UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 3

Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas.

Título: Gestor de dependencias para el sistema de Gestión Integral de Aduanas 1.0

Autor (a): Yurismara León Ricardo

Tutor (es): Ing. Manuel R Almaguer Ochoa
Ing. Yasel Antonio Romero Piñeiro

La Habana, Junio del 2012

"Año 54 del Triunfo de la Revolución"

Declaración de Autoría

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

Para que así conste firmo la presente a los ____ días del mes de ____ del año 2013.

Yurismara León Ricardo

_____

Firma del Autor

Ing. Manuel R. Almaguer Ochoa

_____

Firma del Tutor

Ing. Yasel A. Romero Piñeiro

_____

Firma del Tutor

"Se pueden adquirir conocimientos y conciencia a lo largo de toda la vida, pero jamás en ninguna otra época de su existencia una persona volverá a tener la pureza y el desinterés con que, siendo joven, se enfrenta a la vida.

Agradecimientos

A mi mamá, madre excepcional que me ha apoyado para superar todos los retos que me ha puesto la vida, me ha dado fuerza para llegar hasta aquí y poder luchar por el éxito, la que me ha dado todo sin importar los sacrificios que ha tenido que hacer para mantenerme y llegar hasta donde he llegado hoy.

A mi hermana Yudisleydis que vive convencida de que no la voy a defraudar y me ha dado mucho apoyo emocional y eso me ha dado fuerzas para seguir adelante ante las adversidades.

A mi sobrina Meylan que sé que aunque es pequeña y no sabe, ha sido un gran apoyo para su tía y eso me ha proporcionado mucha energía para superarme.

A mi abuela Juana que siempre me apoyó y me alentó a superarme y estando grave al borde de su muerte siguió alentándome a seguir adelante, abuela te extraño mucho, espero que donde quiera que estés me estés mirando y estés orgullosa de tu nieta que te ama.

A mis tíos y primos por su apoyo y ayuda.

Le agradezco a mi tutor Yasel, por su apoyo y ayuda, por alentarme a creer en mí.

A mi amiga Lilian Hernández, más que amiga hermana por su apoyo, ayuda por ser incondicional y por estar presente tanto en momentos difíciles como de alegría, gracias por esa amistad inquebrantable que comenzó en nuestro primer año.

A mi amiga Juana por darme su apoyo y amistad.

A mi amiga Yudy, por su apoyo, por alentarme a mirar adelante cuando las cosas van mal, por estar presente en los momentos difíciles, por ser incondicional y darme la mano para continuar con la lucha.

A Ivan, por darme su ayuda y apoyo cada vez que lo necesité, por ser una de las personas con las que siempre podré contar y por darme la oportunidad de ser su amiga.

A mi amiga Liliam, por su apoyo, ayuda y por ser una amiga incondicional.

A mi amiga Yiset, por ser una persona incondicional y por darme su apoyo.

A Aliuska por estar ser parte de mi círculo de amistades.

A Dagmar, por su ayuda, paciencia y dedicación con programación, por ser incondicional.

Le agradezco a Adrian por su ayuda y por alentarme siempre.

Le agradezco a todas aquellas personas que hicieron más a mena mi estancia en la uci, gracias.

Dedicatoria

Dedico mi esfuerzo a las personas más importantes en mi vida, a mi madre querida, que es mi tesoro más preciado, mi razón de ser, de existir, de seguir adelante en la vida, mi fuerza, mi apoyó, mi guía, te dedico este título mami, porque tú has sido mi motor impulsor.

A mi hermana y a mi sobrinita que son mi todo, mi vida y mi tesoro.

A mi abuela que aunque no está físicamente yo la llevo en el corazón, este título es también para ti.

A mi padre, que aunque no he tenido su apoyo lo quiero con la vida.

Resumen

Actualmente, llevar el control de las dependencias es una tarea de mucha responsabilidad e importancia para los subsistemas que integran al sistema Gestión Integral de Aduanas (GINA). La presente investigación tuvo como objetivo fundamental concebir un componente que gestione las dependencias de dicho sistema, permitiendo emplear menos tiempo y recursos en el proceso de despliegue, soporte y mantenimiento en el ciclo de vida de los productos de software del mismo. Se realizó un estudio previo de los sistemas informáticos que gestionan las dependencias de proyectos, analizando sus principales características. Posteriormente se llevó a cabo un estudio de las tecnologías y herramientas que fueron usadas para idear la solución. Haciendo uso del Modelo de Desarrollo del Centro de Informatización de la Gestión de Entidades, se obtuvo un componente que se integró al sistema GINA, el cual es capaz de mantener el control de las dependencias entre los subsistemas.

Contenido

Contenido

Contenido

Introducción

En el año 2002 surge la Universidad de las Ciencias Informáticas (UCI) como parte del proceso de informatización de la sociedad cubana y con ella una nueva era abriendo puertas a una industria de producción de software, brindando servicios a empresas nacionales e internacionales. La UCI está comprometida con el desarrollo de aplicaciones y servicios informáticos con el objetivo de servir de soporte a la industria cubana de la informática. (1)

Con este compromiso de informatización, se establecen varias relaciones de empresas con la UCI, y entre ellas la Aduana General de la República de Cuba (AGR), para informatizar los principales procesos que allí llevan a cabo.

La AGR es el organismo encargado de regular el control aduanero. Esta organización se crea el 5 de febrero de 1963, con el propósito de garantizar la seguridad nacional de Cuba. (2)

La AGR necesita sistemas que garanticen mejorar la capacidad de gestión de la aduana y su eficacia administrativa con modernización tecnológica. Por esta razón, busca elevar continuamente la excelencia de sus servicios.

En la actualidad el crecimiento acelerado de la ciencia trae consigo la colaboración entre procesos de software por lo que aumenta la homogeneidad entre diferentes procesos para optimizar tiempo y crear proyectos colaborativos que incrementan la eficiencia, eficacia y el ahorro de tiempo logrando un objetivo más abarcador. La amplia gama de productos de software a veces carecen de una integración entre ellos, lo que hace que se pierda tiempo en desarrollar nuevos sistemas sin tener en cuenta que a veces la solución está en la integración de uno o más productos de software. Existen muchas empresas desarrolladoras de software que han dado magníficas soluciones, garantizando mayor flexibilidad de integración y soporte de sus productos como un valor agregado que satisface al cliente.

La UCI se encuentra en estos momentos inmersa en el desarrollo de sistemas de software a partir de convenios que han sido firmados con diferentes empresas. Muchos de los productos que se implementan exigen que tengan un control minucioso de las relaciones entre los subsistemas, los cuales son los encargados de realizar cada una de las subtareas en que se divide la tarea principal de los productos que serán instalados finalmente en dichas empresas. Los clientes exigen en sus contratos que sus productos sean altamente integrables en el proceso de instalación, actualización, configuración y eliminación de paquetes de software, lo

que les garantice confiabilidad e integridad de la información a la hora de desplegar el producto, así como la gestión del soporte como valor agregado en la vida útil del software.

Con el objetivo de llevar el control de los procesos de la AGR, se desarrolla el sistema de Gestión Integral de Aduanas (GINA) por parte del Departamento de Soluciones para la Aduana perteneciente al Centro de Informatización de la Gestión de Entidades de la UCI. El mismo está compuesto por un conjunto de subsistemas, clasificados en aplicaciones y complementos, donde las aplicaciones pueden estar formadas por varios módulos.

Actualmente en el sistema GINA, no se garantiza la gestión de las dependencias entre los subsistemas en el proceso de despliegue, soporte y mantenimiento en el ciclo de vida de estos productos de software, lo que trae consigo la instalación del sistema completo cuando se va a desplegar un subsistema, dificultando el cumplimiento de dicho proceso y ocupando mucho más tiempo al realizarlo.

El desconocimiento de las dependencias existentes entre estos subsistemas provoca que se extienda el despliegue de un determinado subsistema, aumentando el tiempo y los recursos destinados a dicho proceso.

Otro problema surge al dar soporte y mantenimiento a un subsistema cuando ha sido modificado después de su despliegue, puesto que al este relacionarse con otros subsistemas, dicho cambio puede repercutir sobre estos últimos viéndose afectado el funcionamiento del sistema.

Estos problemas persisten debido a que no se dispone de una herramienta capaz de gestionar las dependencias de forma automatizada.

A partir de la problemática planteada surge el siguiente **problema a resolver:** ¿Cómo lograr una correcta gestión de dependencias entre los componentes del sistema GINA de la Aduana General de la República, de manera que se emplee menos tiempo y recursos en el proceso de despliegue, soporte y mantenimiento en el ciclo de vida de sus productos de software?

El **objeto de estudio** de este trabajo son los gestores de dependencias en aplicaciones informáticas. El **campo de acción** se restringe a aquellos gestores de dependencia que utilizan como lenguaje de programación PHP.

Introducción

Se define como **objetivo general** desarrollar un componente gestor de dependencias que permita emplear menos tiempo y recursos en el proceso de despliegue, soporte y mantenimiento durante el ciclo de vida de los productos de software del sistema GINA.

La **idea a defender** del presente trabajo plantea que con el desarrollo de un sistema gestor de dependencias, se logrará garantizar que se emplee menos tiempo y recursos en el proceso de despliegue, soporte y mantenimiento en el ciclo de vida de los productos de software para el sistema GINA.

Se trazan como **objetivos específicos:**

1. Elaborar el marco teórico relativo a la investigación para obtener un estado del arte que permita tener un conocimiento amplio sobre el tema.
2. Realizar el análisis y diseño de la propuesta de solución que permita identificar los requerimientos significativos para el sistema y generar los artefactos correspondientes.
3. Implementar la solución modelada de manera que cumpla con las necesidades del cliente.
4. Validar la solución mediante pruebas unitarias y funcionales aplicadas al sistema para garantizar la calidad del producto.

Como **posibles resultados** se obtendrá un componente que gestione las dependencias entre los subsistemas del sistema GINA.

Para cumplir con el objetivo y lograr una solución adecuada a la problemática especificada, se plantean las siguientes **tareas de la investigación**:

- Estudio de la evolución y el comportamiento de los sistemas informáticos similares al que se desea obtener en la investigación.
- Estudio de las tecnologías y lenguajes existentes.
- Realización de un análisis de la información relacionada con los sistemas informáticos desarrollados con el *framework* Symfony.
- Especificación de los requisitos.
- Descripción de los requisitos.
- Validación de los requisitos.
- Diseño e implementación de las interfaces de usuario.
- Diseño del diagrama de clases del diseño.

- Diseño de los diagramas de secuencia.
- Implementación de los requisitos.
- Diseño de los casos de prueba.
- Aplicación de las pruebas al sistema.
- Documentación de los resultados de las pruebas al sistema.

En el transcurso del desarrollo de este trabajo se hace uso de algunos métodos científicos de la investigación que ayudan a guiar exitosamente el desarrollo de la solución.

1. **Histórico-lógico:** permite realizar un estudio del estado del arte del tema a investigar. De esta manera se puede conocer acerca de la existencia y características de sistemas de este tipo que hayan sido desarrollados anteriormente.
2. **Analítico-sintético:** permite analizar teorías y documentos, y a partir de ello realizar un estudio previo acerca de las características y la necesidades de una herramienta que facilite la obtención de dependencia entre subsistemas del sistema GINA.
3. **Modelación**: permite realizar abstracciones para dar una explicación de la realidad existente. Su condición principal es la relación entre el modelo y el objeto que se modela, que en este caso sería el sistema en cuestión.
4. **Observación**: permitió detectar de forma directa la necesidad de un sistema gestor de dependencias automático.

El presente documento se encuentra estructurado en tres capítulos de la forma siguiente:

Capítulo 1 **"Fundamentación teórica"**: donde se estudian y documentan un conjunto de conceptos necesarios para el entendimiento de los términos del trabajo. Además realiza un estudio y análisis de las herramientas que se usan actualmente para detectar dependencias haciendo uso de los gestores de dependencias.

Capítulo 2 **"Análisis y diseño de la propuesta de solución del sistema"**: se realiza la descripción de todos los artefactos que se generan durante el análisis y diseño del sistema. Se definen los requerimientos del sistema, el diseño de los prototipos de interfaz de usuario, diagramas de clases, diagramas de secuencia, diagramas de componentes, diagrama de paquetes y diagramas de actividades con estereotipos web.

Introducción

Capítulo 3 **“Implementación y prueba de la Solución”:** se construye la solución propuesta, pasando por la etapa de implementación y se documentan a partir de la solución, los resultados obtenidos de las pruebas realizadas al sistema en diferentes escenarios.

Capítulo 1. Fundamentación teórica.

1.1 Introducción

El presente capítulo describe los principales elementos que fundamentan teóricamente el desarrollo de una herramienta para la gestión de dependencias del sistema GINA, incluyendo un estudio de algunas de las soluciones existentes en el mercado mundial para gestionar dependencias, que se ejecutan en sistemas distribuidos similares a la solución que se propone con la investigación y sus fundamentales conceptos asociados.

1.2 Conceptos básicos

Aplicación

Una aplicación es un tipo de programa informático diseñado como una herramienta para permitir a un usuario realizar uno o diversos tipos de trabajo. Es una solución informática para la automatización de tareas complicadas. Las aplicaciones desarrolladas «a medida» ofrecen gran potencia ya que están exclusivamente diseñadas para resolver un problema específico. (3)

Módulo

Un módulo es una porción de una aplicación, es una de las varias tareas que debe realizar un programa para cumplir con su función u objetivos, y una extensión más ligera y flexible para la presentación de contenido. Pueden asociarse a diferentes componentes. (3)

Plugin (complemento)

Un *plugin* es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica, es ejecutada por la aplicación principal. También se conoce como *plugin*, conector o extensión. (4)

Dependencia de *software*

Una dependencia de *software* es una aplicación o una biblioteca requerida por otro programa para poder funcionar correctamente. La información de dependencia se define como los requisitos y capacidades (5). Las dependencias son los vínculos y tipos de enlaces, entre todas las tareas de un proyecto y se utilizan para identificar las relaciones entre las piezas de *software*.

1.3 Sistemas informáticos para la gestión de dependencias

A continuación se muestra un estudio de algunas herramientas de gestión de dependencias, teniendo en cuenta los siguientes indicadores para su caracterización: lenguaje de programación (PHP), adaptable a *framework* Symfony 1.x, código abierto y tipo de licencia.

1.3.1 Composer

Composer es una herramienta que permite declarar dependencias entre las librerías de un proyecto y las instala. Gestiona las dependencias escritas en el lenguaje PHP, está desarrollado sobre la base del *framework* Symfony 2.0, es de código abierto. (6)

1.3.2 PEAR

PEAR o PHP *Extension and Application Repository*, es un entorno de desarrollo y sistema de distribución para componentes de código PHP. Proporciona una manera fácil de instalar, desinstalar o actualizar con nuevos paquetes. (7) Gestiona las dependencias explícitamente entre paquetes PEAR, es una herramienta de código abierto.

1.3.3 YUM

YUM *(Yellow Dog Updater, Modified)* es un gestor de paquetes avanzados incluido por defecto en un sistema operativo llamado Fedora, basado en Linux. Permite instalar, desinstalar y actualizar paquetes resolviendo las dependencias automáticamente. (8) Es una herramienta libre, escrita en Python, diseñada para la gestión de paquetes en distribuciones de GNU/Linux.

1.3.4 Maven

Maven es una herramienta de gestión de proyectos, basada en un fichero central, donde se define todo lo que necesita un proyecto. Maneja las dependencias del proyecto, compila, empaqueta y ejecuta las pruebas. (9). Es una herramienta libre, gestiona las dependencias de cualquier proyecto basado en Java.

1.3.5 Resumen del estudio del estado del arte

Estas aplicaciones informáticas no se toman como solución debido a que no cumplen con los indicadores definidos, reflejándose de forma explícita en la siguiente tabla:

	Gestor de dependencia			
Indicadores	Composer	Pear	Maven	Yum
Lenguaje de Programación (PHP)	✓	✓		
Adaptable a Symfony 1.x	*			

Código abierto	✓	✓	✓	✓
Herramienta libre	✓	✓	✓	✓
* Es posible adaptarlo al *framework* Symfony 1, aunque el propio creador del *framework* no lo recomienda.				

Tabla 1. Indicadores a medir para la caracterización de los gestores de dependencias.

Composer es la herramienta que más se acerca a la solución deseada debido a que cumple, hasta cierto punto, con los indicadores establecidos, porque a pesar de que es adaptable a versiones anteriores del *framework* Symfony2, no se toma como solución porque su adaptación implicaría mucho más tiempo y recursos que los requeridos para desarrollar una nueva herramienta que gestione las dependencias en versiones inferiores al *framework* Symfony2.

1.4 Modelo de desarrollo, tecnologías y herramientas utilizadas

Hoy en día el desarrollo de *software* está apuntando hacia la web, y sus productos son cada vez más enfocados hacia las aplicaciones web, la cual representa un elemento que ha contribuido considerablemente al desarrollo de numerosas empresas. Estas aplicaciones presentan ventajas significativas en cuanto al ahorro de tiempo de los procesos, compatibilidad y disponibilidad de la información.

En estudios previos realizados al desarrollo del sistema GINA se definió la línea base de la arquitectura la cual está compuesta por las principales tecnologías empleadas en el desarrollo del sistema. A continuación se muestra una descripción de la metodología de desarrollo y de algunas tecnologías y herramientas haciendo énfasis en sus principales características.

1.4.1 Modelo de desarrollo

La metodología de desarrollo de software es el marco usado para estructurar, planear y controlar el proceso de desarrollo de un proyecto, que en dependencia de la plataforma de desarrollo y lenguaje de programación, utiliza herramientas que permiten obtener los diferentes artefactos y el producto final. (10)

El sistema GINA surge como resultado de la aplicación del modelo de desarrollo elaborado por el centro CEIGE, por lo que se escoge el mismo como modelo de desarrollo para la solución propuesta. Dicho modelo tiene en cuenta las actividades de cada una de las fases y áreas de procesos que plantea 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. (10)

En la **figura 1** se pueden apreciar las disciplinas que integran al modelo de desarrollo.

Fig.1 Ciclo de vida de proyectos del CEIGE.

Descripción de cada una de las disciplinas del modelo de desarrollo:

- **Estudio preliminar**: es donde se realizan las actividades relacionadas con la planeación del proyecto y se lleva a cabo un estudio inicial de la organización, permitiendo obtener información sobre el alcance del proyecto y realizar estimaciones de tiempo, esfuerzo y costo.
- **Modelado del negocio:** es la fase que 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:** en esta fase el esfuerzo principal es desarrollar un modelo del sistema que se va a construir. Incluyendo un conjunto de 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:** en esta fase se modela el sistema para que soporte todos los requisitos, lo que contribuye a una arquitectura sólida y estable para convertirse en un plano para la próxima fase.
- **Implementación:** a partir de la fase anterior lleva a cabo la implementación del sistema

en términos de componentes, es decir, ficheros de código fuente, scripts, ejecutables y similares.

