a continuación.

```
// web/js/rc/trabajadores /mainTrabajadores.js

{
    text: 'Historial',
     xtype: 'button',
     tooltip: 'Consultar el historial de un trabajador',
```

```
        iconCls: 'iconFolderFind',
        handler: function() {

if(Ext.getCmp(Ext.getCmp('mainTrabajadores').idTabPanel).getActiveTab().getSelectionModel().h
asSelection()) {
            var record =
Ext.getCmp(Ext.getCmp('mainTrabajadores').idTabPanel).getActiveTab().getSelectionModel().get
Selected();
            var idPersona = record.data.idPersona;

            var win = new ConsultarHistorial({
              iconCls: 'iconGrid',
              title: 'Consultar Historial',
              idPersona: idPersona,
              modoEdit: false
            });
            win.show();
          } else {
            Ext.MessageBox.show({
              title: 'Informaci&oacute;n',
              msg: "Para consultar el historial de un trabajador primeramente debe seleccionarlo.",
              buttons: Ext.MessageBox.OK,
              icon: Ext.MessageBox.INFO,
              minWidth: screen.width * 30 / 100
            });
          }
        }
      }
```

La ejecución del código anterior envía el identificador de la persona a la cual se le solicitó ver el historial hacia la clase de JavaScript ***consultarHistorial.js*** y esta a su vez ejecuta dos funcionalidades implementadas en la clase controladora ***actions.class.php***: ***execute DatosTrabajador()***, encargada de mostrar los datos personales del trabajador seleccionado y ***executeArbolHistorialPersona()*** encargada de la obtención de su historial. (**Figura 3.3**)

El fragmento de código que se muestra a continuación retorna los datos personales mencionados anteriormente.

```
//apps/rc/modules/historial/actions/actions.class.php


try {
    $idPersona = $this->getRequestParameter('idPersona');
    $criteria = new Criteria();
    $criteria->add(PersonaPeer::ID_PERSONA, $idPersona);
    $persona = PersonaPeer::doSelect($criteria);
    $syscomponent = SysComponentsLocator::getInstance();
    $personaComponent = $syscomponent->getComponent('persona');
    $params['idPersona'] = $idPersona;
```

```
    $p = $personaComponent->executeService(new sfEvent(null, 'persona.buscarPorIdPersona',
array("idPersona" => $params['idPersona'])));
    $datosPersonales = $p->getDatosPersonales();

    $pais = $datosPersonales->getIdPais();
    $syscomponent = SysComponentsLocator::getInstance();
    $componenteContacto = $syscomponent->getComponent('tc');
    $ct = new Criteria();
    $ct->add(TcPaisPeer::ID_PAIS, $pais);
    $param = array(
      'criteria' => $ct,
      'nombreTc' => TcPaisPeer::TABLE_NAME,
      'estado' => SisTcTablaPeer::VIGENTE
    );
    $pais = $componenteContacto->executeService(new sfEvent(null, 'tc.obtenerDadoCriteria',
$param));
    $nacionalidad = $pais[0]->getDescripcion();
    $trabajador = RcTrabajadorExternoPeer::obtenerTrabajadorDadoIdPersona($idPersona);
    $codigo = $trabajador['codigo'];

    If ($persona) {
      foreach ($persona as $dato) {
        $datosPersona = array('nombre' => $dato->getPrimerNombre() . " " . $dato-
getSegundoNombre() . " " . $dato->getPrimerApellido(),
          'tipo' => $dato->GetTipo(),
          'nacionalidad' => $nacionalidad,
          'tipo' => $trabajador['tipoTrabajador'],
          'codigo' => $trabajador['codigo']
        );
      }
      return $this->renderText(json_encode($datosPersona));
    } else {
      throw new Exception("El trabajador referenciado no existe en BD.");
    }
  } catch (Exception $ex) {
    $msg = 'El trabajador referenciado no existe en BD.';
    return $this->renderText("{\"success\": false, \"msg\": '" . $msg . "'}");
  }
 }
```

Figura 3.3 Interfaz principal del requisito Obtener historial de Incidencias de un Trabajador Externo.

El fragmento de código que se presenta a continuación muestra cómo se procede a la hora de realizar el llenado del árbol de exploración del historial situado a la izquierda de la pantalla que se mostró en la *Figura 3.3.*

```
//apps/rc/modules/historial/actions/actions.class.php

try {
    $nodo = $request->getParameter('node');
    $idPersona = $request->getParameter('idPersona');
    if (isset($nodo) && isset($idPersona)) {
      $datosNodo = explode("/", $nodo);
    }
    else
      throw new Exception("Error durante la recepción de los datos");

    $switchChoice = explode("_", $datosNodo[0]);
    $result = array();

    switch ($switchChoice[0]) {
      case 'ROOT':
        $result = HpRegistroIncidenciaPeer::obtenerIncidenciasAnno($idPersona);
        break;

      case 'MES':
        $anno = $datosNodo[1];
        $result = HpRegistroIncidenciaPeer::obtenerIncidenciasMes($idPersona, $anno);
```

```
        break;

      case 'TIPO':
        $mes = explode("_", $datosNodo[0]);
        $anno = explode("_", $datosNodo[1]);
        $result = HpRegistroIncidenciaPeer::obtenerIncidenciasTipo($idPersona, $anno[1], $mes[1]);
        break;

      case 'INCIDENCIA':
        $mes = explode("_", $datosNodo[1]);
        $tipo = explode("_", $datosNodo[0]);
        $anno = explode("_", $datosNodo[2]);
        $result = HpRegistroIncidenciaPeer::obtenerIncidencias($idPersona, $anno[1], $mes[1], $tipo[1]);
        break;
    }
        return $this->renderText(json_encode($result));
  } catch (Exception $ex) {
    $msg = 'El trabajador no posee incidencias.';
    return $this->renderText("{\"success\": false, \"msg\": '" . $ex->getMessage() . "'}");
  }
```