- **Pruebas internas:** en esta fase 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, es decir requisitos que el producto debería cumplir y que aún no los cumple.
- **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. (10)

1.4.2 Lenguajes de programación

Un lenguaje de programación no es más que un conjunto de sintaxis y reglas semánticas que definen los programas del computador. Es una técnica estándar de comunicación para entregarle instrucciones al computador. Un lenguaje le da la capacidad al programador de especificarle al computador, qué tipo de datos actúan y qué acciones tomar bajo una variada gama de circunstancias, utilizando un lenguaje relativamente próximo al lenguaje humano (11)**.**

Actualmente se pueden encontrar varios lenguajes de programación que son utilizados para el desarrollo de distintas soluciones, partiendo de aquellos privativos por los cuales es necesario pagar una patente y llegando hasta los lenguajes de carácter libre cuyo soporte es dado por una comunidad de usuarios que deseen colaborar.

1.4.3 Lenguaje utilizado en el lado del cliente

El siguiente lenguaje fue elegido para realizar la programación en el lado del cliente, este es el lenguaje que se ejecuta en el navegador del usuario.

JavaScript es un lenguaje de programación del lado del cliente inventado por Brendan Eich en la empresa *Netscape Communications*. Usado en las páginas web para agregar mayor funcionalidad, interacción o animaciones. Es un lenguaje de script multiplataforma orientado a objetos. Es un lenguaje pequeño y ligero; no es útil como un lenguaje independiente, más bien está diseñado para una fácil incrustación en otros productos y aplicaciones, tales como los navegadores web. (12)

Entre sus principales características es posible citar:

- Es interpretado por el cliente.

- Está basado en objetos.
- Su código se integra en las páginas XHTML, incluido en las propias páginas.
- Las referencias a objetos se comprueban en tiempo de ejecución, por lo tanto no se compila.
- Es independiente de la plataforma, por lo que se ejecuta en cualquier sistema operativo. (13)

En el sistema GINA se utiliza JavaScript, para manejar objetos dentro de las páginas web. El cual facilita la programación de páginas interactivas.

1.4.4 Lenguajes utilizados en el lado del servidor

El sistema GINA se ha implementado sobre la base del PHP como lenguaje de programación del lado del servidor.

El PHP es un lenguaje de script incrustado dentro del HTML. La meta de PHP es permitir rápidamente a los desarrolladores la generación dinámica de páginas. (14)

Cuenta con cuatro grandes características que hacen que este lenguaje sea distinguido entre los demás existentes en el mundo:

- **Velocidad:** PHP es rápido al ser ejecutado, lo cual es importante, debido a que no crea demoras en la máquina, por esta razón no requeriere demasiados recursos de sistema.
- **Estabilidad:** PHP goza de la ayuda de una gran comunidad de desarrollo, permitiendo que los fallos de funcionamiento se encuentren y se reparan rápidamente. PHP utiliza su propio sistema de administración de recursos y dispone de un sofisticado método de manejo de variables, conformando un sistema robusto y estable.
- **Seguridad:** PHP provee diferentes niveles de seguridad, estos pueden ser configurados de archivos de configuración que se encuentran en el servidor.
- **Simplicidad:** es un lenguaje de programación simple de implementar por lo que usuarios con experiencia en C y C++ podrán utilizar PHP rápidamente. (15)

Para el desarrollo del sistema GINA se emplea el lenguaje de programación PHP en su versión 5.3.8 o superior.

1.4.5 Lenguajes de marcado

Los lenguajes de marcado suelen confundirse con lenguajes de programación. Sin embargo, no son lo mismo, debido a que el lenguaje de marcado no tiene funciones aritméticas o variables,

como sí poseen los lenguajes de programación. Históricamente, el marcado se usaba y se usa en la industria editorial y de la comunicación, así como entre autores, editores e impresores. (16)

Un **lenguaje de marcas** es un lenguaje que anota el texto de modo que el ordenador puede manipularlo. La mayoría de los lenguajes de marcas son legibles debido a que las anotaciones están escritas de forma tal que se puedan distinguir de los textos. (16)

A continuación se muestran algunos lenguajes de marcado que son utilizados por el *framework* Symfony.

YAML *(Ain't Another Markup Language)* en castellano quiere decir*, "YAML* no es otro lenguaje de marcado*.* Es un estándar para la serialización de datos amigable con los humanos para todos los lenguajes de programación. Es un formato ideal para archivos de configuración. Los archivos YAML son tan expresivos como los archivos XML y tan fáciles de leer como los archivos INI. (17)

Uno de los objetivos del componente YAML de Symfony es encontrar el balance adecuado entre características y velocidad. Es compatible con las características necesarias para manipular archivos de configuración. (17)

Este exporta un analizador real y es capaz de analizar un gran subconjunto de especificaciones YAML, para todas las necesidades de configuración. También significa que el analizador es bastante robusto, fácil de entender, y lo suficientemente simple como para ampliarlo. También es capaz de volcar arreglos de PHP a YAML con apoyo a objetos, y el nivel de configuración en línea es bastante bueno. Es compatible con la mayoría de los tipos YAML integrados como fechas, números enteros, octales, booleanos, y mucho más. Completa compatibilidad para referencias, alias y completa fusión de claves. (17)

XML *(Extensible Markup Language)* es un lenguaje de etiquetas, es decir, cada paquete de información está delimitado por dos etiquetas como se hace también en el lenguaje HTML, pero XML separa el contenido de la presentación. (18)

XML se plantea como un lenguaje estándar para el intercambio de información entre diferentes programas de una manera segura, fiable y libre, debido a que no pertenece a ninguna compañía. A partir de un documento en XML se pueden generar archivos PDF y en otros

formatos. De esta forma, la información puede ser presentada de una manera visual para su lectura por las personas y el XML sólo quedará para ser entendido por los programas; aunque si hacemos un esfuerzo, vemos que es fácil para una persona extraer la información de un documento XML directamente. (18)

INI (extensión de archivo) es un archivo de texto con un formato bastante simple, con posibilidad de contener secciones, declaradas con una línea [nombre_sección]. Son archivos bastante simples y pueden llegar a ser útiles, aunque no sean demasiado flexibles. Tiene un formato específico y se usa como repositorio de datos. (19)

Tipos de entradas de un archivo **.INI**:

- **Secciones:** permiten agrupar parámetros relacionados. Por ejemplo: "Parámetros de red".
- **Valores:** definen parámetros y su valor. Primero se define el nombre del parámetro y después su valor separado por el signo de igualdad (=).
- **Comentarios:** permiten explicar el propósito de una sección o parámetro. Los comentarios comienzan con el carácter punto y coma (;). (19)

Estructura del archivo INI

- Los archivos INI utilizan corchetes para incluir secciones. En estas secciones se encuentran las claves y los valores de las claves.
- La clave y su valor están separados mediante el símbolo (=).
- Los nombres de sección se escriben entre corchetes ([UnaSeccion]). Los archivos INI no son tan populares como los archivos XML, debido a que no manejan muy bien grandes cadenas de texto. (19)

Como se ha visto, YAML es mucho más rápido de escribir que XML (debido a que no hacen falta las etiquetas de cierre y el uso continuo de las comillas) y es mucho más poderoso que los tradicionales archivos **.ini** (puesto que estos últimos no soportan la herencia y las estructuras complejas). Por este motivo, el *framework* Symfony utiliza el formato YAML como el lenguaje preferido para almacenar su configuración.

Por los motivos antes mencionados, se decidió utilizar YAML para generar las dependencias existentes, debido a que el núcleo del *framework* Symfony posee un componente que se encarga del manejo de ficheros YAML. El formato YAML indica su estructura mediante la tabulación y es muy rápido de escribir.

1.5 Frameworks

Un *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 otro *software*, para ayudar a desarrollar y unir los diferentes componentes de un proyecto. (20)

1.5.1 Framework Ext JS

Ext JS es una librería JavaScript ligera y de alto rendimiento, compatible con la mayoría de los navegadores, nos permite crear páginas e interfaces web dinámicas. (21)

Esta librería JavaScript facilita el desarrollo de aplicaciones web multiplataforma. Ext JS ofrece la posibilidad de emplear un gran número de componentes visuales que mejoran la calidad de las aplicaciones. Plantea validaciones de formularios, basándose en expresiones regulares y tipos de datos.

La utilización de este *framework* de JavaScript favorece la separación de las capas de la vista con la del controlador desde el punto de vista productivo, debido a que el código utilizado en la primera es solamente JavaScript y no es necesario utilizar ningún tipo de código PHP, así los desarrolladores pueden centrarse más en el aprendizaje de un solo lenguaje. Además, al soportar la serialización de objetos mediante la tecnología de Notación de Objetos de JavaScript (JSON), permite que los datos enviados desde el controlador como respuesta a la vista contengan sólo las propiedades de dichos objetos, pero no el comportamiento, minimizando los posibles errores de programación y los accidentes de que los objetos sean modificados erróneamente desde la vista. (21)

Para validar datos antes de ser enviados al servidor y para proveer al sistema de una buena apariencia y usabilidad, es utilizado el *framework* Ext JS en su versión 3.0.

1.5.2 Framework Symfony

Symfony es un completo *framework* diseñado para optimizar, el desarrollo de aplicaciones web. Este *framework* separa la lógica del negocio, la lógica del servidor y la presentación de la aplicación web. Facilita varias herramientas y clases encaminadas a minimizar el tiempo de desarrollo de una aplicación web compleja. (22)

Las clases de la capa del modelo se generan automáticamente, en función de la estructura de datos de la aplicación. La librería Propel es un proyecto de software libre, es una de las mejores capas de abstracción de objetos/relacional disponibles en PHP 5, se encarga de esta

generación automática, ya que crea el esqueleto o estructura básica de las clases y genera automáticamente el código necesario. (22)

Symfony está desarrollado completamente con PHP 5. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Su carácter libre y multiplataforma lo hacen uno de los *framework* de PHP más usados en el mundo. (22)

Symfony se diseñó para que se ajustara a los siguientes requisitos:

- Funciona con todas las bases de datos comunes (MySQL, PostgreSQL, SQLite, Oracle, MS SQL Server).
- Sencillo de usar en la mayoría de casos, pero lo suficientemente flexible como para adaptarse a los casos más complejos.
- Basado en la premisa de "convenir en vez de configurar", en la que el desarrollador sólo debe configurar aquello que no es convencional.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web.
- Preparado para aplicaciones empresariales y adaptables a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- Código fácil de leer que incluye comentarios de *phpDocumentor* y que permite un mantenimiento muy sencillo.
- Fácil de extender, lo que permite su integración con librerías desarrolladas por terceros. (22)

Para el desarrollo del sistema GINA es utilizado el *framework* Symfony en su versión 1.2.8, para proporcionar robustez y seguridad al futuro producto de software.

1.6 Entornos de Desarrollo Integrado (IDE)

Un Entorno de Desarrollo Integrado en inglés (*Integrated Development Environment* o IDE) es un programa compuesto por una serie de herramientas que utilizan los programadores para desarrollar código. (23)

1.6.1 NetBeans IDE

NetBeans IDE es una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Es una aplicación de código abierto ("*open source*") diseñada para el desarrollo de aplicaciones fácilmente portables entre las distintas plataformas, haciendo uso de la tecnología Java. Puede servir para cualquier otro lenguaje de programación. Existe

además un número importante de módulos para extender el NetBeans IDE. (24) Posee un amplio soporte para el lenguaje PHP así como para los *framework* Symfony y Ext JS.

Para el desarrollo de la solución se ha elegido NetBeans IDE, en su versión 7.2.1, como Entorno de Desarrollo Integrado.

1.7 Herramientas de diseño

En la actualidad, varias empresas se han expandido a la adquisición de herramientas CASE, estas son las siglas en inglés que se utilizan para referirse a Ingeniería de *Software* Asistida por Computadora, con el único fin de aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero. Estas herramientas pueden automatizar aspectos claves de todo el proceso de desarrollo de un sistema e incrementar su posición en el mercado competitivo.

1.7.1 Visual Paradigm for UML

Visual Paradigm for UML (VP-UML) es un modelador UML que permite el diseño de sistemas con todo tipo de diagramas UML. También es utilizado para diseñar diagramas de casos de uso, diagramas de requerimiento y diseño de bases de datos relacionales. Con VP-UML, el equipo de desarrollo de software puede realizar el análisis y diseño de sistemas con eficacia. (25)

Visual Paradigm es una herramienta multiplataforma, se integra con algunas herramientas realizadas en Java entre las cuales se puede citar el NetBeans IDE.

Para la realización del diseño de la solución se elige la herramienta Visual Paradigm for UML en su versión 3.4.

1.8 Conclusiones parciales

En el presente capítulo se realizó un estudio de los principales sistemas de gestión de dependencias, el cual permitió tomar a Composer como guía base para dar solución al problema planteado. Se sentaron las bases que fundamentaron las tecnologías y herramientas a utilizar.

Como línea base de la arquitectura se empleó el modelo de desarrollo propuesto por el CEIGE, Ext JS en su versión 3.0 como *framework* en el lado del cliente y Symfony en su versión 1.2.8 como *framework* en el lado del servidor.

Capítulo 2. Análisis y Diseño

2.1 Introducción

En el presente capítulo se aborda la solución para el problema planteado, desarrollándose el análisis y diseño de la misma. Se realiza un levantamiento de los requisitos funcionales del sistema a implementarse, igualmente la descripción de los artefactos del análisis y diseño que se generan durante el flujo de trabajo. Se obtienen además varios diagramas como guía para el proceso de implementación del sistema.

2.2 Propuesta de solución

El *framework* Symfony presenta deficiencias en cuanto a la relación entre aplicaciones. Debido a esto las mismas no pueden acceder a los métodos o atributos de otras aplicaciones del mismo proyecto, estas se ejecutan de forma independiente respecto a las demás aplicaciones del mismo proyecto.

Por lo anteriormente planteado se decide el desarrollo de un *plugin* o complemento al cual será nombrado sfDependenciasPlugin para la obtención de dependencias de los componentes del sistema GINA. Este complemento aportará la capacidad para realizar la gestión de las dependencias de los componentes de forma automática.

Razones que llevaron al desarrollo del *plugin*:

- Para permitir la gestión de las dependencias entre los componentes que integran al sistema GINA.
- Para apoyar fácilmente la gestión de las dependencias del sistema GINA.
- Para brindarle servicios al instalador del sistema GINA.
- Para facilitar el proceso de despliegue, soporte y mantenimiento de una aplicación del sistema GINA.

El *plugin* sfDependenciasPlugin presenta varias características que proporcionan la correcta gestión de las dependencias, a continuación se muestran las mismas.

Características del *plugin*:

- Almacena las dependencias de cada componente en un fichero YAML. Este fichero se creará mediante líneas de comando, el mismo en un principio sólo tendrá la estructura por niveles en que se organizará cada uno de los componentes del sistema GINA.

- La actualización del fichero se realizará mediante dos formas: Interfaz de Líneas de Comando **(CLI)** e interfaz web. El fichero se actualizará mediante líneas de comandos con los componentes reales existentes del sistema GINA. Dará tres opciones de actualización, la primera opción será actualizar en el fichero sólo las aplicaciones del sistema, la segunda opción, actualizar solo los *plugins* y por último actualizará el fichero con todos los componentes. Las dependencias de un componente con otro se actualizará de forma manual a través de una interfaz web para comodidad del usuario.

- Posibilita la consulta de las dependencias de tres formas: CLI, Interfaz Web, Servicios Web. Mediante la CLI, introduciendo un comando se podrán observar las dependencias de los componentes. La interfaz web permitirá al usuario consultar las dependencias de los componentes, conocer algunos datos generales de los mismos, además de mostrarle las dependencias inversas para estar al tanto de los componentes que dependen de un componente determinado. Se proporcionarán servicios web al instalador del sistema GINA para la consulta de las dependencias de los componentes que se deseen instalar en las aduanas, además de facilitar los procesos de despliegue, soporte y mantenimiento de los componentes a instalar.

- Posibilita habilitar y deshabilitar los componentes registrados. (Interactúa con el componente de Administración). Esta función posibilitará habilitar o deshabilitar el componente deseado cuando se realice la instalación de uno de los mismos y se decida que no es necesario que se encuentre activa una o varias de sus dependencias, para este proceso es necesario que el componente administración le proporcione permisos al gestor de dependencias para realizar correctamente dichas tareas.

Una de las desventajas que presenta el *plugin* es que mediante las líneas de comando no se gestionarán las dependencias de los componentes en el fichero, las mismas se gestionarán mediante la interfaz web. Esta desventaja no solo la tiene este gestor de dependencias, aún ninguno de los gestores de dependencias existentes gestionan las dependencias mediante líneas de comando.

La CLI es un medio de interacción con un programa de ordenador en el que se emiten sucesivas líneas de texto, es decir líneas de comandos definidas ya por el desarrollador, para ejecutar las diferentes tareas que debe realizar el *plugin* como antes se mencionó en algunas de las características planteadas anteriormente.

Las dependencias también podrán ser consultadas por pantalla, brindando una mayor visualización de la información y un fácil manejo las de las dependencias de manera más amigable para el usuario.

La solución que se describe en el presente trabajo presenta una estructura de directorios mostrada en la **figura 2.**

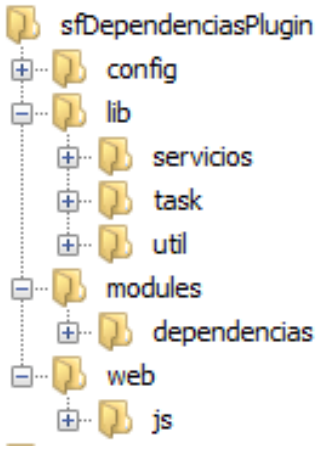


Fig. 2 Estructura del *plugin* sfDependenciasPlugin

A continuación se describen los archivos que constituyen la configuración de la solución elaborada.

Config: almacena toda la configuración del *plugin*. Contiene los siguientes archivos de configuración:

- ✓ **estructura-menu.yml:** contiene la configuración de la administración del *plugin*.
- ✓ **services.yml:** contiene los servicios definidos a utilizar por el instalador del sistema GINA cada vez que el mismo necesite la funcionalidad especificada.
- ✓ **view.yml:** contiene la configuración de las vistas.

Lib: contiene las clases utilizadas por el *plugin* organizadas por directorios.

- ✓ **Tareas:** ubicadas en el directorio *task*, se ejecutarán mediante líneas de comando utilizando el *script* del *framework* Symfony de php que se encuentra en la raíz del proyecto. Dentro de *task* se encuentra un archivo llamado *skeleton***,** el cual contiene tres archivos *.yml* que contienen la estructura que mostrarán cada uno de los componentes en el fichero *dependencias.yml***.**

- ✓ **sfUtilDependencias.php:** contiene todas las funciones a realizar por el componente.
- ✓ **serviciosDependencias.php:** almacena los servicios que serán brindados al instalador del sistema GINA.

Modules: almacena el **módulo dependencias** que define las características del *plugin*, el cual está estructurado por un directorio.

- ✓ **actions**: contiene la clase *actions.class.php* la cual almacena todas las acciones del módulo.
- ✓ **templates:** contiene la plantilla *indexSuccess.php* correspondiente a las acciones del módulo y fichero *_datosGeneralesComponentePartial.php* empleado para mostrar detalladamente las datos del gestor de dependencias.

Web: contiene un directorio *js* con los archivos de JavaScript.

La forma en la que el *framework* Symfony carga los *plugins* permite que los proyectos puedan utilizarlos como si fueran parte del propio proyecto, es por ello que este *plugin* podrá ser usado en cualquier versión del *framework* de Symfony 1.x.

2.3 Técnicas para la captura de requisitos.

Las técnicas para la captura de requisitos hacen más eficiente el proceso de recopilación de los mismos. Es una de las tareas más importantes a realizar para el desarrollo del software. En muchos casos se requiere tiempo para llegar a especificar claramente lo que el cliente espera del producto requerido, por lo que se hace necesario el empleo de técnicas que permitan establecer una buena comunicación con los interesados. (26)

A continuación se enuncia la técnica utilizada durante el proceso de desarrollo del *software* para recopilar los requisitos que darán paso al desarrollo del componente.

Se utilizó la técnica **tormenta de ideas**, donde se obtuvo un conjunto de recomendaciones y de ideas dadas por los especialistas del sistema GINA, que evitarán problemas en el desarrollo del *plugin* sfDependenciasPlugin.

2.4 Requisitos

Los requisitos de *software* son la descripción completa de lo que debe hacer el sistema, son las descripciones del comportamiento del *software* que se va a desarrollar para un buen funcionamiento del mismo.

Especificación de los requisitos funcionales para el componente Dependencias

La especificación de los requerimientos tiene como propósito establecer lo que debe hacer el sistema, que se convierte en definir los requerimientos funcionales, además de especificar los requerimientos no funcionales que son propiedades o cualidades que el componente debe cumplir.

Requisitos funcionales del componente Dependencias:

R1 Habilitar componentes instalados: el sistema debe permitir habilitar los componentes del sistema GINA de forma visual y por líneas de comando.

- R1.1 Habilitar componentes del negocio por pantalla.
- R1.2 Habilitar componentes del negocio por líneas de comando.

R2 Deshabilitar componentes instalados: el sistema debe permitir deshabilitar los componentes que sean seleccionados. Los mismos se señalaran en rojo al deshabilitarse.

- R2.1 Deshabilitar componentes del negocio por pantalla.
- R2.2 Deshabilitar componentes del negocio por líneas de comando

R3 Generar ficheros de dependencias: el sistema debe permitir generar un *fichero.yml* con las dependencias de cada componente existentes en el sistema GINA mediante líneas de comando.

- R3.1 Generar fichero de dependencia para los módulos creados, por líneas de comando.
- R3.2 Generar fichero de dependencia para los módulos nuevos, por líneas de comando.

R4 Gestionar dependencias: el sistema debe permitir gestionar las dependencias a los componentes del sistema GINA de forma visual y por líneas de comando.

R5 Mostrar dependencias del negocio: el sistema debe permitir mostrar las dependencias de cada uno de los componentes existentes en el sistema GINA de forma visual y por líneas de comando.

- R5.1 Mostrar dependencias del negocio por líneas de comando.
- R5.2 Mostrar dependencias del negocio por pantalla.

R6 Brindar estado de las dependencias haciendo uso de servicios web: el sistema debe permitir brindar el estado de las dependencias existentes en el sistema GINA haciendo uso de los servicios web.

En el **Anexo 1** se encuentran las descripciones de los requisitos antes planteados.

2.5 Técnicas para la validación de requisitos

La validación de requisitos tiene como misión demostrar que la definición de los requisitos puntualizan realmente el sistema que el usuario necesita o desea. (26)

Para lograr una correcta validación de los requisitos, se propone la **revisión técnica formal** de la documentación de los requisitos propuestos, para validar la correcta interpretación de la información que desea ser transmitida.

También se propone obtener la definición de requisitos de **prototipos de interfaz,** para comprobar si se han entendido los requisitos y permitir al usuario hacerse a la idea de la estructura de la interfaz del sistema con el usuario. Para que entienda que lo que está viendo es un prototipo y no el sistema final.

2.6 Interfaces del componente Gestión de Dependencias

A continuación se muestran los prototipos de interfaz de usuario más significativos para el sistema.

Requisito funcional R1-R2 Habilitar y Deshabilitar componentes instalados.

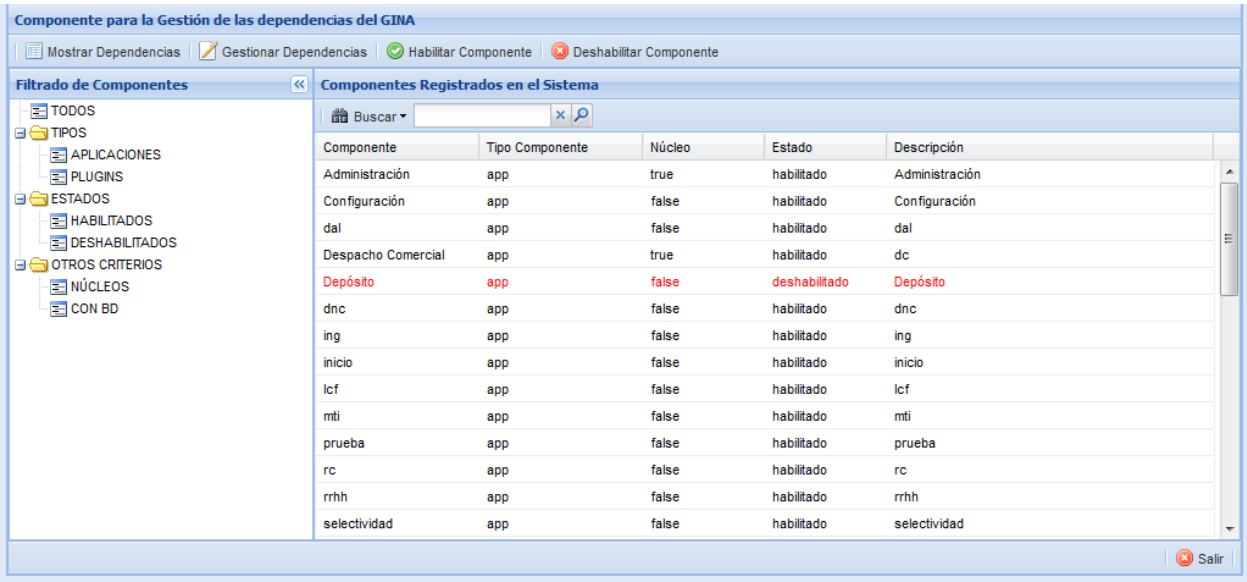


Fig. 3 Prototipo de interfaz de usuario. Habilitar y Deshabilitar componentes.

Requisito funcional R4 Gestionar dependencias.

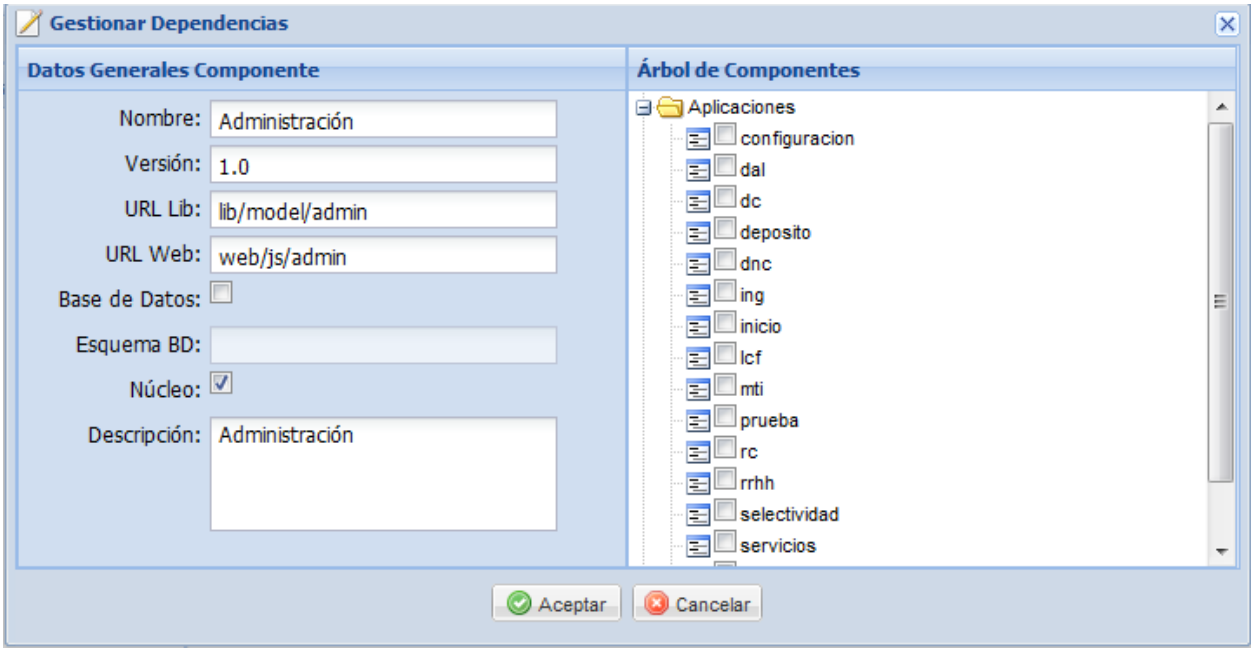


Fig. 4 Prototipo de interfaz de usuario. Gestionar dependencias.

Requisito funcional R5 Mostrar dependencias.

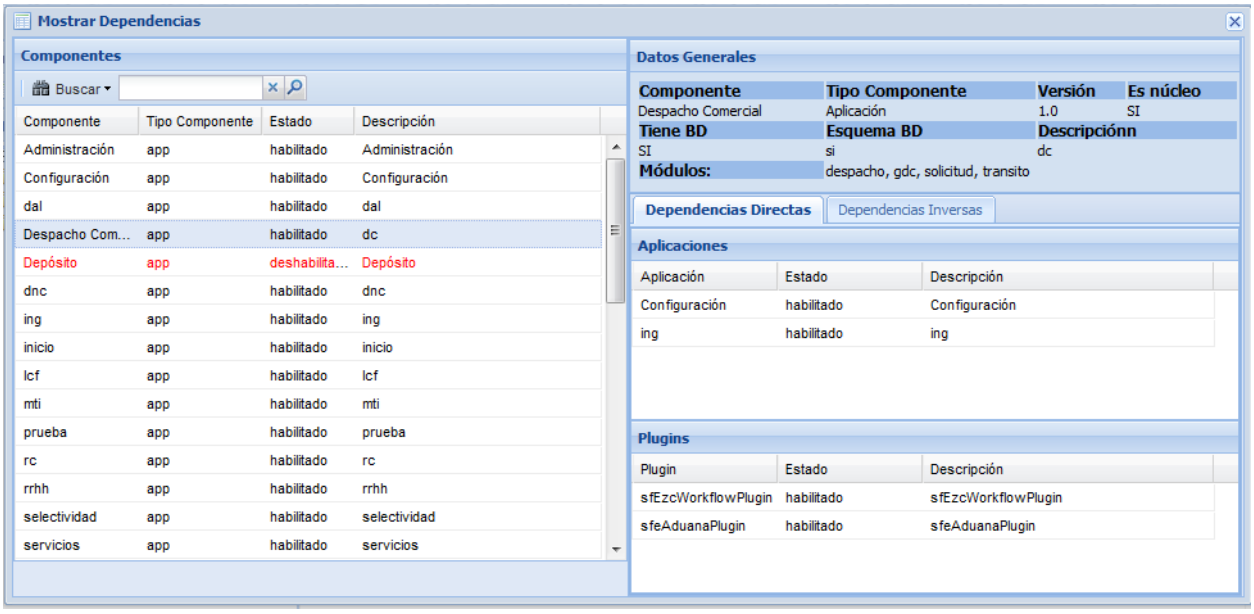


Fig. 5 Prototipo de interfaz de usuario. Mostrar Dependencias.

2.6.1 Requisitos no funcionales

Teniendo en cuenta la importancia de los requisitos no funcionales los cuales describen las características que debe tener el sistema para mejor funcionamiento, se muestra un listado de los mismos, definidos ya en el proyecto para la solución propuesta:

Usabilidad

El sistema desarrollado debe prestar facilidades de usabilidad que satisfagan las necesidades de todos los usuarios. Este contara con un menú que les permita a los usuarios acceder a las principales funciones que son de su interés; así como brindar a los usuarios la posibilidad de interactuar con el producto sin tener previa preparación, solo con los conocimientos necesarios del negocio. (27)

Fiabilidad

- ✓ **Disponibilidad:** La aplicación será capaz de estar operativa durante el mayor tiempo posible para brindar sus servicios a los usuarios ininterrumpidamente durante el tiempo que estos lo necesiten. El tiempo para la mantención de las aplicaciones no debe exceder el 10% con respecto al tiempo que la aplicación debe estar disponibles. (27)

Soporte

- ✓ La aplicación será capaz de ejecutarse sobre los sistemas operativos Windows y Linux, en un navegador Firefox 3.5 o superior.
- ✓ Sera programado en PHP 5.3.5 o superior.
- ✓ El sistema poseerá una alta seguridad. (27)

Restricciones de diseño

El subsistema realizado para el sistema GINA incluye el manejo de varios asuntos que se enlazan entre sí, por lo que constantemente se necesita acceder a información común para varios de estos procesos, es aquí donde entra la reutilización de dicha información, de forma tal que los datos no aparezcan duplicados. (27)

Interfaz

Para los requisitos de interfaz se consulta la descripción de los requisitos funcionales del subsistema donde se detallarán todos los requisitos de interfaz a continuación del prototipo de interfaz de usuario que se propone. (27)

2.7 Patrones de diseño utilizados

Un patrón de diseño es una solución estándar para un problema común de programación, una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios, una estructura de implementación que logra una finalidad determinada, manera práctica de describir aspectos de la organización de un programa. (28)

Los patrones brindan la facilidad de reutilizar el conocimiento de desarrolladores, clasificando y describiendo problemas y soluciones a problemas que surgen con frecuencia durante el desarrollo, por lo que se convierten en un historial que ayuda al equipo de desarrollo a no cometer errores ya descritos.

A continuación se muestran los patrones de diseño que se emplean en el desarrollo de la solución.

2.7.1 Patrones GRASP

Los patrones GRASP (Patrones Generales de Software para Asignar Responsabilidades) representan los principios básicos de la asignación de responsabilidades a objetos, expresados en forma de patrones.

- ✓ **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. La clase *dependencias/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. (29)

- ✓ **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. La clase *dependencias/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. (29)

- ✓ **Controlador**: se emplea la página *admin_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. (29)

2.7.2 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. (30)

- ✓ **Comando** *(Command):* este patrón permite solicitar una operación a un objeto sin conocer realmente el contenido de esta operación, ni el receptor real de la misma. Para ello se encapsula la petición como un objeto. Se observa en la clase sfWebFrontController, en el método *dispatch().* Esta clase está por defecto y es la encargada de establecer el módulo y la acción que se va a usar según la petición del usuario. (30)
- ✓ **Decorador** *(Decorator):* 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. (30)

2.7.3 Patrón Modelo Vista Controlador (MVC) en Symfony

El *framework* Symfony utiliza el patrón arquitectónico MVC el cual permite llevar a cabo una programación multicapa, separando a la interfaz del usuario, la lógica del control y los datos de la aplicación en tres componentes diferentes.

- ➢ **Modelo**: representa toda la información con la que opera la aplicación, o sea su lógica de negocio, responde a las peticiones de estado que vienen de la vista, gestiona el comportamiento y los datos del dominio y responde a instrucciones de cambio de estado provenientes del controlador. (29)

- **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. (29)
- **Controlador**: es el encargado de procesar las interacciones del usuario y ejecuta los cambios adecuados en el modelo o en la vista. La arquitectura MVC separa la lógica de negocio (el modelo) y la presentación (la vista), lo que permite un mantenimiento más sencillo de las aplicaciones. El controlador es el encargado de aislar al modelo y a la vista de los detalles del protocolo usado para las peticiones (HTTP, consola de comandos, email, etc.). (29)

En este caso la solución propuesta no utiliza base de datos, debido a que las dependencias de cada uno de los componentes que integran el sistema GINA se encuentran almacenadas en el archivo de configuración *dependencias.yml* para la gestión de las mismas.

La implementación que realiza el *framework* Symfony de la arquitectura MVC incluye varias clases como son:

- **sfController**: es la clase del controlador y se encarga de decodificar la petición y transferirla a la acción correspondiente.
- **sfRequest**: guarda todos los elementos que integran la petición (parámetros, cookies, cabeceras, etc.)
- **sfResponse**: posee las cabeceras de la respuesta y los contenidos. El contenido de este objeto se convierte en la respuesta HTML que se remite al usuario.
- El *singleton* de contexto (que se obtiene mediante **sfContext::getInstance()**)**:** guarda una referencia a todos los objetos que constituyen el núcleo del *framework* Symfony y puede ser accedido desde cualquier parte de la aplicación. (29)

En la **figura 6** se muestra la estructura de este patrón.

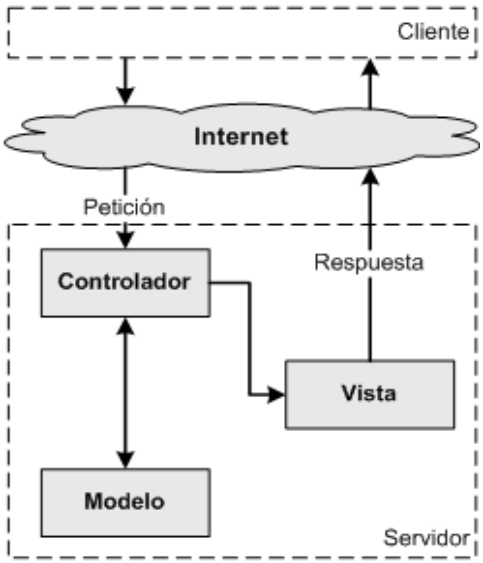


Fig.6 Patrón MVC en el *framework* Symfony.

2.8 Diseño del sistema

Durante esta fase es modelado el sistema para que soporte todos los requisitos. Esto contribuye a una arquitectura sólida y estable que se convierte en un plano para la próxima fase. Los artefactos generados en esta etapa son más formales y específicos de una implementación. En caso de llevarse a cabo la reutilización de componentes software ya desarrollados, durante esta fase se ajusta el modelado existente a los requisitos actuales. (10)

2.8.1 Diagrama de clases del diseño

Los diagramas de clases describen la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Estos son utilizados durante el proceso de análisis y diseño, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro. (28)

2.8.1.1 Diagrama de clases del componente Dependencias

En la **figura 7** se muestra el diagrama de clases del componente Dependencias para el sistema GINA. Este diagrama se compone por varias clases distribuidas en tres colores para mejor comprensión de la distribución de estas en el negocio.

La clase en color gris es la clase principal, es la contenedora de todas las funcionalidades a realizar por el componente. Las clases representadas con color amarillo muestran cada una de las tareas que se realizan por líneas de comando. La clase de color verde representa la entidad encargada del manejo de los servicios que se le ofrecerán al instalador del sistema GINA.

A continuación se muestra el diagrama de clases lógicas del componente Dependencias.