Como se puede apreciar, la funcionalidad está organizada mediante una sentencia ***switch()*** compuesta por cuatro bloques. Cada uno de ellos actualiza los atributos que va a retornar de forma escalonada hasta llegar a obtener las incidencias del trabajador, en ese caso el valor *leaf*, el cual indica si se está o no al final del árbol, actualiza su valor a *true* y se realiza la llamada a la funcionalidad *executeMostrarDetallesIncidencia ()*.*(Figura 3.4)*

```
//apps/rc/modules/historial/actions/actions.class.php

/*
* Accion que retorna los detalles de una incidencia dado el idRegistroIncidencia
* @param sfWebRequest $request
* @return renderPartial
*/
public function executeMostrarDetallesIncidencia(sfWebRequest $request) {
  try {
    $idRegistroIncidencia = $this->getRequestParameter('id');
    $result = HpRegistroIncidenciaPeer::ObtenerRegistroIncidenciaPorId($idRegistroIncidencia);

    return $this->renderPartial('incidenciasPartial', array(
        'idPersona' => $result['idPersona'],
        'idIncidencia' => $result['idIncidencia'],
        'fechaSuceso' => $result['fechaSuceso'],
        'fechaRegistro' => $result['fechaRegistro'],
        'tipo' => $result['tipo'],
        'descripcion' => $result['descripcion']
    ));
```

```
    } catch (Exception $ex) {
      $msg = 'El trabajador no posee incidencias.';
      return $this->renderText("{\"success\": false, \"msg\": '" . $msg . "'}");
    }
  }
```

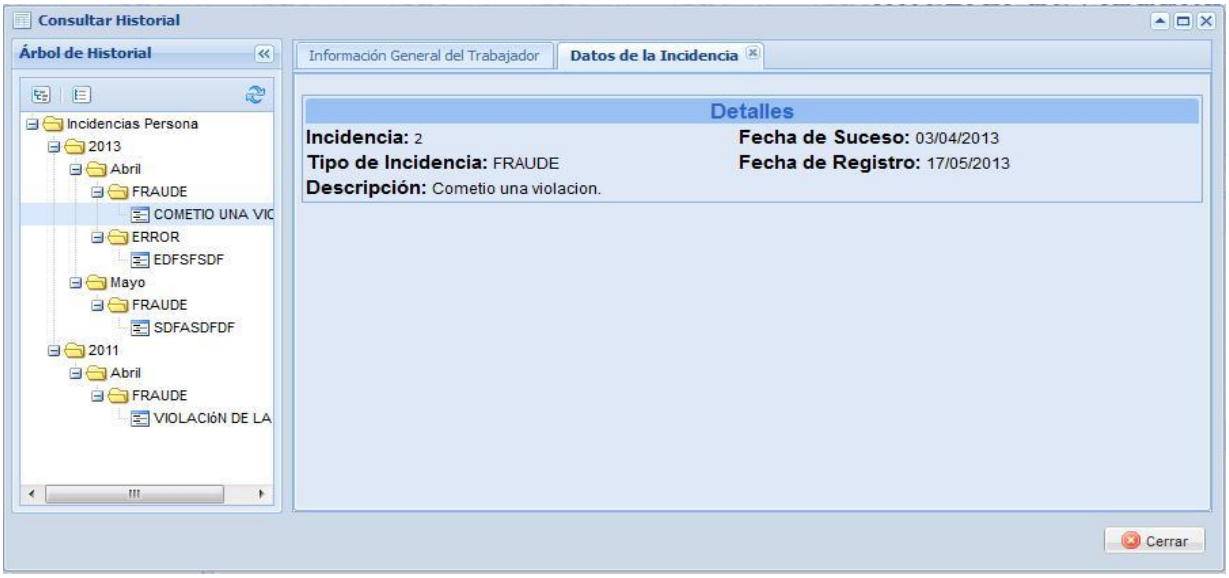


Figura 3.4 Pantalla con la descripción detallada del historial de incidencias del trabajador.

3.6. Pruebas de validación

Al desarrollar sistemas informáticos se corre un alto riesgo de que se produzcan errores, los cuales pueden ocurrir desde el comienzo del proceso, ya sea en la definición de los objetivos, el diseño, la implementación o en otras fases.