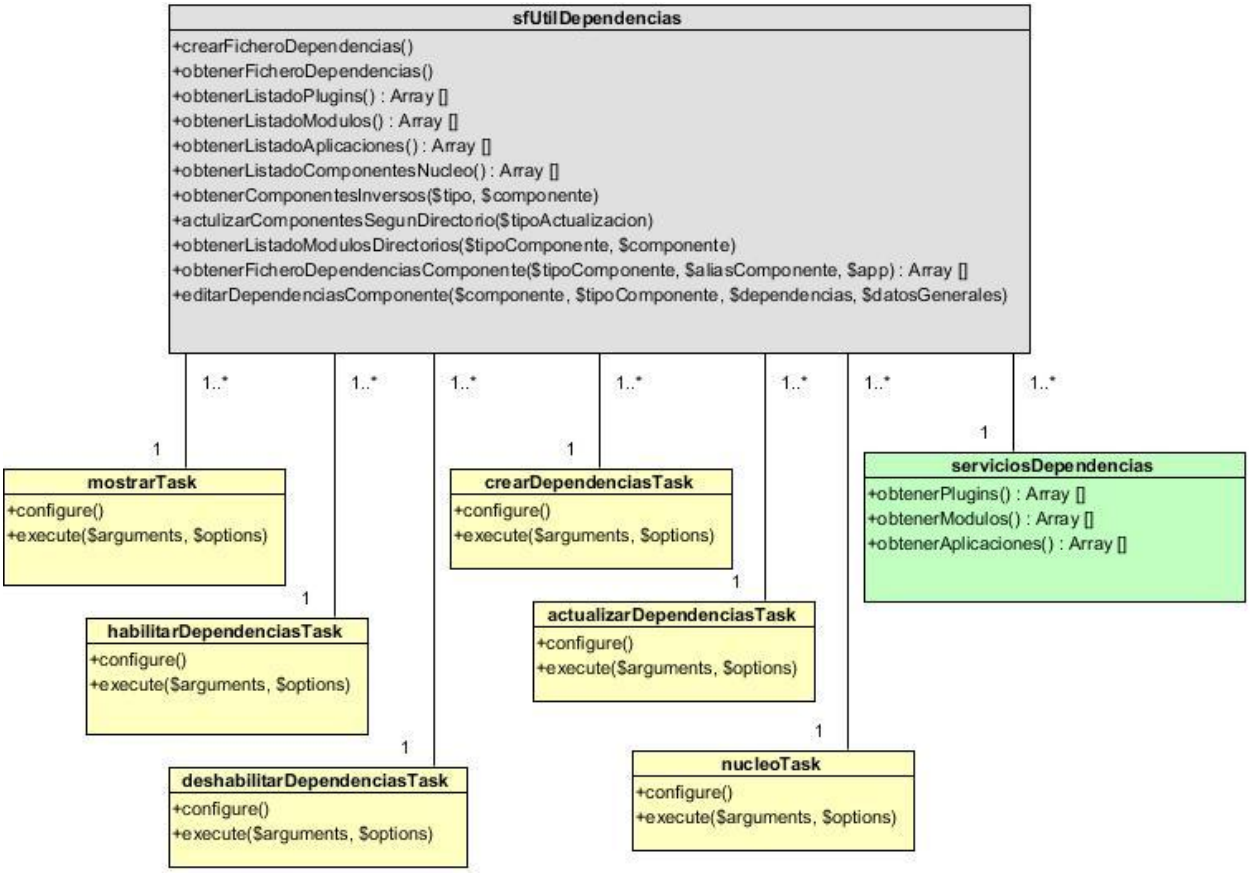


Fig.7 Diagrama de clases.

2.8.1.2 Diagrama de clases del diseño con estereotipos web

Para mejor comprensión del funcionamiento del componente Dependencias se muestra uno de los diagramas de clases del diseño con estereotipos web, perteneciente al requisito funcional **Gestionar dependencias**. En el mismo se observa la página servidora *actions.class* encargada de atender las peticiones realizadas por el controlador frontal**,** las mismas se redireccionan al *layout.php*, también conocida como plantilla global, la cual es común para todas las páginas del componente, esta carga la página *indexSuxccess.php* o *página cliente* la cual importa las interfaces de usuario que se encuentran en el paquete *InterfacesJS*.

El resto de los diagramas se encuentran en los anexos.

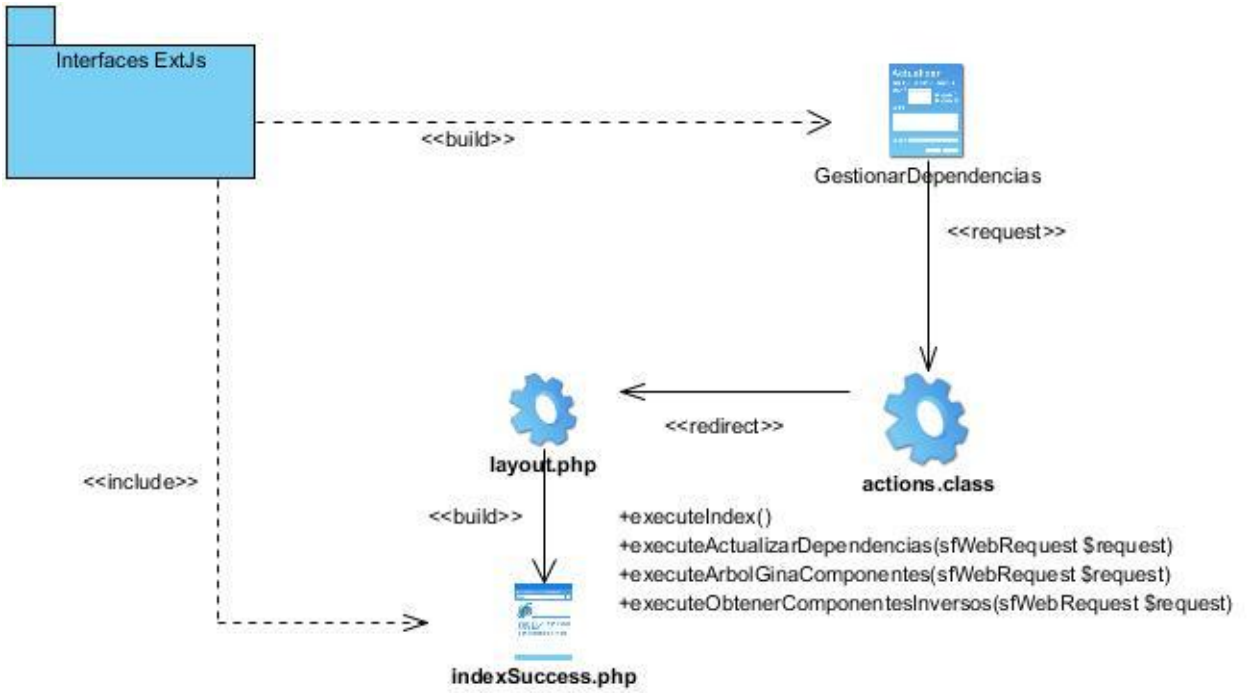


Fig.8 Gestionar dependencias.

2.8.2 Diagrama de secuencia

Los diagramas de secuencia muestran el intercambio de mensajes (es decir la forma en que se invocan) en un momento dado. Estos diagramas ponen especial énfasis en el orden y el momento en que se envían los mensajes a los objetos. (28)

En este epígrafe se muestra el diagrama de secuencia de uno de los requisitos más significativos, **Gestionar dependencias**. Para mejor comprensión se realiza una breve descripción del diagrama.

Al observar la **figura 9** se puede visualizar el inicio del proceso al seleccionar la opción *gestionar dependencias*, del menú *Gestionar Dependencias*, enviando la petición realizada a la clase *action.class* la cual es la encargada de controlar las peticiones realizadas haciendo llamadas a la funcionalidad *executeActualizarDependencias(sfWebRequest $request)* y devolviendo las dependencias actualizadas.

Requisito Funcional R3.1 Gestionar dependencias.

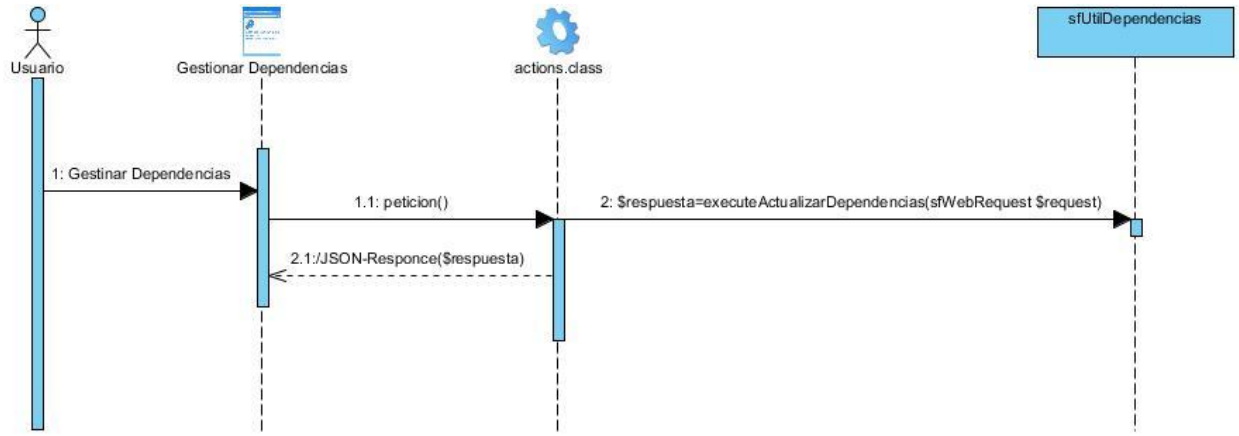


Fig.9 Diagrama de secuencia para el requisito funcional Gestionar dependencias**.**

2.8.3 Diagrama de paquetes

El diagrama de paquetes que se muestra a continuación en la **figura 10**, se encuentra compuesto por un paquete principal denominado *sfDependenciasPlugin* el cual se encarga de gestionar todo lo referente a la gestión de las dependencias de cada uno de los componentes que se encuentran dentro del sistema GINA. El componente hace uso del *plugin sfUtilPlugin* para posibilitar la interacción a través de servicios web con los subsistemas que integran al sistema GINA.

Funcionando de forma externa se encuentran las librerías del *framework* Symfony ubicadas en el paquete *Symfony* con el cual interactúan todos los compontes del sistema GINA.

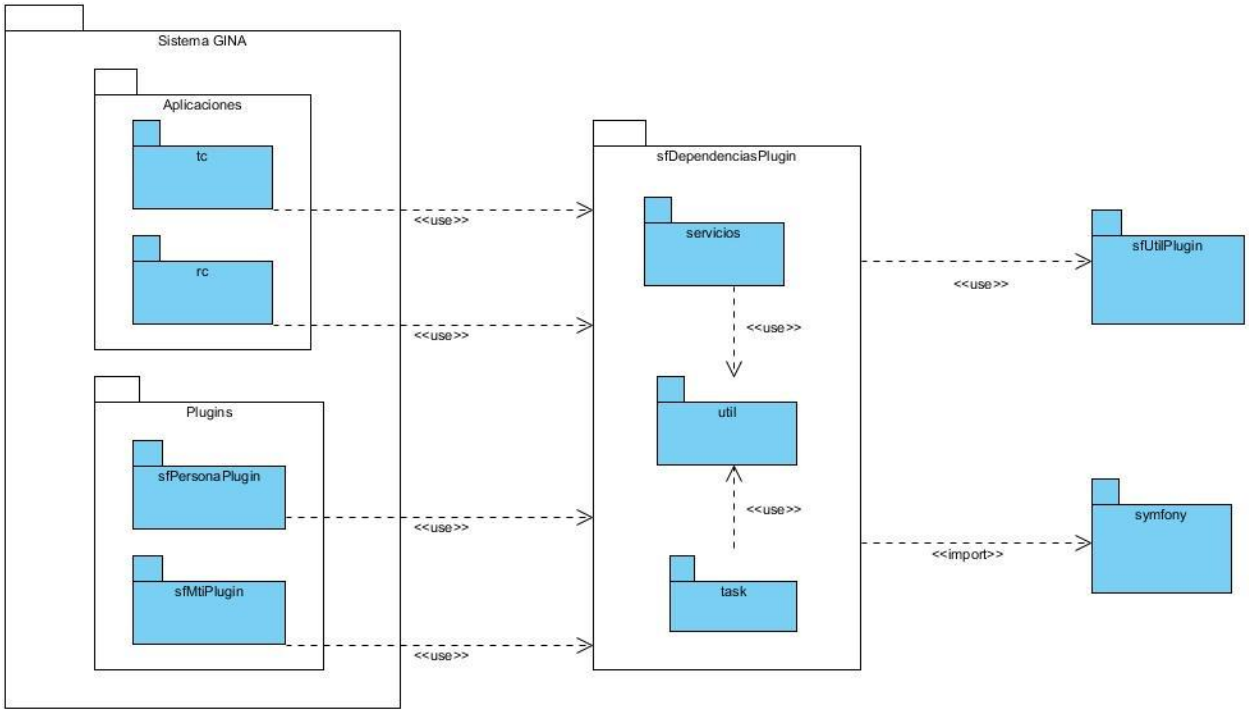


Fig.11 Diagrama de Paquetes.

Todas las aplicaciones y *plugins* que se encuentren en el mismo proyecto utilizarán las funcionalidades del componente de dependencias sfDependenciasPlugin.

2.9 Diagrama de componentes

Un diagrama de componentes representa cómo un sistema de *software* es dividido en componentes y muestra las dependencias entre los mismos. Los componentes físicos incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables, o paquetes. Los diagramas de componentes prevalecen en el campo de la arquitectura de *software* pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema. (31)

En la **figura 12** se muestra el diagrama de componentes que representa las relaciones del Gestor de Dependencias con otros componentes del sistema GINA, dígase el Instalador y la *cliSymfony*, a los cuales le brinda servicios mediante la interfaz *serviciosDependencias*, además de los servicios de seguridad que le ofrece el componente *administración*.

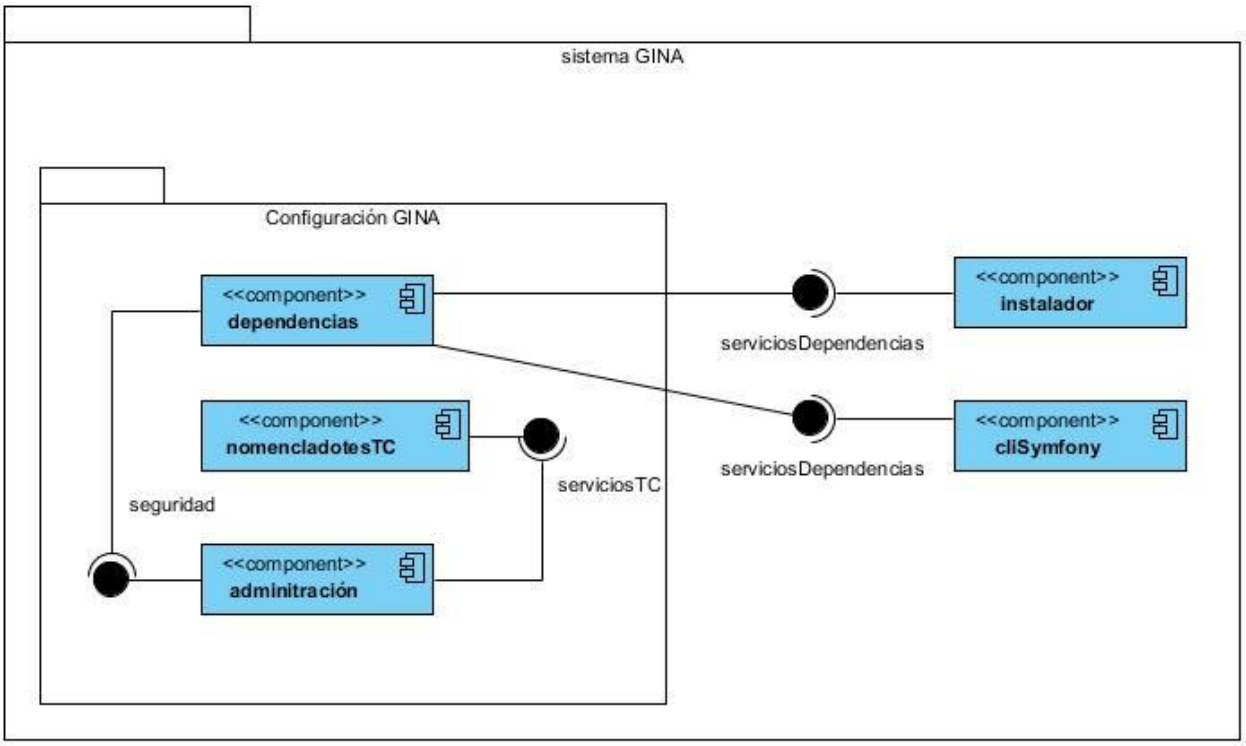


Fig.12 Diagrama de componentes.

2.10 Métricas para la evaluación del diseño

Las métricas de evaluación del diseño ayudan a medir la calidad de los atributos internos del software. Lo que permite evaluar los modelos de análisis y de diseño, proporcionando una indicación de la complejidad de diseños técnicos y de código fuente, ayudando a la calidad durante el desarrollo del sistema y en el diseño de pruebas más efectivas. (32)

Métricas de Relaciones entre clases (RC)

Está dado por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad:

- **Acoplamiento:** un aumento del RC implica un aumento del acoplamiento de la clase.
- **Complejidad de mantenimiento:** Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
- **Reutilización:** un aumento del RC implica una disminución en el grado de reutilización de la clase.
- **Cantidad de pruebas:** un aumento del RC implica un aumento de la cantidad de pruebas de unidad necesarias para probar una clase. (33)

A continuación se muestran los resultados obtenidos de los atributos de calidad.

Resultados obtenidos:

Acoplamiento: Bajo

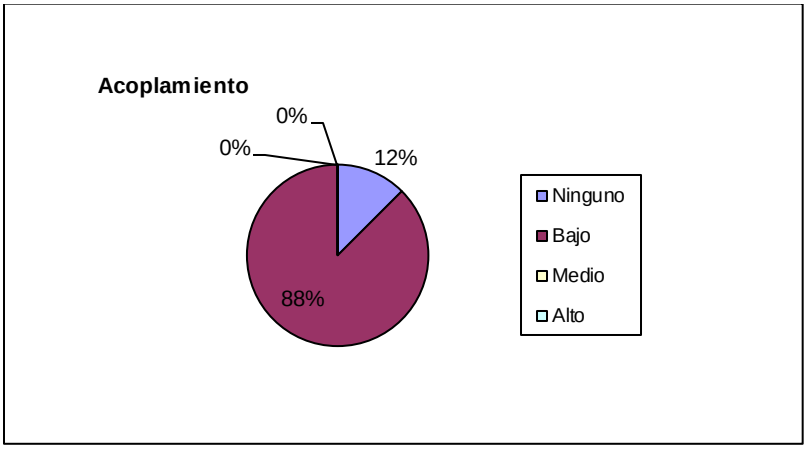


Fig.13 Resultados de la evaluación de la métrica RC en el atributo acoplamiento.

Complejidad de mantenimiento: Baja

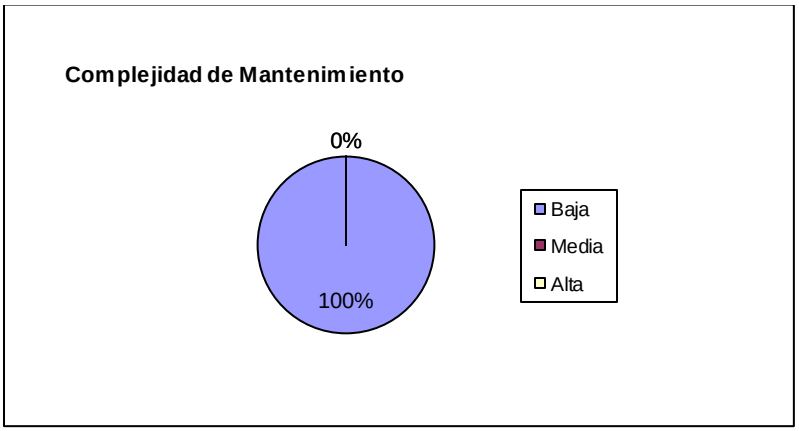


Fig.14 Resultados de la evaluación de la métrica RC en el atributo complejidad de mantenimiento.

Cantidad de pruebas: Baja

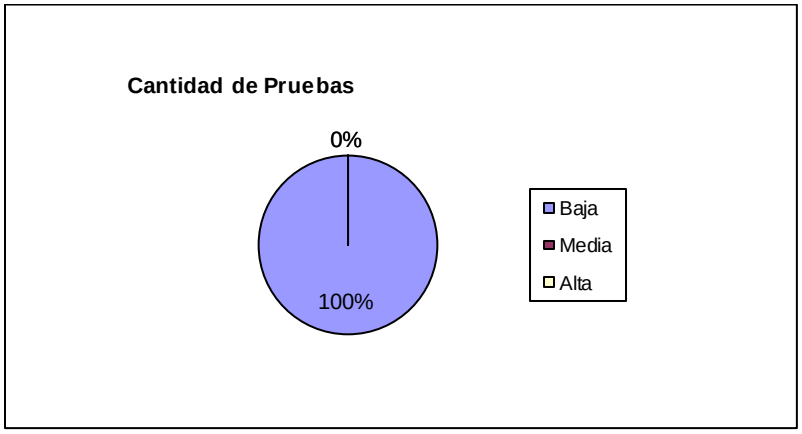


Fig.15 Resultados de la evaluación de la métrica RC en el atributo cantidad de pruebas.

Reutilización: Alta

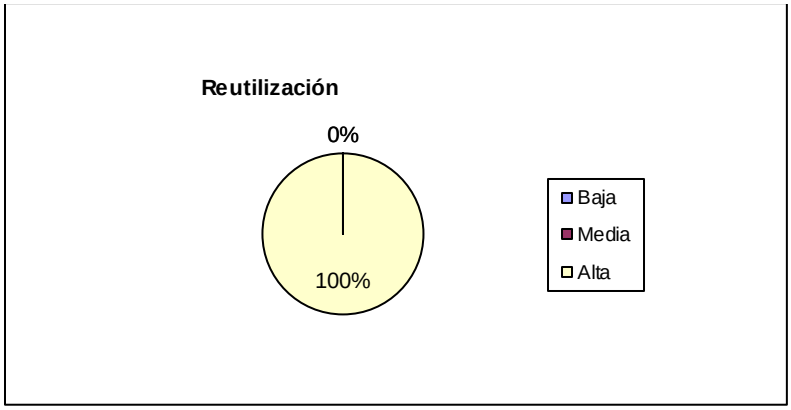


Fig.16 Resultados de la evaluación de la métrica RC en el atributo reutilización.

Los resultados obtenidos luego de aplicar la medición, demuestran que las clases del diseño están entre los límites aceptables de calidad, teniendo en cuenta que el 100% de las clases se encuentran en la categoría baja, lo que demuestra la elevada reutilización, bajo acoplamiento, complejidad y cantidad de pruebas en el diseño propuesto.

Métricas propuestas por Lorenz y Kidd:

Lorenz y Kidd propusieron las métricas de diseño para medir las características estáticas de un producto software, estas están basadas en tres categorías: métricas de tamaño, métricas de herencia, métricas de las características internas de las clases.

- **Métricas de tamaño**: miden la cantidad de responsabilidad que tiene una clase, es una medida de la cantidad de colaboración utilizada.

- **Métricas de herencia**: es el número total de métodos que se definen en una subclase, mide la calidad de uso de la herencia.
- **Métricas de características internas de una clase:** miden las propiedades intrínsecas del software como el tamaño y la complejidad. (33)

Estas métricas están enfocadas a las características internas del diseño orientado a objetos con medidas objetivas.

Tamaño operacional de clase TOC

Está dado por el número de los métodos asignados a una clase y evalúa los siguientes atributos de calidad:

- **Responsabilidad:** un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
- **Complejidad de implementación:** un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
- **Reutilización:** un aumento del TOC implica una disminución del grado de reutilización de la clase.

Si existen valores grandes de TOC, éstos mostrarán que una clase puede tener demasiadas responsabilidades, lo cual reducirá la reusabilidad de la clase, y complicará la implementación y la comprobación. Por otra parte cuanto menor sea el valor medio para el tamaño, más probable es que las clases existentes dentro del sistema se puedan reutilizar ampliamente. (33)

Resultados obtenidos:

Responsabilidad: Baja

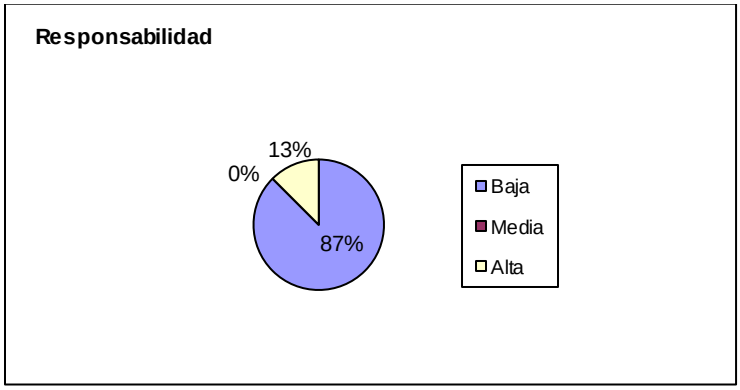


Fig.17 Resultados de la evaluación de la responsabilidad.

Complejidad: Baja

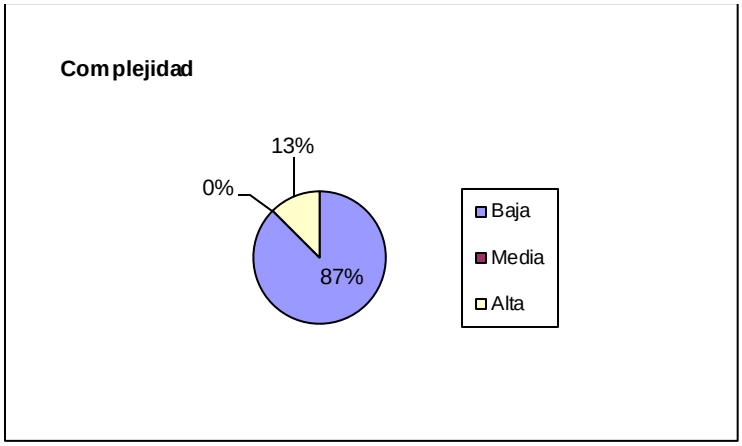


Fig.18 Resultados de la evaluación de la complejidad.

Reutilización: Alta

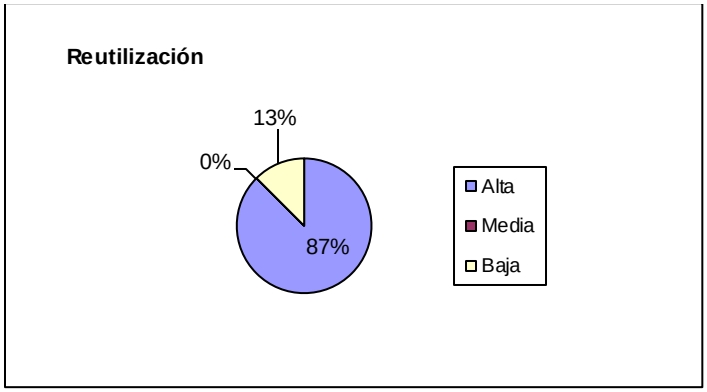


Fig.19 Resultados de la evaluación de la reutilización.

Después de mostrados los datos en las gráficas anteriores se concluye que la complejidad y la responsabilidad del diseño realizado son bajas, por lo que no se presentará problemas en la implementación y las clases dentro del sistema se podrán reutilizar ampliamente.

2.11 Conclusiones parciales

En el presente capítulo se llevó a cabo la etapa de análisis y diseño la cual posibilitó la obtención de los principales artefactos que serán la base para el desarrollo del componente a desarrollar. Se mostraron las técnicas de captura y validación de los requisitos propuestos para un correcto desarrollo del componente a realizar. Para la evaluación del diseño se aplicó la métrica TOC, donde se obtuvo como resultado que las clases del componente Dependencias podrán reutilizarse ampliamente.

Capítulo 3. Implementación y Pruebas del Sistema

3.1 Introducción

El presente capítulo será el encargado de dar una visión de la implementación del sistema, profundizando en el código obtenido para facilitar el entendimiento de este. En el mismo se describen los artefactos que se generan durante esta fase. Mostrará los resultados y estándares de codificación empleados durante la implementación.

3.2 Implementación del sistema

El propósito principal de la implementación es desarrollar la arquitectura y el sistema como un todo, de forma más específica, los propósitos de la implementación son:

- ✓ Planificar las integraciones del sistema necesarias en cada iteración, siguiendo para ello un enfoque incremental, lo que da lugar a un sistema que se implementa en una sucesión de pasos pequeños y manejables.
- ✓ Implementar las clases y subsistemas encontrados durante el diseño. En particular las clases se implementan como componentes de fichero que contienen el código fuente.
- ✓ Probar los componentes individualmente, y a continuación integrarlos compilándolos y enlazándolos en uno o más ejecutables, antes de ser enviados para ser integrados y llevar a cabo las comprobaciones del sistema. (31)

3.2.1 Estándares de codificación

Los estándares de codificación son una guía para el desarrollo del sistema y sirven para regir el código de la implementación del mismo. (34)

El objetivo fundamental de los estándares de codificación es estructurar el código fuente de forma común en todo el sistema. Deben servir para identificar de forma sencilla cuál es el objetivo y las funcionalidades que brinda cada una de las clases y funciones, es necesario que esto se pueda identificar a simple vista. Además debe servir de guía para los implementadores que tienen que continuar el desarrollo de las aplicaciones.

Para definir el estilo de codificación a seguir en la aplicación se utilizó la notación estándar establecida para aplicaciones desarrolladas en el lenguaje PHP.

A continuación se muestran las principales reglas de codificación utilizadas en el desarrollo de la solución bajo la tecnología y el lenguaje PHP y la arquitectura regida por la utilización del *framework* Symfony.

Cada una de las reglas fue obtenida del documento “Propuesta de un Estándar de Codificación” del departamento de Soluciones Aduaneras del CEIGE, el cual se encuentra en el repositorio del proyecto Aduana.

3.2.1.1 Reglas generales

Para la apertura y cierre de las etiquetas del lenguaje PHP será utilizada la siguiente notación:

```
<?php
//Código
?>
```

Los encabezados en las funciones y acciones deberán contener el requisito a que da solución, los parámetros que necesita así como la respuesta esperada:

```
/**
* Funcionalidad para obtener el fichero de las dependencias dado un componente
* @params String TipoComponente [plugin, app, module]
* @return Array Fichero convertido en arreglo
**/
```

3.2.1.2 Reglas en las acciones

- Dentro de las especificaciones del *framework* Symfony está que cada una de las acciones debe comenzar con la palabra *execute*.
- Todos los nombres de acciones deben estar en la nomenclatura CamelCase comenzando por la palabra *execute* como son: *executeMostrarDependencias, executeCrearDependencias*, etc.
- Los nombres de las acciones deben especificar, con la menor cantidad de palabras, cuál es el objetivo de la acción.

3.2.2 Tratamiento de errores

El control de posibles errores a ocurrir en un determinado sistema, garantiza el correcto funcionamiento del mismo, por lo que se hace necesario identificar y controlar las posibles vías por las que pudiesen ocurrir estos problemas.

El sistema en desarrollo presenta varias características que rigen el comportamiento de tratamiento de errores, una de las mismas la constituye el hecho de que hace un uso intensivo del tratamiento de excepciones otorgado por el lenguaje de programación usado (PHP), el cual brinda la posibilidad de capturar excepciones ocurridas por entrada de datos erróneos u ocurrencias de errores en tiempo de ejecución. Un correcto uso de esta estructura permite manejar de manera conveniente los errores que ocurran, además de impedir que el sistema

deje de funcionar en algún momento, aspecto que contribuye de forma favorable a la disponibilidad del mismo.

Los errores ocurridos son registrados en los archivos log que presenta el *framework* Symfony haciendo uso de una estructura otorgada por este, la cual hace el manejo de estos registros una tarea sencilla, de esta forma quedan registrados para un futuro análisis los eventos ocurridos.

3.2.3 Comunicación entre las capas

El *framework* Symfony hace uso del patrón arquitectónico MVC, el cual interactúa con las capas del negocio permitiendo mayor organización del sistema además de incrementar su rendimiento.

En el siguiente epígrafe se ejemplifica el proceso de gestión de dependencias de los componentes del sistema GINA para el requisito funcional **Gestionar dependencias por pantalla**.

3.2.3.1 Ejemplo: Gestionar Dependencias

El proceso comienza a partir de la entrada al componente *Gestión de Dependencias*, el cual muestra en su interfaz principal un listado de componentes de los cuales se seleccionará el componente deseado a gestionar dependencias.

En la **figura 20** se muestra la interfaz principal, en la misma se observan los componentes reales, existentes en el sistema GINA para la gestión de sus dependencias.

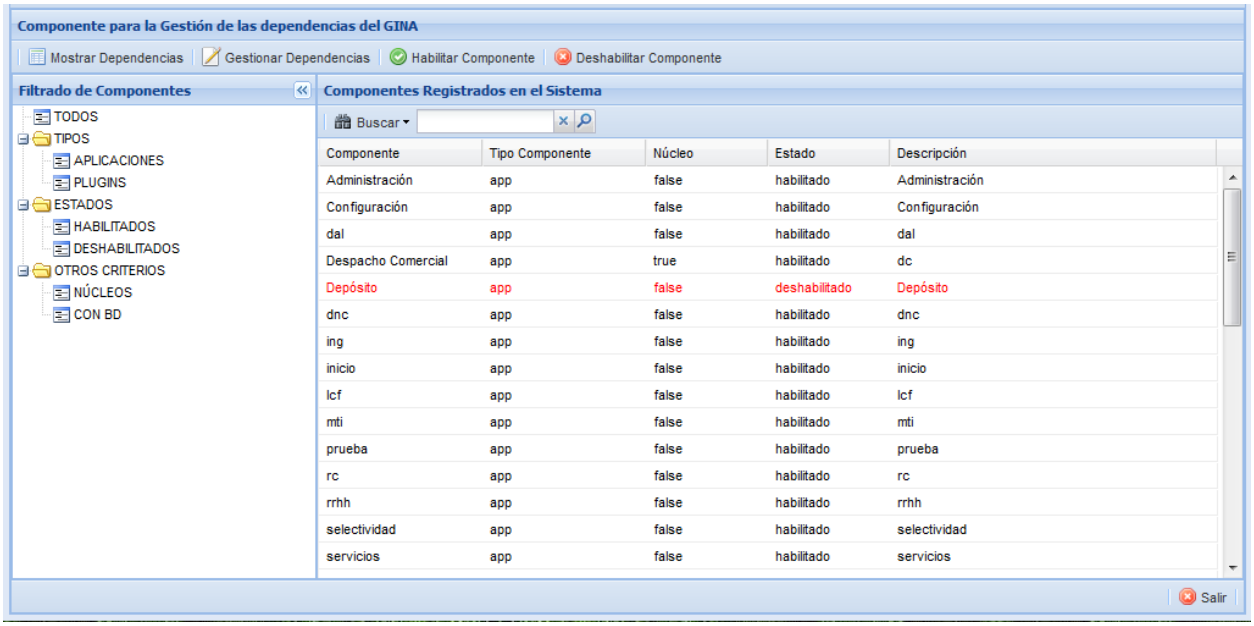


Fig.20 Componente para la gestión de las dependencias del sistema GINA.

Para acceder a gestionar las dependencias de uno de los componentes del listado mostrado en la figura anterior se selecciona un componente y posteriormente se ejecutará el botón **Gestionar Dependencias** cuyo código se muestra a continuación.

```
//web/sfDependenciasPlugin/js/mainDependencias.js
    }
text: 'Gestionar Dependencias',
xtype: 'button',
tooltip: 'Gestiona las dependencias del sistema',
iconCls: 'iconAdd',
handler: function() {
    if (Ext.getCmp('gridComponentesHome').getSelectionModel().hasSelection())
        dependencias = new GestionarDependencias().show();
    else {
        Ext.MessageBox.show({
            title: 'Informaci&oacute;n',
            msg: "Para gestionar las dependencias de un componente primeramente debe
seleccionarlo del listado.",
            buttons: Ext.Msg.OK,
            icon: Ext.MessageBox.INFO,
            width: 400
        });
    }
}
}
```

La ejecución del código anterior envía el identificador del componente al cual se le solicitó gestionar las dependencias hacia la clase de JavaScript *gestionarDependencias.js* y esta a su vez ejecuta cuatro funcionalidades implementadas en la clase controladora *actions.class.php*: *executeObtenerPantallaDependencias(sfWebRequest $request),* encargada de mostrar **"Datos Generales"** del componente en cuestión y el **"Árbol de Componentes"** mediante las

funcionalidades *executeObtenerDatosGeneralesComponente (sfWebRequest $request)*, la cual captura los datos del componente seleccionado, estos datos pueden ser modificados por el usuario, la funcionalidad *executeArbolGinaComponentes(sfWebRequest $request)* se encarga de listar los componentes del árbol, captura los componentes seleccionados, y executeActualizarDependencias(sfWebRequest $request) encargada de actualizar las dependencias del componente.