El proceso de pruebas tiene como objetivos fundamentales planificar las pruebas necesarias en cada iteración, incluyendo las pruebas de integración y las pruebas de sistema. Diseñar e implementar las pruebas creando los casos de pruebas que especifican que probar. Además de realizar las diferentes pruebas y manejar los resultados de cada una sistemáticamente de forma que las no conformidades puedan ser corregidas. (32)

Para realizar la validación de la solución propuesta se aplicarán las pruebas unitarias y funcionales como técnicas de caja negra para comprobar la validez en las respuestas del programa ante las acciones del usuario y la calidad de las salidas en dependencia de las entradas.

3.6.1. Pruebas unitarias

Las pruebas unitarias del *framework Symfony* son archivos PHP normales cuyo nombre termina en *Test.php* que se encuentran en el directorio *test/unit/* de la aplicación y se realizan haciendo uso del *framework Lime* que viene incluido en *Symfony*. Su sintaxis es sencilla y fácil de implementar, es válido aclarar que las mismas son hechas por los propios desarrolladores.

Para ejecutar el conjunto de pruebas, se utiliza la tarea *test:unit* desde la línea de comandos. El resultado de esta tarea en la línea de comandos permite localizar fácilmente las pruebas que han fallado y las que se han ejecutado correctamente. Además de que a una misma funcionalidad se le pueden hacer varias pruebas.

Para realizar la validación se ejecutaron un total de 4 pruebas unitarias a diferentes funcionalidades por ser las más significativas para el funcionamiento de la solución:

Funcionalidades	Descripción
registrarIncidencia	Devuelve un objeto incidencia en caso de ser registrada correctamente.
buscarIncidenciasTrabajador	Devuelve un arreglo de incidencias dado un ***idPersona***.
obtenerEstadosTrabajador	Devuelve un arreglo de estados dado un ***idTrabajador***.
datosEntidadHistorial	Devuelve un arreglo con los datos de la entidad dado un ***idEntidad***.

Tabla 3.1 Funcionalidades tomadas como muestra para la aplicación de las pruebas unitarias.

A continuación se muestran las pruebas unitarias del *framework* Symfony aplicadas a la funcionalidad **registrarIncidencia** que debe registrar en la base de datos la incidencia con los datos que introdujo el usuario.

No.	Descripción	Resultado Esperado	Resultado Obtenido
1	Tipo de resultado	Objeto Registro Incidencia	Objeto Registro Incidencia

2	Cadena	Cadena	Cadena
3	Cadena	Cadena	Cadena
4	Cadena	Cadena	Cadena

Tabla 3.2 Resultados esperados y obtenidos al aplicar las pruebas unitarias del *framework* Symfony.

Las pruebas unitarias del framework Symfony para el caso anterior se definieron de la siguiente forma:

```
$datosIncidencia = array('datosRegIncidencia' => array(
      'descripcion' => 'Violacion de la ley',
      'fecha' => '2013-05-08',
      'fechaRegistro' => '2013-05-17 16:05:46',
      'anno' => '2013',
      'mes' => '05',
      'idPersona' => 82129,
      'idIncidencia' => 2,
      'idRegistroIncidencia' => 62 ),
    'idAduana' => 11,
    'idObjeto' => '' );
$incidencia = HpRegistroIncidenciaPeer::registrarIncidencia($datosIncidencia);
$t->isa_ok($incidencia, 'HpRegistroIncidencia', "Resultado correcto -- Objeto Registro Incidencia
--");
if ($incidencia) {
  $t->isa_ok($incidencia->getAnno(), 'string', "Resultado correcto -- Cadena --");
  $t->isa_ok($incidencia->getMes(), 'string', "Resultado correcto -- Cadena --");
  $t->isa_ok($incidencia->getDescripcion(), 'string', "Resultado correcto -- Cadena --");
}
```

El ejecutar estas pruebas el resultado arrojado es el siguiente:

```
\Desarrollo\www\ GINA\>php Symfony test:unit Hp
1..4
#----- Iniciando el proceso de pruebas ------
#-----------------------------------------------------------------------
# Prueba Registro de Incidencias
ok 1-- Resultado correcto -- Objeto Registro Incidencia –
ok 2---Resultado correcto -- Cadena –
ok 3---Resultado correcto -- Cadena –
ok 4---Resultado correcto -- Cadena –
#-----------------------------------------------------------------------
Looks like everything went fine.
```

El resultado anterior demuestra que las 4 pruebas realizadas se ejecutaron correctamente.

Las pruebas unitarias arrojaron los resultados esperados. Es válido destacar que cada línea prueba una única cosa, de esa forma es posible detectar mayor cantidad de errores, lo que

posibilita que cada funcionalidad pueda ser evaluada con un número elevado de pruebas así como con diferentes datos.

De la misma forma se le realizaron las pruebas al resto de las funcionalidades obteniéndose, en la mayoría de los casos, los resultados propuestos.

3.6.2 Pruebas funcionales

Las pruebas funcionales fueron ejecutadas por calidad interna del centro CEIGE. El objetivo fundamental de estas pruebas es validar si el comportamiento observado del software cumple o no con sus especificaciones definidas. La prueba funcional toma el punto de vista del usuario. Las funciones son probadas ingresando las entradas y examinando las salidas. La estructura interna del programa raramente es considerada.

A continuación se muestra el diseño del caso de prueba correspondiente el requisito funcional: Registrar Incidencia.

Escenario	Descripción	Flujo central
EC 1.1 Registrar Incidencia	Muestra la interfaz “Registrar Incidencia”, con los campos para que el usuario introduzca los datos de la incidencia y luego los guarda.	1. Selecciona la opción “**Gestión de trabajadores**”. 2. Selecciona el trabajador que se le va a registrar la incidencia. 3. Da clic en la opción “**Historial Trabajador**” del menú y selecciona “**Gestionar Incidencia**”. 4. Presionar el botón **“Registrar”** y se muestra su interfaz. 5. Se llenan los campos de la pantalla y da clic en la opción “**Aceptar**”. 6. Valida los datos introducidos. 7. Se guarda la incidencia registrada.

EC 1.2 Datos incorrectos	Se realiza este escenario en el caso cuando los datos son incorrectos	1. Selecciona la opción "**Gestión de trabajadores**". 2. Selecciona el trabajador que se le va a registrar la incidencia. 3. Da clic en la opción "**Historial Trabajador**" del menú y selecciona "**Gestionar Incidencia**". 4. Presionar el botón "**Registrar**" y se muestra su interfaz. 5. Se llenan los campos de la pantalla y da clic en la opción "**Aceptar**". 6. Valida los datos introducidos. 7. Muestra el mensaje: "Existen campos con datos Incorrectos"

Tabla 3.3 Diseño de casos de prueba para el RF Registrar Incidencia.

Los escenarios de pruebas descritos anteriormente fueron probados con los datos que se muestran en la tabla 3.3.

Escenario	Fecha Suceso	Incidencia	Descripción	Aduana	Objeto	Respuesta del sistema	Respuesta de la Prueba
EC 1.1 Registrar Incidencia	V 2/10/2012	V Incidencia1	N/A vacío	N/P	N/P	Verifica los datos y guarda la incidencia con los datos introducidos, Muestra el mensaje" La incidencia ha sido registrada correctamente "	Verifica los datos y guarda la incidencia con los datos introducidos, muestra el mensaje "La incidencia ha sido registrada correctamente"
	V 2/10/2012	V Incidencia1	V Ha cometido una Violación……	V AEPH	V Ley General de Recursos Humanos		No guarda la incidencia, porque en el campo descripción no estaba validada con un límite de caracteres.

EC 1.2 Datos incorrectos	I : 12345	I: 36	N/A vacío	N/P	N/P	Muestra el mensaje: "Existen campos con datos Incorrectos"	Muestra el mensaje: "Existen campos con datos Incorrectos"

Tabla 3.4 Juego de datos el RF Registrar Incidencia.

Como se pudo apreciar, en la segunda prueba que se le aplicó al escenario de prueba 1.1 se detectó una no conformidad. La cual consiste en que al usuario introducir los caracteres del campo descripción el sistema no valida la funcionalidad al no tener limitada la cantidad de caracteres a introducir.

3.6.3. Resumen del proceso de validación

Las pruebas de Caja Negra fueron aplicadas al resto de los RF de la misma forma que se describió en el epígrafe anterior. Los resultados por escenarios para cada una de las funcionalidades se describen a continuación:

Funcionalidades:

- ✓ Eliminar incidencia a un trabajador: 1 escenario.
- ✓ Modificar incidencia a un trabajador: 2 escenarios, 1 no conformidad.
- ✓ Registrar Incidencia a un trabajador: 2 escenarios, 1 no conformidad.
- ✓ Obtener historial de estados de un Trabajador Externo: 1 escenario.
- ✓ Obtener historial de Incidencias de un Trabajador Externo: 1 escenario.
- ✓ Obtener historial de los estados de una entidad: 1 escenario.
- ✓ Generar Reportes: 7 escenarios.

De forma general, las funcionalidades fueron descompuestas en 15 escenarios, detectándose 2 no conformidades para un 13 por ciento en la primera iteración y un 87 por ciento libre de no conformidades. *(Figura 3.5)*

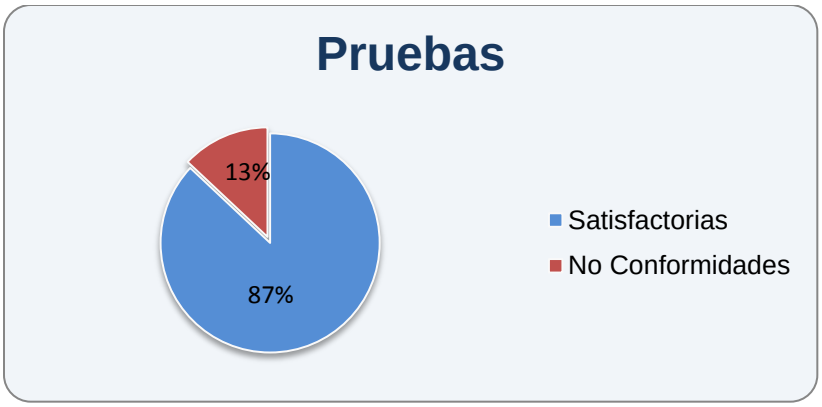


Figura 3.5 Resultados de las pruebas de validación.