El fragmento de código que se muestra a continuación es el encargado de mostrar los datos generales de la interfaz *Gestionar Dependencias*.

```
// plugins/sfDependenciasPlugin/modules/dependencias/actions/action.class.php

public function executeObtenerDatosGeneralesComponente(sfWebRequest $request) {
        try {
            $componente = $this->getRequestParameter('componente');
            $tipo = $this->getRequestParameter('tipo');

            $fichero = sfUtilDependencias::obtenerFicheroDependenciasComponente($tipo, $componente, null);

            return $this->renderPartial('datosGeneralesComponentePartial', array(
                        'nombre' => $fichero['nombre'],
                        'tipo' => $fichero['tipo'],
                        'descripcion' => $fichero['descripcion'],
                        'nucleo' => $fichero['nucleo'],
                        'version' => $fichero['version'],
                        'baseDatos' => $fichero['base-datos'],
            ));
        } catch (Exception $exc) {
            return $this->renderText("{\"success\": false, \"msg\": '" . $ex->getMessage() . "'}");
        }
    }
```

El fragmento de código que se muestra a continuación es el encargado de mostrar el árbol de componentes de la interfaz *Gestionar Dependencias*.

```
// plugins/sfDependenciasPlugin/modules/dependencias/actions/action.class.php

public function executeArbolGinaY(sfWebRequest $request) {
        $nodo = $this->getRequestParameter('node');
        $result = array();

        if ($nodo == 'node') {
            $result[] = array(
                'text' => 'Aplicaciones',
                'id' => 'apps',
                'leaf' => false,
                'tipo' => 'root_apps'
            );
            $result[] = array(
                'text' => 'Plugins',
                'id' => 'plugins',
                'leaf' => false,
                'tipo' => 'root_plugins'
            );
        } else {
            if ($nodo === 'apps') {

                $ruta = sfConfig::get('sf_apps_dir');
```

```
            $dh = opendir($ruta);

            while (($file = readdir($dh)) !== false) {
                if (is_dir($ruta . '/' . $file) && $file != "." && $file != ".." && $file != ".svn") {
                    $result[] = array(
                        'text' => $file,
                        'id' => 'apps/' . $file,
                        'leaf' => false,
                        'tipo' => 'app',
                        'checked' => false
                    );
                }
            }
        } elseif ($nodo === 'plugins') {

            $ruta = sfConfig::get('sf_plugins_dir');
            $dh = opendir($ruta);

            while (($file = readdir($dh)) !== false) {

                if (is_dir($ruta . '/' . $file) && substr($file, -6) === 'Plugin')  {
                    //solo si el archivo es un directorio, distinto que "." y ".."
                    $result[] = array(
                        'text' => $file,
                        'id' => 'plugins/' . $file,
                        'leaf' => true,
                        'tipo' => 'plugins',
                        'checked' => false
                    );
                }
            }
        } else {
            list($prefix, $app) = explode('/', $nodo);

            $ruta = sfConfig::get('sf_apps_dir');
            $ruta = $ruta . '/' . $app . '/modules/';
            $dh = opendir($ruta);

            while (($file = readdir($dh)) !== false) {

                if (is_dir($ruta . '/' . $file) && $file != "." && $file != ".." && $file != ".svn") {
                    //solo si el archivo es un directorio, distinto que "." y ".."
                    $result[] = array(
                        'text' => $file,
                        'id' => 'apps/' . $app . '/modules/' . $file,
                        'leaf' => true,
                        'tipo' => 'module',
                        'checked' => false
                    );
                }
            }
        }
    }
    return $this->renderText(json_encode($result));
}
```

El fragmento de código que se muestra a continuación actualiza las dependencias y los datos generales mencionados anteriormente.

```
// plugins/sfDependenciasPlugin/modules/dependencias/actions/action.class.php
public function executeActualizarDependencias(sfWebRequest $request){
    try {
        $dependencias = $this->getRequestParameter('dependencias');
        $componente = $this->getRequestParameter('idComponente');
        $tipoComponente = $this->getRequestParameter('tipoComponente');
        $datosGenerales = array(
            'nombre' => $this->getRequestParameter('nombre'),
            'version' => $this->getRequestParameter('version'),
```

```
            'descripcion' => $this->getRequestParameter('descripcion'),
        );
        $result=sfUtilDependencias::editarDependenciasComponente($componente,$tipoComponente,
        json_decode($dependencias), $datosGenerales);
          if ($result)
            return $this->renderText('{"success": true, msg:"Las dependencias han sido actualizadas correctamente"}');
    } catch (Exception $ex) {
        return $this->renderText('{"success": false, msg:"' . $ex->getMessage() . '"}');
    }
}
```

A continuación se muestra la interfaz donde se gestionan las dependencias de los componentes del sistema GINA.

Esta interfaz permite introducir datos generales del componente en cuestión, además de seleccionar las dependencias de este con aplicaciones y *plugins*.

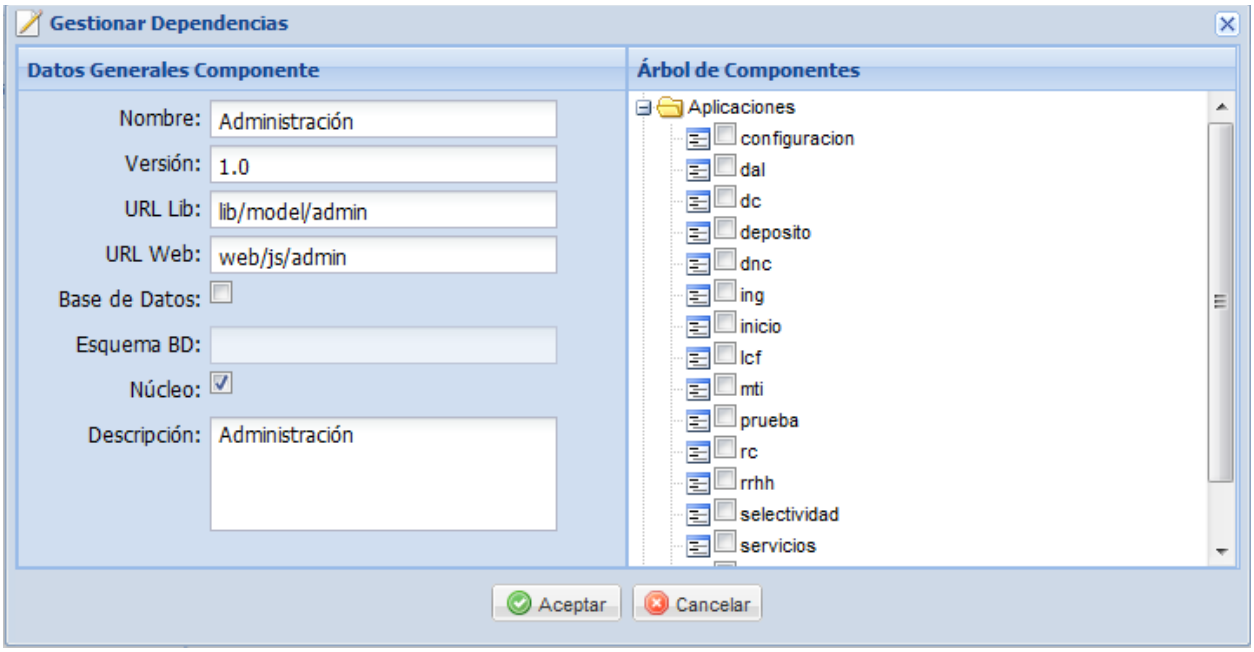


Fig.21 Interfaz para Gestionar Dependencias.

3.3 Validación de la solución

Las pruebas constituyen el único instrumento adecuado para determinar el estado de la calidad de un producto de *software*. En este proceso se ejecutan pruebas dirigidas a componentes del *software* o al sistema de *software* en su totalidad, con el objetivo de medir el grado en que este cumple con los requerimientos.

En este capítulo se mostrará el proceso de pruebas realizado al componente Dependencias, argumentando inicialmente en las diferentes técnicas de pruebas de software que existen.

3.3.1 Técnicas de evaluación de software

La calidad de un sistema está determinada, entre otras cosas, por la coincidencia entre lo que se concibió y los requisitos establecidos en la primera fase. Para comprobar el grado de cumplimiento de estos requisitos se usan dos tipos de pruebas de software las cuales se describen a continuación:

Técnicas de Evaluación Estáticas: buscan faltas sobre el sistema en reposo. Estudian los distintos modelos que componen el sistema de software buscando posibles faltas en los mismos. Así pues, estas técnicas se pueden aplicar, tanto a requisitos como a modelos de análisis, diseño y código. (35)

Técnicas de Evaluación Dinámicas: generan entradas al sistema con el objetivo de detectar fallos cuando el sistema ejecuta dichas entradas. Los fallos se observan cuando se detectan incongruencias entre la salida esperada y la salida real. La aplicación de técnicas dinámicas es también conocida como pruebas de *software* o *testing* y se aplican generalmente sobre código. (35)

Como se indicó anteriormente, las técnicas de evaluación dinámica proporcionan distintos criterios para generar casos de prueba que provoquen fallos en los programas. Estas técnicas se agrupan en:

Técnicas de caja blanca o estructurales: se basan en un minucioso examen de los detalles procedimentales del código a evaluar, por lo que es necesario conocer la lógica del programa.

Técnicas de caja negra o funcionales: realizan pruebas sobre la interfaz del programa a probar, entendiendo por interfaz las entradas y salidas de dicho programa. No es necesario conocer la lógica del programa, únicamente la funcionalidad que debe realizar. (35)

3.3.1.1 Estrategias de pruebas de Caja Negra aplicadas

Las estrategias de pruebas de caja negra consisten en pasos a seguir a la hora de evaluar dinámicamente un sistema de *software*, comenzando por los componentes más simples y más pequeños e ir avanzando progresivamente hasta probar todo el *software* en su conjunto. (35)

3.3.1.2 Pruebas Funcionales

Las pruebas funcionales constituyen una estrategia de evaluación de *software*. 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. (35)

Estas pruebas funcionales están basadas en las estrategias de pruebas de caja negra, las cuales consisten en pasos a seguir a la hora de evaluar dinámicamente un sistema de *software*, comenzando por los componentes más simples y más pequeños e ir avanzando progresivamente hasta probar todo el *software* en su conjunto.

Aplicación de pruebas funcionales al sistema

A continuación se muestra el diseño de caso de pruebas para el **RF Gestionar dependencias por pantalla**. Se encuentra distribuido en cinco escenarios de pruebas, de forma que las funcionalidades sean probadas de forma eficiente.

Escenario	Descripción	Nombre	Versión	ERL Lib	ERL Web	Esquema BD	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Gestionar Dependencias	Se gestiona correctamente las dependencias del componente.	V dato	V dato	V dato	V dato	V dato	V dato	Se actualizan las dependencias correctamente y se muestra un mensaje indicando este resultado.	1. Clic en el botón "Gestión de Dependencias" 2. Seleccionar uno de los componentes registrados en el sistema 3. Clic en el botón "Gestionar Dependencias". 4. Introducir datos generales del componente. 5. Seleccionar dependencias en el "Árbol de Componentes". 6. Clic en el botón "Aceptar"
		V dato	I dato	V dato	V dato	V dato	V dato		
		V dato	V dato	I dato	V dato	V dato	V dato		
		V dato	V dato	V dato	I dato	V dato	V dato		
		V dato	V dato	V dato	V dato	I dato	V dato		
		V dato	V dato	V dato	V dato	V dato	I dato		
EC 1.2 Gestionar Dependencias con Datos Generales Incompletos	Se realiza este escenario en el caso que se deje de insertar un dato necesario		V dato	V dato	V dato	V dato	V dato	Muestra un mensaje :Error en el campo señalado en “rojo”	1. Clic en el botón "Gestión de Dependencias" 2. Seleccionar uno de los componentes registrados en el sistema 3. Clic en el botón "Gestionar Dependencias". 4. Introducir datos generales del componente. 5. Seleccionar dependencias en el "Árbol de Componentes". 6. Clic en el botón "Aceptar"
		V dato	V dato	V dato	V dato		V dato		
		N/A	N/A	N/A	N/A	N/A	N/A		
EC 1.3 Gestionar Dependencias con Datos Generales incorrectos	Se realiza este escenario en el caso que se inserte algún dato incorrecto	I dato	V dato	V dato	V dato	V dato	V dato	Señala los campos vacíos en rojo y no permite ejecutar la actualización de las dependencias.	1. Clic en el botón "Gestión de Dependencias" 2. Seleccionar uno de los componentes registrados en el sistema 3. Clic en el botón "Gestionar Dependencias". 4. Introducir datos generales del componente. 5. Seleccionar dependencias en el "Árbol de Componentes". 6. Clic en el botón "Aceptar"
		V dato	I dato	V dato	V dato	V dato	V dato		
		V dato	V dato	V dato	I dato	V dato	V dato		

		V dato V dato	V dato V dato	V dato V dato	V dato V dato	I dato V dato	V dato I dato		
EC 1.4 Gestionar Dependencias sin introducir Datos Generales	Se realiza este escenario en el caso que no se inserte algún dato	N/A	N/A	N/A	N/A	N/A	N/A	Señala los campos vacíos en rojo y no permite ejecutar la actualización de las dependencias.	1. Clic en el botón "Gestión de Dependencias" 2. Seleccionar uno de los componentes registrados en el sistema 3. Clic en el botón "Gestionar Dependencias". 4. Introducir datos generales del componente. 5. Seleccionar dependencias en el "Árbol de Componentes". 6. Clic en el botón "Aceptar"
EC 1.5 Gestionar Dependencias sin seleccionar componente.	Al intentar gestionar las dependencias de un componente sin ser seleccionado, resulta que no se puede si no está seleccionado.	N/A	N/A	N/A	N/A	N/A	N/A	Muestra un mensaje de información para gestionar las dependencias de un componente primeramente debe seleccionarlo del listado.	1. Clic en el botón "Gestión de Dependencias". 2. Clic en el botón "Gestionar Dependencias".

Tabla 2 Diseño de casos de prueba para el RF Gestionar Dependencias por Pantalla.

Los escenarios de pruebas descritos anteriormente fueron probados con los juegos de datos que se muestran en la **tabla 3**.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Nombre	Campo de texto	No	Caracteres [A..Z, a..z,]
2	Descripción	Campo de texto	Sí	Caracteres [A..Z, a..z,]
3	Verción	Campo de texto	Sí	Caracteres numéricos [0..9]
4	URL Lib	Campo de texto	Sí	Caracteres especiales [_, -, /], Caracteres alfanuméricos [A..Z, a..z, 0..9]
5	URL Web	Campo de texto	Sí	Caracteres especiales [_, -, /], Caracteres alfanuméricos [A..Z, a..z, 0..9]
6	Esquema BD	Campo de texto	No	Caracteres [A..Z, a..z,]

Tabla 3 Juego de datos para el RF Gestionar dependencias por pantalla.

Las pruebas funcionales realizadas al sistema fueron exitosas, no se detectaron no conformidades. Por lo que el sistema se encuentra funcionando correctamente.

A continuación se muestra la **figura 22** con los resultados de las pruebas funcionales.

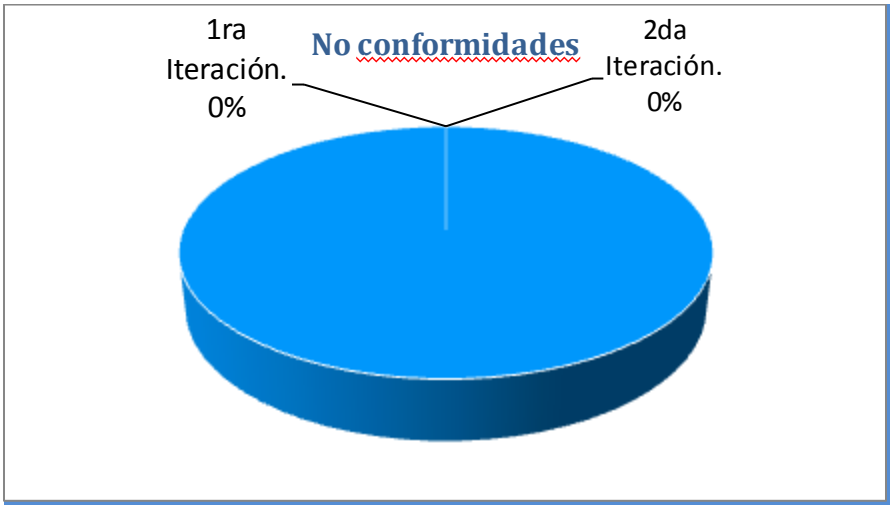


Fig.22 Resultados de las pruebas funcionales**.**

En el **Anexo 4** se encuentran los Casos de Pruebas de los restantes requisitos.

3.3.1.3 Pruebas Unitarias

Las pruebas unitarias aseguran que un único componente de la aplicación produce una salida correcta para una determinada entrada. Este tipo de pruebas son una estrategia de prueba de caja negra, validan la forma en la que las funciones y métodos trabajan en cada caso particular. Son las que se encargan de un único caso cada vez, lo que significa que un único método puede necesitar varias pruebas unitarias si su funcionamiento varía en función del contexto. (22)

Las pruebas unitarias del *framework* Symfony son archivos PHP normales cuyo nombre termina en *Test.php* y que se encuentran en el directorio *test/unit/* de la aplicación. Su sintaxis es sencilla y fácil de implementar. Cada prueba unitaria consiste en una llamada a un método de la instancia de *lime_test*. El último parámetro de estos métodos siempre es una cadena de texto opcional que se utiliza como resultado del método. 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 comando es muy explícito, lo que permite localizar fácilmente las pruebas que han fallado y las que se han ejecutado correctamente. (22)

Aplicación de pruebas unitarias al sistema

A continuación se muestran las pruebas unitarias del *framework* Symfony aplicadas a algunas de las funcionalidades del **RF Mostrar Dependencias** las cuales consisten en dado un tipo de componente y un componente, retornar las dependencias del mismo.

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

```
$aplicaciones = sfUtilDependencias::obtenerListadoAplicaciones();
$plugins = sfUtilDependencias::obtenerListadoPlugins();
$componentes = array_merge($aplicaciones, $plugins);
$fileDependencias = sfUtilDependencias::obtenerFicheroDependencias();

$t->is(count($aplicaciones), 18, 'Comprobando que devuelva todas las aplicaciones');
$t->is(count($plugins), 20, 'Comprobando que devuelva  todos los plugins');
$t->isa_ok($fileDependencias, 'array', 'Comprobando que devuelva un arreglo');
$t->is($componentes['sfLcfPlugin']['tipo'],'plugin', 'Comprobando que devuelva un plugin');
```

Al ejecutar estas pruebas el resultado arrojado es el siguiente:

```
ok 1 - Comprobando que devuelva todas las aplicaciones
ok 2 - Comprobando que devuelva todos los plugins
ok 3 - Comprobando que devuelva un arreglo
ok 4 - Comprobando que devuelva un plugin

Looks like everything went fine.
```

Como se observa, el resultado final muestra que la prueba realizada se ejecutó correctamente.

3.4 Aporte y novedad de la solución

Con la obtención de la solución propuesta, se introdujo una herramienta para la gestión de dependencias de los componentes que integran al sistema GINA nunca antes implementada, contribuyendo de esta forma a la independencia tecnológica del país y al proceso de informatización de la sociedad cubana.

Haciendo uso de herramientas y tecnologías libres, en su gran mayoría, fue posible obtener una solución que es capaz de integrarse e interactuar con los subsistemas del sistema GINA, además es integrable al *framework* Symfony 1.x, por lo que podrá ser utilizada por otros sistemas que hagan uso de estas versiones del *framework*.

3.5 Conclusiones parciales

En este capítulo se obtuvieron los artefactos pertenecientes a la etapa de implementación. Se definieron además los estándares de codificación usados para la implementación de la solución anteriormente modelada que permitieron crear un código de fácil entendimiento. De igual forma se analizaron los resultados obtenidos de las pruebas funcionales realizadas a un requisito funcional seleccionado con el objetivo de validar la solución obtenida. El análisis de estos datos

arrojó resultados satisfactorios por lo que concluye que la solución desarrollada se encuentra en condiciones para ser usada por el sistema GINA.

Conclusiones Generales

Una vez concluido el presente trabajo de diploma se logra el cumplimiento de los objetivos trazados para la investigación. A partir de ello se arriba a las siguientes conclusiones:

- ✓ Se caracterizaron los sistemas informáticos mencionados arrojando al gestor de dependencias Composer como punto de partida para la solución.
- ✓ Se identificaron los requisitos funcionales más significativos para el sistema y se generaron los artefactos correspondientes para mejor entendimiento de la solución.
- ✓ La caracterización de un conjunto de herramientas y tecnologías, así como su posterior empleo, posibilitó la obtención de la solución propuesta, cumpliendo con las tareas y objetivos propuestos.
- ✓ El componente obtenido durante la investigación permitió el control de las dependencias entre los subsistemas del sistema GINA.
- ✓ 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.

Recomendaciones

Los objetivos trazados en la presente investigación fueron alcanzados satisfactoriamente, sin embargo se recomienda:

- Continuar el estudio de los estándares relacionados con los servicios web con el objetivo de proveer a la solución de las mejores prácticas en este sector.
- Continuar con el seguimiento de las actualizaciones de las herramientas y tecnologías informáticas empleadas en la solución para garantizar mejoras en futuras versiones del componente.
- Analizar la posibilidad de incorporarle a la solución obtenida un módulo que haga uso de base de datos, en combinación con el estándar YML, para almacenar las dependencias entre los componentes, logrando con ello un mayor rendimiento y escalabilidad del sistema.

Glosario de Términos

AGR: Aduana General de la República.

CEIGE: Centro de Informatización de la Gestión de Entidades.

CLI: Interfaz de Líneas de Comando.

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.

GINA: Gestión Integral Aduanera.

IDE: Entorno Integrado de Desarrollo (*Integrated Development Environment*).

JSON: Notación de Objetos de JavaScript.

Plugin: Pequeño programa que añade alguna función a otro programa, habitualmente de mayor tamaño. Un programa puede tener uno o más conectores. Son muy utilizados en los programas navegadores para ampliar sus funcionalidades.

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.

PHP: PHP *Hypertext Pre-processor* es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas.

Symfony: framework de trabajo para la realización de aplicaciones web escrito en PHP 5.

UCI: Universidad de las Ciencias Informáticas.

XML: *Extensible Markup Language*, es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C)

YAML: *(Ain't Another Markup Language)* es un estándar para la serialización de datos

Bibliografía

1. Universidad de las Ciencias Informáticas. [En línea] [Citado el: 04 de Junio de 2013.] http://www.uci.cu/historia.

2. **Aduana General de la República.** Aduana - CUBA. [En línea] [Citado el: 04 de Junio de 2013.] http://www.aduana.co.cu/.

3. **Ceruzzi, Paul E.** *A History of Modern Computing.* Cambridge : MIT Press, (2000). ISBN 0-262-03255-4.

4. Wordpress Plug-in. *Description of the Wordpress Plug-in architecture.* [En línea] 2007. http:// codex. wordpress. org/ Plugin_API.

5. **IBM.** IBM. [En línea] 2013. [Citado el: 15 de 01 de 2013.] www.pic.dhe.ibm.com.

6. **Composer.** Composer. [En línea] 15 de 01 de 2013. [Citado el: 15 de 01 de 2013.] http://getcomposer.org/.

7. PEAR-PHP Extension and Application Repository. [En línea] The PEAR Group, 2001. WWW.pear.php.net.

8. YUM. [En línea] www.slideshare.net/gnrfan/gestor-de-paquetes-yum.

9. Apache Maven Project. [En línea] The Apache Software Foundation, 2002. [Citado el: 16 de 01 de 2013.] www.maven.apache.org.

10. **Entidades, Centro de Informatización de Gestión de.** *Modelo de Dasarrollo de Software.* 2012.

11. Lenguajes de Programación. [En línea] Universidad Tecnológica Nacional - Facultad Regional Tucumán , 2000. [Citado el: 5 de diciembre de 2012.] http://frt.utn.edu.ar/sistemas/paradigmas/lenguajes.htm.

12. Diccionario Web - Estudio de Contenidos Digitales. [En línea] Pixelkit, 2009. [Citado el: 4 de diciembre de 2013.] www.pixelkit.cl/diccionario-web/..

13. **Universidad Carlos III de Madrid.** *JavaScript.* [En línea] 2006. [Citado el: 4 de diciembre de 2013.] www.perso.wanadoo.es/javascript_12.

14. PHP: Hypertext Preprocessor. [En línea] Comunidad PHP, 2001. www.php.net.

15. Programación en Castellano. *¿Por qué elegir PHP?* [En línea] 1998. http://www.programacion.com/articulo/por_que_elegir_php_143.

Bibliografía

16. **Barzanallana, Rafael.** Informática. Lenguajes. [En línea] Universidad de Murcia, 3 de noviembre de 2012. www.um.es/docencia/barzana/DIVULGACION/INFORMATICA/Que-son-lenguajes-marcado.html.

17. **Webtutoriales.** YAML. [En línea] 26 de Noviembre de 2011. www.webtutoriales.com.

18. **Lescano, Walter Sagástegui.** Aprender a Programar. [En línea] 2011. www.aprenderaprogramar.com.

19. Libros Web. [En línea] www.librosweb.es.

20. CodeBox. *Glosario.* [En línea] 2008. http://www.codebox.es/glosario.

21. Ext JS en Español. [En línea] Comunidad de Desarrollo de Ext JS. www.extjs.es.

22. **Potencier, Fabien.** *Symfony la guía definitiva.* 2008.

23. *Entornos de Desarrollo Integrado.* [En línea] Escuela de Ingeniería Informática Universidad de Oviedo, 2008. http://petra.euitio.uniovi.es/~i1667065/HD/documentos/Entornos%de%Desarrollo%Integrado.pdf..

24. Portal del IDE Java de Código Abierto. [En línea] Oracle Corporation, 2010. www.netbeans.org.

25. *UML, BPMN and Database Tool for Software Development.* [En línea] Visual Paradigm International, 2009. www.visual-paradigm.com.

26. **Escalona, María José y Koch , Nora.** Departamento de Lenguajes y Sistemas Informáticos. *Ingeniería de Requisitos en Aplicaciones para la Web – Un estudio comparativo .* [En línea]

27. **Entidades, Centro de Informatización de la Gestión de.** *Especificación de Requisitos de Software.* 2010.

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

29. **Potencier, Fabien.** *Más con Symfony.* 2009.

30. **Fowler, Martin.** *Patterns of Enterprise Application Architecture.* ISBN: 0-32112-742-0.

31. **Booch, Grady, Rumbaugh, James y Jacobson, Ivar.** *El Lenguaje Unificado de Modelado Manual de Referencia.* s.l. : Addison-Wesley.

32. *SIMETSE – SIstema de METricas.* **Lemus, Yuniersy Peña y Hernández Díaz, Yudisleidys .** Ciudad de la Habana: Universidad de la Ciencias Informaticas : s.n., 2007.

33. **Pressman, Roger S.** *Ingienería de Requisitos Un enfoque práctico.* 2005.

34. *Estándares de codificación para Proyectos Aduana.* s.l. : Centro de Gestión de Entidades, Universidad de las Ciencias Informáticas.

Bibliografía

35. **Juristo, Natalia, Moreno, Ana M. y Vegas, Sira.** *Técnicas de Evaluación de Software.* 2006.

36. **Larman, Craig.** *UML y Patrones, Introducción al Análisis y Diseño Orientado a Objetos.* Montevideo Uruguay : Prentice Hall, Pearson, 2001. 053681.

37. NetBeans. [En línea] [Citado el: 03 de marzo de 2012.] http://netbeans.org/features/index.html.

38. Universidad Carlos III de Madrid. *JavaScript.* [En línea] 2006. [Citado el: 4 de diciembre de 2013.] http://perso.wanadoo.es/javascript_12.

39. **Reyero, Eusebio.** Metodologías de diseño. *Metodologías de diseño.* [En línea] 2008. www.thespacer.net/blog/metodologias-de-diseno/.

40. *Ingeniería de Requisitos en Aplicaciones para la Web - Un estudio comparativo*. **José Escalona, María and Koch, Nora.** Universidad de Sevilla: Departamento de Lenguajes y Sistemas Informáticos : s.n., 2002.

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

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

43. **Pressman, Roger S.** *Ingeniería del Software. Un enfoque práctico. (Sexta Edición).* 2005. ISBN: 9701054733.

Anexos

Anexo 1: Descripción de los requisitos funcionales del componente *sfDependenciasPlugin*.

RF Habilitar componentes instalados por pantalla.

Precondiciones	Habilitar el componente con la información de sus dependencias.	
Flujo de eventos		
Flujo básico Habilitar componentes instalados por pantalla.		
No	**Actor**	**Sistema**
1.	Selecciona el componente que desea activar.	
2.		Muestra los componentes deshabilitados vigentes de la aplicación con sus informaciones actuales.
3.	Accede a la opción *"Habilitar Componente"*.	
4.	Oprime el botón "*Habilitar Componente"*.	
5.		Habilita el componente seleccionado.
6.		Concluye el requisito
Pos-condiciones		
1	Se habilitan los componentes seleccionados.	
Flujos alternativos		
Flujo alternativo		
No	**Actor**	**Sistema**
1		
2		
Pos-condiciones		

1.		
Validaciones		
6.		
Relaciones	**Requisitos Incluidos**	No aplicable.
	Extensiones	No aplicable.
Requisitos especiales	No aplicable	
Asuntos pendientes	No aplicable	

RF Habilitar componentes instalados por líneas de comando.

Precondiciones	Habilitar el componente con la información de sus dependencias.
Flujo de eventos	
Flujo básico	**Habilitar componentes instalados por líneas de comando.**

No	**Actor**	**Sistema**
1.	Escribe en la consola un comando para habilitar el componente.	
2.	Introduce el comando para activar el componente.	
3.		Activa el componente seleccionado.
4.		Concluye el requisito

Pos-condiciones	
2	Se habilitan los componentes seleccionados.
Flujos alternativos	
Flujo alternativo	

No	Actor	Sistema
1		
2		
Pos-condiciones		
2.		
Validaciones		
7.		

Relaciones	**Requisitos Incluidos**	No aplicable.
	Extensiones	No aplicable.
Requisitos especiales	No aplicable	
Asuntos pendientes	No aplicable	

RF Deshabilitar componentes instalados por pantalla.

Precondiciones	Deshabilitar el componente con la información de sus dependencias.
Flujo de eventos	
Flujo básico	**Deshabilitar componentes instalados.**

No	Actor	Sistema
	Selecciona el componente que desea deshabilitar.	
		Muestra los componentes habilitados vigentes de la aplicación con sus informaciones actuales.
	Accede a la opción *"Deshabilitar Componentes"*.	

	Oprime el botón *"Deshabilitar Componentes"*.	
		Deshabilita el componente seleccionado.
		Concluye el requisito

Pos-condiciones	
3	Se deshabilitan los componentes seleccionados.

Flujos alternativos		
Flujo alternativo		
No	**Actor**	**Sistema**
1		
2		
Pos-condiciones		
3.		
Validaciones		
7.		

Relaciones	**Requisitos Incluidos**	No aplicable.
	Extensiones	No aplicable.
Requisitos especiales	No aplicable	
Asuntos pendientes	No aplicable	

RF Deshabilitar componentes instalados por líneas de comando.	
Precondiciones	Deshabilitar el componente con la información de sus dependencias.
Flujo de eventos	

Flujo básico	**Deshabilitar componentes instalados.**	
No	**Actor**	**Sistema**
1.	Escribe en la consola un comando para deshabilitar componentes.	
2.	Introduce el comando para desactivar el componente.	
3.		Desactiva el componente introducido.
4.		Concluye el requisito
Pos-condiciones		
1	Se desactivan los componentes seleccionados.	
Flujos alternativos		
Flujo alternativo		
No	**Actor**	**Sistema**
1		
2		
Pos-condiciones		
1.		
Validaciones		
1.		
Relaciones	**Requisitos Incluidos**	No aplicable.
	Extensiones	No aplicable.
Requisitos especiales	No aplicable	
Asuntos pendientes	No aplicable	

RF Gestionar dependencias por pantalla.

Precondiciones	Gestionar dependencias de los componentes.

Flujo de eventos

Flujo básico Gestionar dependencias por pantalla.

No	Actor	Sistema
1.	Selecciona el componente que desea generarle las dependencias.	
2.		Muestra los componentes vigentes de la aplicación con sus informaciones actuales.
3.	Oprime el botón *"Gestionar Dependencias"*.	
4.		Muestra el prototipo de interfaz opción *"Gestionar Dependencias"*.
5.	Selecciona las dependencias del componente.	
6.	Oprime el botón *"Aceptar"*.	
7.		Actualiza las dependencias seleccionadas.
8.		Concluye el requisito

Pos-condiciones

2	Se actualizan las dependencias seleccionadas.

Flujos alternativos

Flujo alternativo

No	Actor	Sistema
1		
2		

Pos-condiciones		
2.		
Validaciones		
2.		
Relaciones	**Requisitos Incluidos**	No aplicable.
	Extensiones	No aplicable.
Requisitos especiales	No aplicable	
Asuntos pendientes	No aplicable	

RF Gestionar dependencias del negocio por líneas de comando.

Precondiciones	Gestionar dependencias de los componentes.
Flujo de eventos	
Flujo básico	**Gestionar dependencias del negocio por líneas de comando.**

No	**Actor**	**Sistema**
1.	Escribe en la consola un comando para actualizar dependencias.	
2.	Introduce el comando para actualizar las dependencias del componente.	
3.		Actualiza las dependencias.
4.		Concluye el requisito

Pos-condiciones	
1.	Se actualizan las dependencias seleccionadas.
Flujos alternativos	

Flujo alternativo		
No	**Actor**	**Sistema**
1		
2		
Pos-condiciones		
1.		
Validaciones		
1.		
Relaciones	**Requisitos Incluidos**	No aplicable.
	Extensiones	No aplicable.
Requisitos especiales	No aplicable	
Asuntos pendientes	No aplicable	

RF Mostrar dependencias por líneas de comando.		
Precondiciones	Mostrar dependencias de los componentes.	
Flujo de eventos		
Flujo básico	**Mostrar dependencias del negocio por líneas de comando.**	
No	**Actor**	**Sistema**
1.	Escribe en la consola un comando para mostrar dependencias de los componentes.	
2.	Introduce el comando para mostrar dependencias.	
3.		Muestra los componentes con sus

	dependencias.
4.	Muestra las dependencias existentes.
5.	Concluye el requisito

Pos-condiciones
3 Se muestran las dependencias de los componentes.

Flujos alternativos
Flujo alternativo

No	**Actor**	**Sistema**
1		
2		

Pos-condiciones
2.

Validaciones
2.

Relaciones	**Requisitos Incluidos**	No aplicable.
	Extensiones	No aplicable.
Requisitos especiales	No aplicable	
Asuntos pendientes	No aplicable	

RF Mostrar dependencias por pantalla.

Precondiciones	Mostrar dependencias de los componentes.
Flujo de eventos	

Flujo básico Mostrar dependencias de los componentes por pantalla.		
No	**Actor**	**Sistema**
1.	Selecciona el botón “Mostrar Dependencias”.	
2.		Muestra el prototipo de interfaz *“Mostrar Dependencias”*.
3.	Selecciona un componente	
4.		Muestra los datos generales del componente, las dependencias directas y las dependencias inversas.
5.		Concluye el requisito
Pos-condiciones		
1.	Se muestran dependencias.	
Flujos alternativos		
Flujo alternativo		
No	**Actor**	**Sistema**
1		
2		
Pos-condiciones		
1.		
Validaciones		
2.		
Relaciones	**Requisitos Incluidos**	No aplicable.
	Extensiones	No aplicable.
Requisitos	No aplicable	

especiales	
Asuntos pendientes	No aplicable

RF Generar fichero de dependencia del negocio por líneas de comando.

Precondiciones	Generar fichero de dependencias

Flujo de eventos

Flujo básico Generar fichero de dependencia del negocio para los módulos creados.

No	Actor	Sistema
1.	Escribe en la consola un comando para generar fichero de dependencias.	
2.	Introduce el comando para crear el fichero de dependencia.	
3.		Verifica los componentes que no tienen las dependencias creadas.
4.		Crea el fichero.
5.		Concluye el requisito

Pos-condiciones

4	Se crea el fichero dependencias.

Flujos alternativos

Flujo alternativo

No	Actor	Sistema
1		
2		

Pos-condiciones

2.	

Validaciones		
3.		
Relaciones	**Requisitos Incluidos**	No aplicable.
	Extensiones	No aplicable.
Requisitos especiales	No aplicable	
Asuntos pendientes	No aplicable	

RF Brindar estado de dependencias del negocio.

Precondiciones	Brindar dependencias haciendo uso de servicios web.

Flujo de eventos

Flujo básico Brindar estado de dependencias del negocio.

No	**Actor**	**Sistema**
1.	Introduce un componente por parámetro.	
2.		Muestra una lista con las dependencias de los componentes a través de los servicios web.
3.		Concluye el requisito

Pos-condiciones

5	Se brinda un listado de dependencias.

Flujos alternativos

Flujo alternativo

No	**Actor**	**Sistema**

1		
2		
Pos-condiciones		
3.		
Validaciones		
4.		
Relaciones	**Requisitos Incluidos**	No aplicable.
	Extensiones	No aplicable.
Requisitos especiales	No aplicable	
Asuntos pendientes	No aplicable	

Anexo 2: Diagramas de secuencia.

Diagrama de secuencia: Deshabilitar Componentes.

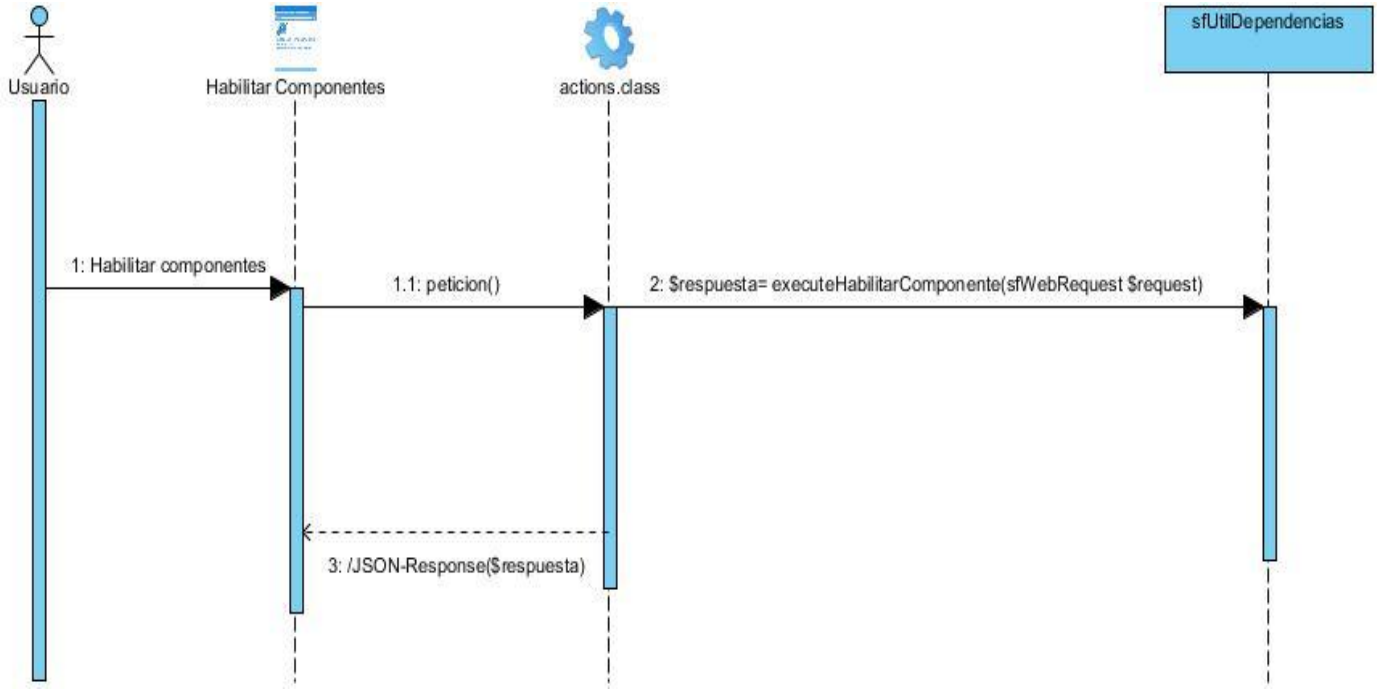


Diagrama de secuencia: Deshabilitar Componentes.

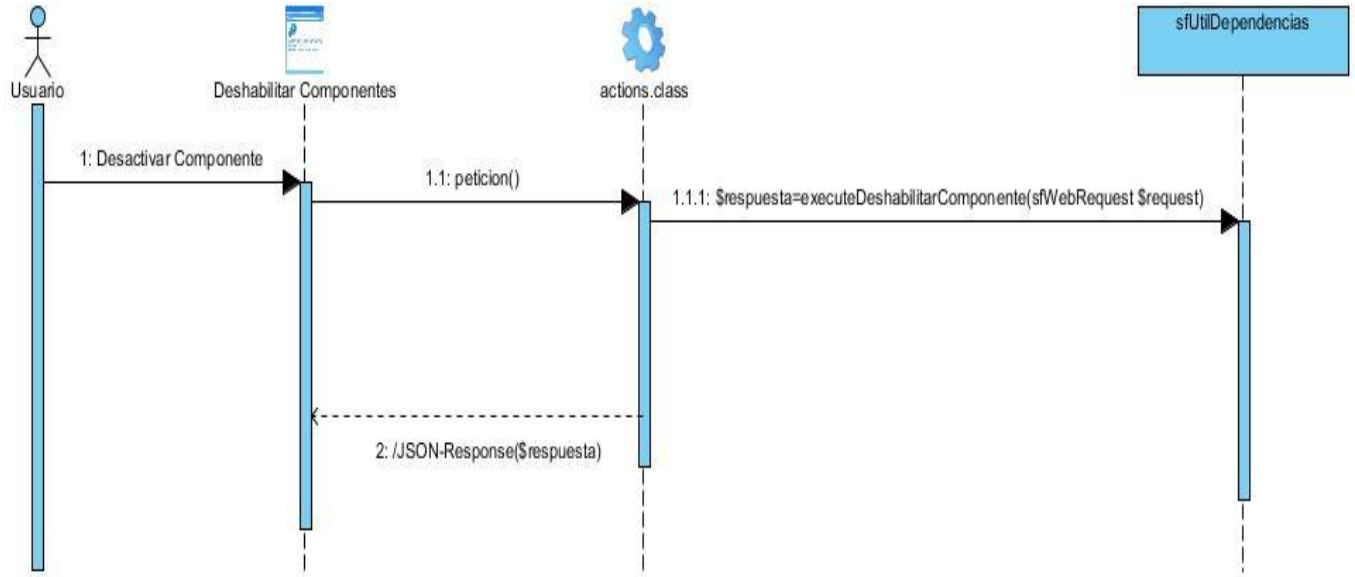
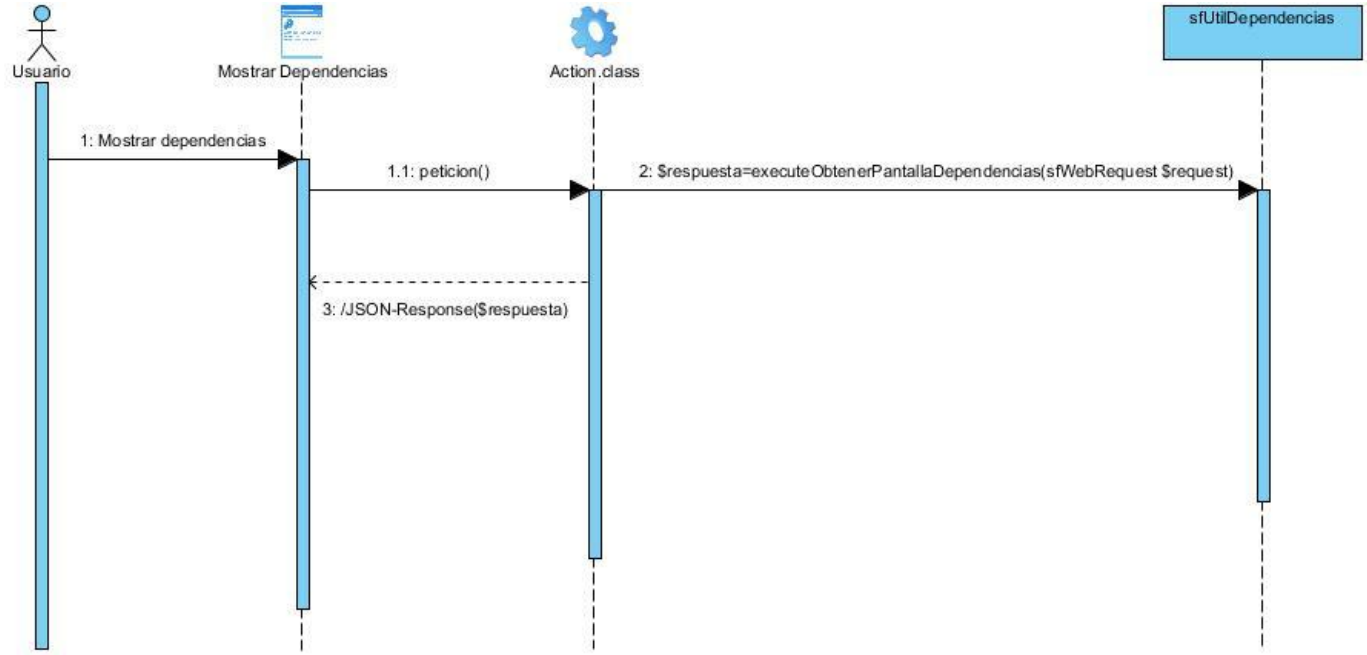
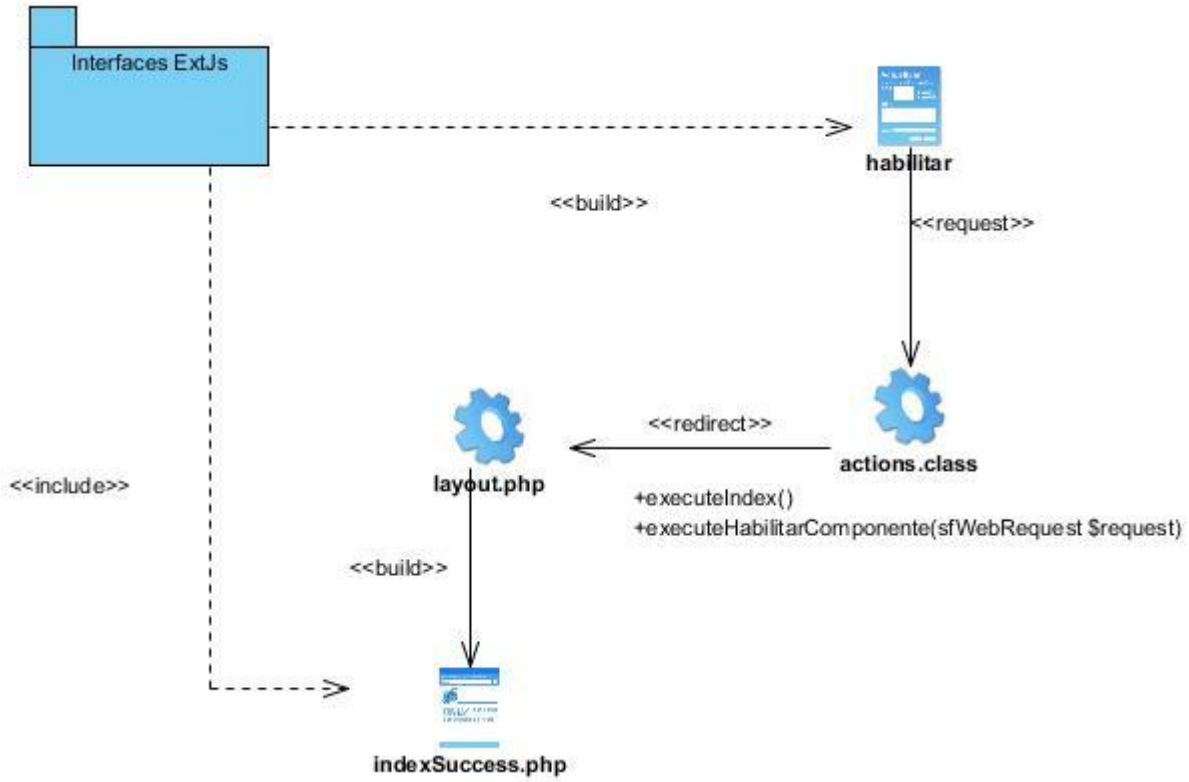


Diagrama de secuencia: Mostrar Dependencias.

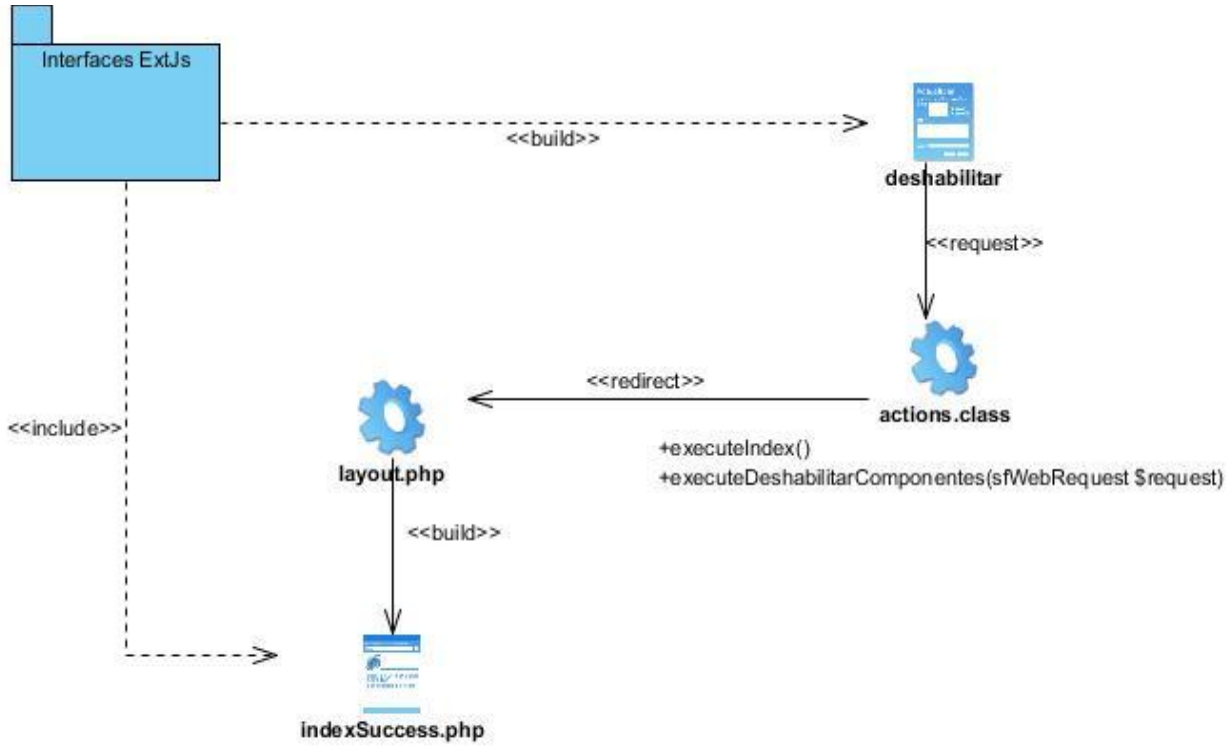


Anexo 3: Diagramas de clases del diseño con estereotipos web.

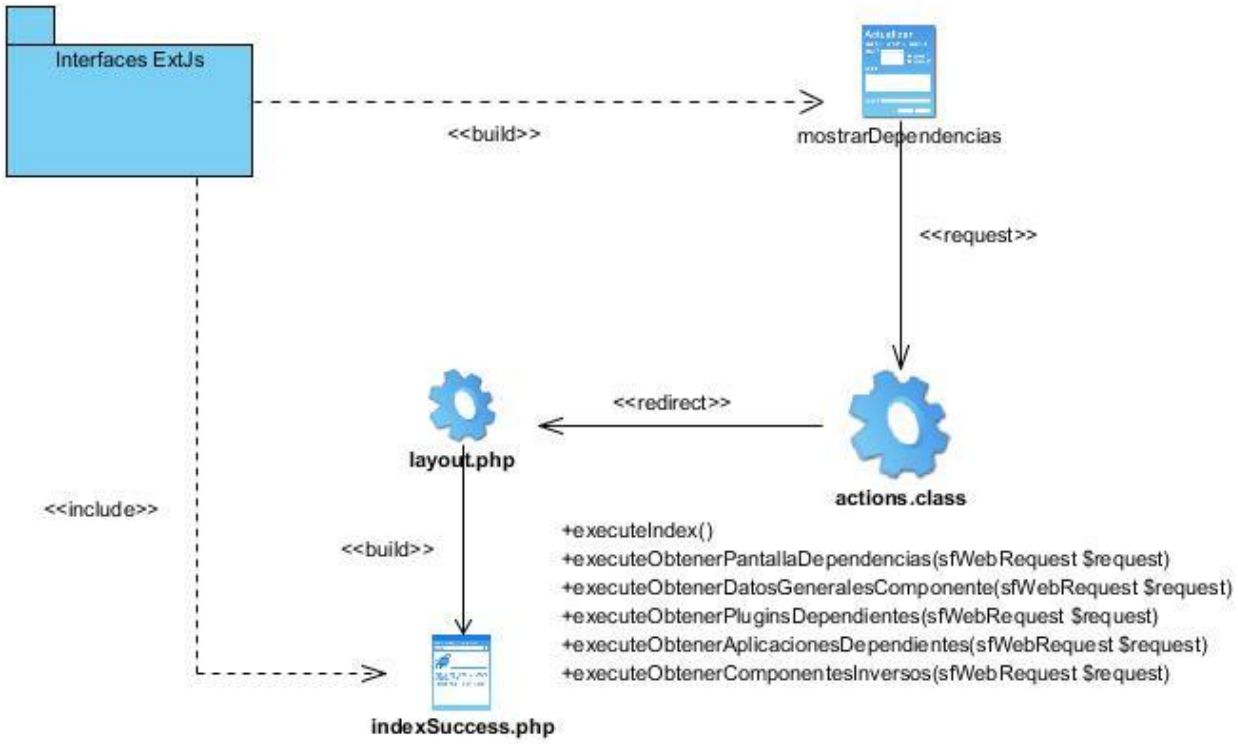
Diagramas de clases del diseño: Habilitar dependencias.



Diagramas de clases del diseño: Deshabilitar componentes.



Diagramas de clases del diseño: Mostrar dependencias.



Anexo 4: Diseño de Casos de Pruebas.

Diseño de Caso de Prueba Deshabilitar Componentes.

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Deshabilitar Componentes	Se deshabilita el componente habilitado	Muestra un mensaje de confirmación: ¿Está seguro de deshabilitar el componente seleccionado?	1. Clic en el botón "Gestión de Dependencias" 2. Seleccionar componente en el panel "Componentes". 3. Clic en el botón "Deshabilitar Componente".
EC 1.2 Habilitar Componentes	Se realiza este escenario para cuando el componente esta deshabilitado	Muestra un mensaje de error: "Este componente ya se encuentra deshabilitado".	1. Clic en el botón "Gestión de Dependencias" 2. Seleccionar componente en el panel "Componentes". 3. Clic en el botón "Deshabilitar Componente".
EC 1.3 Cuando no hay componentes seleccionados	Se realiza este escenario en el caso que no se haya seleccionado un componente	Muestra un mensaje de confirmación: "Para deshabilitar un componente primeramente debe seleccionarlo del listado"	1. Clic en el botón "Gestión de Dependencias". 2. Clic en el botón "Deshabilitar Componentes".

Diseño de Caso de Prueba Habilitar Componentes.

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Habilitar Componentes	Se habilita el componente deshabilitado	Muestra un mensaje de confirmación: ¿Está seguro de habilitar el componente seleccionado?	1. Clic en el botón "Gestión de Dependencias" 2. Seleccionar componente en el panel "Componentes". 3. Clic en el botón "Habilitar Componente".
EC 1.2 Habilitar Componentes	Se realiza este escenario para cuando el componente está habilitado	Muestra un mensaje de error: "Este componente ya se encuentra habilitado".	1. Clic en el botón "Gestión de Dependencias" 2. Seleccionar componente en el panel "Componentes". 3. Clic en el botón "Habilitar Componente".
EC 1.3 Cuando no hay componentes seleccionados	Se realiza este escenario en el caso que no se haya seleccionado un componente	Muestra un mensaje de confirmación: Para habilitar un componente primeramente debe seleccionarlo del listado	1. Clic en el botón "Gestión de Dependencias" 2. Clic en el botón "Habilitar Componentes".

Diseño de Caso de Prueba Mostrar dependencias.

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Mostrar Dependencias	Se muestran las dependencias del componente correctamente.	Muestra los datos Generales del componente y las Dependencias directas con las aplicaciones y plugins	1. Clic en el botón "Gestión de Dependencias" 2. Clic en el botón "Mostrar Dependencias". 3. Seleccionar el componente deseado en el panel "Componentes".
EC 1.2 Mostrar Dependencias inversas	Se realiza este escenario para ver si los componentes tienen dependencias inversas	Muestra los componentes inversos.	1. Clic en el botón "Gestión de Dependencias" 2. Clic en el botón "Mostrar Dependencias". 3. Seleccionar el componente deseado en el panel "Componentes". 3. Seleccionar el tab "Componentes Inversos".

EC 1.3 Cuando no hay dependencias directas	Se realiza este escenario en caso que no haya dependencias directas con aplicaciones.	Muestra en el panel de Aplicaciones el mensaje: "No hay elementos para mostrar"	1. Clic en el botón "Gestión de Dependencias" 2. Clic en el botón "Mostrar Dependencias". 3. Seleccionar el componente deseado en el panel "Componentes".
EC 1.4 Cuando no hay dependencias directas	Se realiza este escenario en caso que no haya dependencias directas con plugins.	Muestra en el panel de Plugins el mensaje: "No hay elementos para mostrar"	1. Clic en el botón "Gestión de Dependencias" 2. Clic en el botón "Mostrar Dependencias". 3. Seleccionar el componente deseado en el panel "Componentes".
EC 1.5 Cuando no hay Dependencias inversas	Se realiza este escenario en caso que no haya dependencias inversas	Muestra en el panel de Componentes el mensaje: "No hay elementos para mostrar"	1. Clic en el botón "Gestión de Dependencias" 2. Clic en el botón "Mostrar Dependencias". 3. Seleccionar el componente deseado en el panel "Componentes". 4. Seleccionar el tab "Componentes Inversos".