Las no conformidades detectadas durante la primera iteración fueron corregidas completamente en la regresión, lo que posibilitó que en la segunda iteración los escenarios arrojaran resultados satisfactorios en todos los casos.

3.7. Aportes de la solución y beneficios esperados

Con el desarrollo de la aplicación propuesta se introdujo una herramienta para la gestión de historiales en la AGR nunca antes implementada, contribuyendo de esta forma a la independencia tecnológica del país y al proceso de informatización de la sociedad cubana.

La solución garantiza el cumplimiento de todas las funcionalidades necesarias para una correcta ejecución del flujo de trabajo en el RCE de la AGR. Al ser un módulo del sistema GINA permite la interacción con los subsistemas, y puede integrarse a los subsistemas que sean implementados más adelante.

El historial se encontrará almacenado en formato digital y ejecutado sobre una tecnología web, posibilitando el acceso a este desde varios puntos del territorio nacional. Esta proveerá reportes que garanticen la toma de decisiones oportuna y rapidez en el procesamiento de informes, los cuales servirán para un mejor análisis de las diferentes situaciones que pueden surgir durante todo proceso.

3.8. Conclusiones parciales

La implementación y calidad del módulo fueron las premisas fundamentales en el desarrollo de este capítulo. En ese sentido se llevó a cabo toda la implementación del sistema donde se evidenciaron las facilidades del *framework* Symfony para potenciar el desarrollo de aplicaciones web dinámicas de gestión. Se obtuvieron los casos de prueba que ayudaron a comprobar el correcto funcionamiento del producto desarrollado aplicando finalmente pruebas unitarias y de funcionalidad. En general, los resultados obtenidos fueron favorables, desde el punto de vista funcional todos los requisitos realizan las funcionalidades requeridas y responden a las necesidades del cliente. Finalmente tras aplicar las pruebas unitarias al sistema quedó demostrada la robustez del sistema implementado.

Conclusiones generales

Al finalizar la presente investigación se lograron cumplir los objetivos planteados de manera satisfactoria, obteniendo como resultado el módulo para manejar el historial de las entidades y los trabajadores externos que pertenecen al registro central de la AGR, que permitirá la centralización del historial de entidades dentro del sistema GINA y el registro de las operaciones de los trabajadores, posibilitando realizar consultas al histórico de ambos.

Con la elaboración del marco teórico de la investigación a partir del estudio del estado del arte se evidenció la carencia de una solución informática capaz de responder a las necesidades y requerimientos de la AGR.

Al realizar el diseño de la solución se obtuvieron las especificaciones del sistema, permitiendo describir y modelar lo que el sistema debe hacer para satisfacer los requerimientos identificados durante el análisis.

Se llevó a cabo la implementación del sistema obteniendo el código fuente de los diseños especificados en el modelo de diseño.

Las pruebas realizadas al sistema implementado demostraron el cumplimiento de diferentes atributos de calidad, lo que evidencia que el sistema cumple satisfactoriamente con los requisitos definidos por los clientes.

Recomendaciones

A lo largo del desarrollo de los objetivos propuestos en esta investigación surgieron nuevas ideas que serían recomendables tener en cuenta:

- ✓ Estudiar nuevos mecanismos que pudieran ser factibles para almacenar y consultar el historial de las operaciones de los trabajadores y entidades en la AGR de Cuba.
- ✓ 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. **AGR.** Sitio de la Aduana cubana. [En línea] [Citado el: 9 de Noviembre de 2012.] http://www.aduana.co.cu/.

2. **Real Academia Española.** *Diccionario.* [En línea] [Citado el: 11 de Diciembre de 2012.] http://buscon.rae.es/draeI/SrvltGUIBusUsual?TIPO_HTML=2&TIPO_BUS=3&LEMA=historial.

3. **Información., Centro Nacional de Tecnologías de.** Gobierno en Línea. [En línea] [Citado el: 11 de Diciembre de 2012.] http://www.gobiernoenlinea.ve/cartelera/Sidunea.html.

4. **Arellano, Ana Alejandra Febres.** *Conociendo al SIDUNEA.* [En línea] [Citado el: 11 de Diciembre de 2012.] http://www.gestiopolis.com/canales2/economia/sidunea.htm..

5. **Korea Customs Service.** *Customs Automation.* [En línea] [Citado el: 13 de Diciembre de 2012.] http://english.customs.go.kr/.

6. **Billy, Carlos Reynoso.** *Introducción a la Arquitectura de Software.* Buenos Aires : Universidad de Buenos Aires, 2004. Vol. I..

7. **Clements, Paul.** *Software Architecture.* Estados Unidos : Software Engineering Institute, 2002. ISBN: 15213-3890.

8. **Shaw, David Garlan and Mary.** *Introducción a la Arquitectura de Software.* New Jersey : World Scientific Publishing Company, 1993.

9. **Alexander, Christopher.** *A Pattern Language.* 1997. ISBN 0-19-501919-9.

10. **Larman, Craig.** *Applying UML and Patterns.* 1997. ISBN 0-13-748880-7.

11. **Hernández, Pedro Veloso.** *Uso de Patrones de Arquitectura.* 2009.

12. **Pantoja, Ernesto Bascón.** *El patrón de diseño modelo -vista - controlador (MVC) y su implementación el Java Swing.* 2004.

13. **Patrones de diseño.** [En línea] Félix Prieto. [Citado el: 17 de enero de 2013.] http://www.infor.uva.es/~felix/datos/priii/tr_patrones-2x4.pdf.

14. **Fowler, Martin.** *Patterns of Enterprise Application Architecture.* ISBM: 0-32112-742-0.

15. **Centro de Informatización de Gestión de Entidades.** *MODELO DE DESARROLLO DE SOFTWARE.* 2012.

16. **Cornejo, José Enrique González.** El Lenguaje de Modelado Unificado (UML). [En línea] [Citado el: 13 de Enero de 2013.] http://www.docirs.cl/uml.htm.

17. **Visual Paradigm International.** UML, BPMN and Database Tool for Software Development. [En línea] [Citado el: 13 de Enero de 2012.] http://www.visual-paradigm.com.

Referencias Bibliográficas

18**. Introducción a XHTML**. [En línea] [Citado el: 15 de Enero de 2013.] http://www.librosweb.es/xhtml/capitulo_1.html.

19. **Pérez, Javier Eguíluz.** *Introducción a XHTML.* 2007.

20. **-. *Introducción a CSS***. 2007.

21. **Madrid, Universidad Carlos III de.** *JavaScript.* [En línea] [Citado el: 14 de Enero de 2013.] http://perso.wanadoo.es/javascript_12.

22. **Veracruz, Instituto Tecnológico de.** Lenguajes de programación del lado Servidor. [En línea] [Citado el: 13 de Enero de 2013.] http://www.prograweb.com.mx/pweb/0203ladoServidor.html.

23. **Flores, Antonio.** *Introducción al Lenguaje de Programación PHP.* Madrid : s.n, 2008.

24**. Frameworks**. [En línea] [Citado el: 13 de Enero de 2013.] http://www.clickmatica.com/tag/frameworks/.

25. **Bullock, Christian.** Ext JS. Comunidad en Español. [En línea] [Citado el: 15 de Enero de 2013.] http://extjses.com.

26. **Zaninotto, Francois Potencier and Fabien.** *Symfony 1.2, la guía definitiva.* 2008.

27. **CAVSI.com.** ¿Qué es un Sistema Gestor de Bases de Datos o SGBD? [En línea] [Citado el: 16 de Enero de 2013.] http://www.cavsi.com/preguntasrespuestas/que-es-un-sistema-gestor-de-bases-de-datos-o-sgbd/.

28. **Martínez, Ing. Jenni Manso.** *Procedimiento para la Ingeniería de Requisitos en el Departamento de Desarrollo de Soluciones para la Aduana del CEIGE.* La Habana : Universidad de las Ciencias Informáticas, 2010.

29. **Sommerville, Ian.** *Ingeniería del Software (séptima edición).* Madrid (España) : Pearson Educación, 2005. 84-7829-074-5.

30. **Lorenz, M. y Kidd, J.** *Object-Oriented Software Metrics.* 1994.

31. **CEIGE, Dpto Aduana.** *Propuesta de un Estándar de Codificación.* 2013.

32. **Addison Wesley, James Jacobson, Ivar, Booch, Grady y Rumbaugh.** *El Proceso Unificado de Desarrollo de Software.* 1999. ISBN: 0-201-57169-2.

Bibliografía Consultada

Ferré Grau, Xavier y Sánchez Segura, María Isabel. 2008. Desarrollo Orientado a Objetos con UML. S.l.: Facultad de Informática – UPM, 2008.

Flores, Antonio. 2008. Introducción al Lenguaje de Programación PHP. Madrid: s.n., 2008.

Meléndrez, Edelsys Hernández. 2006. Cómo escribir una tesis. La Habana: s.n., 2006.

Potencier, Fabien. 2009. El tutorial Jobeet. 2009.

Glosario de Términos

- ***Aduana:*** es la institución de un país encargada de aplicar la legislación aduanera y de recaudar los derechos e impuestos que se aplican a la importación, a la exportación, al movimiento o al almacenaje de mercancías.

- ***AGR:*** Aduana General de la República de Cuba.

- ***CADI:*** Centro de Automatización para la Dirección y la Información.

- ***CEIGE:*** Centro de Informatización para la Gestión de Entidades.

- ***Entidad:*** se emplea para nombrar a una corporación, compañía u otra organización que se toma como persona jurídica.

- ***Ext JS:*** es una librería JavaScript ligera y de alto rendimiento, compatible con la mayoría de navegadores que nos permite crear páginas e interfaces web dinámicas.

- ***Framework:*** un *framework*, es una estructura de soporte definida mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

- ***GINA:*** Gestión Integral de Aduanas.

- ***IDE:*** Entorno de Desarrollo Integrado. Consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI).

- ***JavaScript:*** es un lenguaje de programación interpretado utilizado principalmente en la realización páginas web. Presenta una sintaxis semejante a la del lenguaje Java y el lenguaje C.

- ***OMG:*** de sus siglas en inglés *Grupo de Gestión de Objetos* es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI, CORBA.

Glosario de Términos

- ***PHP:*** PHP Hypertext Pre-processor es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas.
- ***Proceso:*** es un conjunto de actividades (coordinado u organizado) que se realizan o suceden (alternativa o simultáneamente) bajo ciertas circunstancias con un fin determinado.
- ***RCE:*** Registro Central de Entidades.
- ***TOC:*** Tamaño Operacional de Clase.
- ***Trabajador externo:*** es aquel trabajador que pertenece a una entidad pero no a una aduana.

Anexos

Anexo #1 Descripción de los requisitos principales, prototipos y diagramas de secuencia.

Descripción del requisito Registrar Incidencia:

Precondiciones		El especialista de RCE se ha autenticado en el sistema y debe tener los permisos para gestionar las incidencias.
Flujo de eventos		
Flujo básico Registrar Incidencia		
1		Se selecciona en el menú principal de Registro Central de Entidades la opción “Gestión de trabajadores”.
2		Se selecciona un trabajador y despliega la opción “Historial Trabajador”.
3		Se selecciona la opción “Gestionar Incidencia”.
4		Se selecciona en la interfaz mostrada la opción “Registrar”.
5		Se introducen los datos de la Incidencia: Fecha del suceso, Descripción, Aduana, Tipo de objeto asociado y Tipo de incidencia.
6		Se da clic en el botón “Aceptar”.
7		Se validan los datos introducidos.
8		Se muestra el mensaje “La incidencia ha sido registrada correctamente”.
9		Concluye el requisito.
Pos-condiciones		
1		Se registran los datos introducidos correctamente.
Flujos alternativos		
Flujo alternativo 7.a Datos Incorrectos		
1		El sistema señala en rojo el campo con los datos incorrectos.
2		Regresa al paso 5 del flujo básico Registrar Incidencia.
Flujo alternativo *.a Selecciona la opción “Cancelar”		
1		Se regresa a la interfaz “Gestionar Incidencia”.
Pos-condiciones		
1		N/A
Validaciones		
1		Se validan los datos según lo establecido en el Modelo conceptual CIG-ADU-N-CFGr-e1301 Modelo_Conceptual_Registro_Central.doc.
Conceptos	**Incidencia**	Visibles en la interfaz: ▪ Tipo de Incidencia, descripción, fecha de suceso, aduana y tipo de objeto. Utilizados internamente: ▪ Subsistema y autor.
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

Prototipo elemental de interfaz gráfica de usuario:

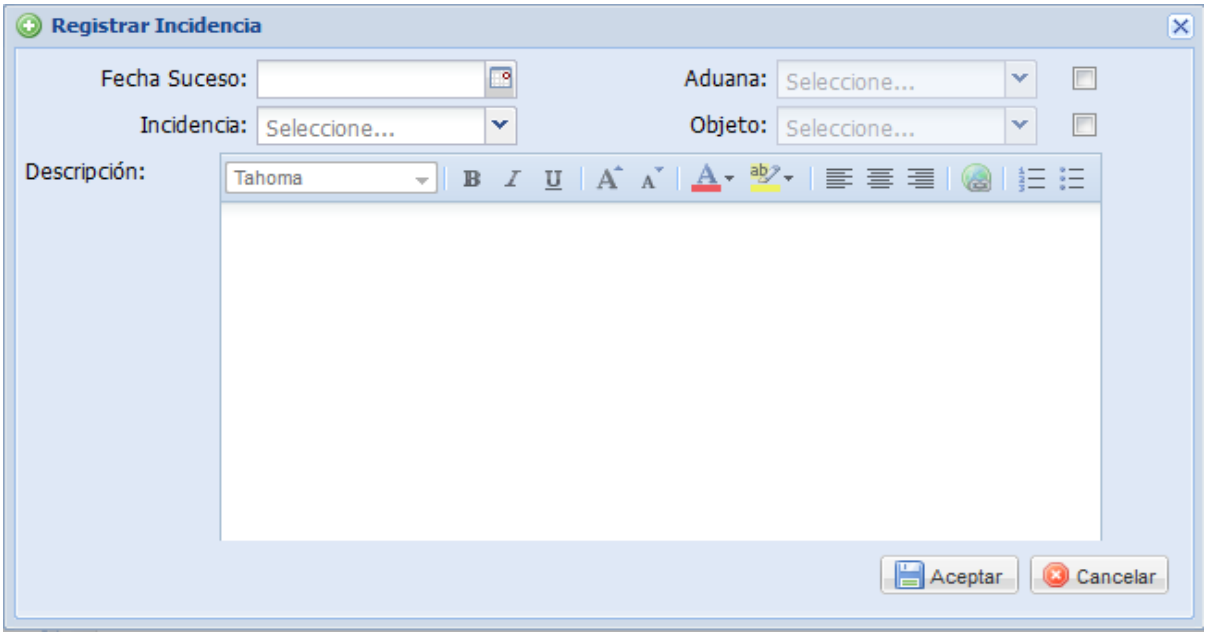
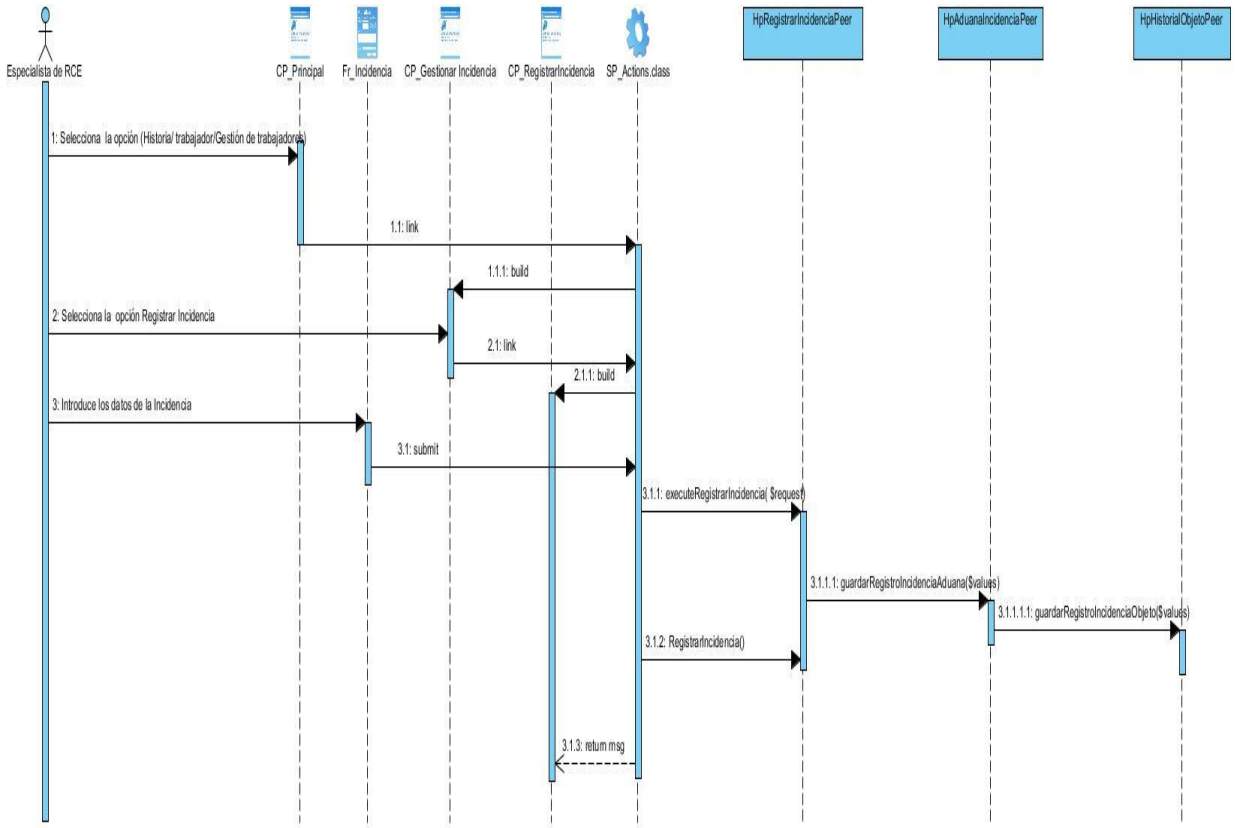


Diagrama de Secuencia:



Descripción del requisito Eliminar Incidencia:

Precondiciones	El especialista de RCE se ha autenticado en el sistema y debe tener los permisos para gestionar las incidencias. Debe existir al menos una incidencia registrada.
Flujo de eventos	
Flujo básico Eliminar Incidencia	
1	Se selecciona en el menú principal de Registro Central de Entidades la opción “Gestión de trabajadores”.
2	Se selecciona un trabajador y despliega la opción “Historial Trabajador”.
3	Se selecciona la opción “Gestionar Incidencia”.
4	Se selecciona la incidencia y presiona el botón “Eliminar”.
5	Se muestra un mensaje para confirmar si se desea eliminar o no la incidencia seleccionada.
6	Selecciona el botón “Si”.
7	Se muestra el mensaje “La incidencia ha sido eliminada correctamente”.
8	Concluye el requisito
Pos-condiciones	
1	Se elimina la incidencia seleccionada de la base de datos.
Flujos alternativos	
Flujo alternativo 5.a Selecciona el botón “No”	
1	Se regresa a la interfaz de “Gestionar Incidencia”.
Pos-condiciones	

2	N/A
Validaciones	
2	
Requisitos especiales	N/A
Asuntos pendientes	N/A

Prototipo elemental de interfaz gráfica de usuario:

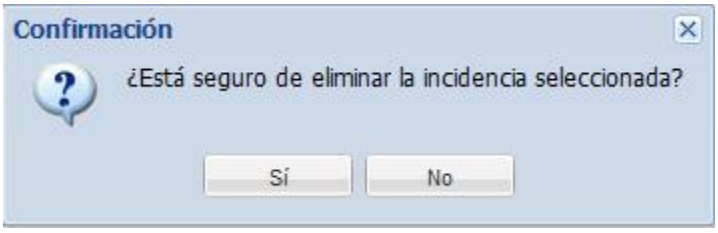


Diagrama de Secuencia: