UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 3

Título: Solución para la generación dinámica de componentes en el Marco de Trabajo Sauxe.

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

Autor: Yankiel González Fuentes

Tutor: MSc. Oiner Gómez Baryolo

Co-Tutora: Ing. Ariadna Rendón Artola

Mayo, 2012

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad # 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Yankiel González Fuentes	MsC. Oiner Gómez Baryolo	Ing. Ariadna Rendón Artola
_____	_____	_____

“El hombre inteligente no es el que tiene muchas ideas,

sino el que sabe sacar provecho de las pocas que tiene."

Anónimo

AGRADECIMIENTOS

A mis padres Odalys y Reymundo que siempre cuidaron de mí, me dieron todo para que solo me concentrara en los estudios, para así poder construir mi futuro, este que construí con esfuerzo y dedicación y perseverancia, nunca lo hubiese podido lograr sin su confianza, su amor y su gran apoyo en todos los momentos difíciles. Gracias por hacerme ver la vida de una manera bella y llena de alegrías, a nunca dejarme vencer por nada ni nadie, a salir siempre al frente de las circunstancias que me agobiaban. De veras infinitas gracias.

A mi abuela Gladys que siempre tuvo un consejo para cada duda o preocupación, un mimo o una palabra dulce para cada travesura a lo largo de todos estos años. Por su amor inmenso y su dedicación.

A mis tías, tíos, primos y primas que siempre me dieron todo su cariño, apoyo y amor en cada momento de mi vida y porque siempre quisieron lo mejor para mí.

Al resto de mi familia por su confianza, dedicación y comprensión que depositaron en mí.

A mis compañeros de estudio que siempre estuvieron ahí cuando los necesité.

A mis hermanas Hanny y Yarimis, por su apoyo, su amistad incondicional, su solidaridad, su ayuda en todo lo que me hizo falta, por nunca decir un no ante cualquier situación, por su sentido de la amistad, por saberme escuchar y por estar allí siempre dispuestas para compartir cada alegría o tristeza, para darme ánimos en los momentos más difíciles por los cuales pasé y para reír cuando había que hacerlo.

A Yulio, mi hermano Yulio que siempre estuvo presente a mi lado cuando más falta me hizo, que siempre me ayudó sin poner una objeción, sin poner obstáculos. Muchas gracias por ser mi hermano.

A todos mis amigos en especial a Chirino, Edwing, Sussy, Yake, Maday, Betty, Patricia, Magui, Ania, Maylin Dayana, Dayana, Yaidel, Adonis, Daynelis, Mili, Ivian, Mely, Zayda, Susana, Yaíma, Yahumara, por poder contar con ellos en todo momento y formar parte de muchas alegrías.

A Yady, Yaíma, Zury y Baby las muchachitas de la Secretaría que siempre me atendieron con mucho amor y cariño cuando tuve que ir a la Secretaría de la Facultad #3, además agradecerles porque esta tesis es también parte de la dedicación de ellas.

A mi tribunal por ser tan especial y apoyarme y ayudarme cuando más lo necesitaba, los cuales son parte importante de este trabajo. A mi oponente Matos corregirme cada uno de los problemas presentes en el trabajo de diploma.

A Fidel por haberme dado la posibilidad de estudiar en la Universidad de las Ciencias Informáticas y formarme como Ingeniero.

A mis profesores que me guiaron siempre hasta llegar a la meta final.

A mi tutor Oiner que me ayudó en el Trabajo de Diploma.

A Ariadna, qué hubiera sido de mi sin tu ayuda, te agradezco cada segundo, minuto, horas, horas que me dedicaste, de verdad te agradezco muchísimo, no eres capaz de imaginarte cuanto, pero es cierto, este trabajo es fruto de dedicación, entrega, consagración, cualquier calificativo que exista es poco comparado con lo que has puesto en el trabajo. De verdad muchas gracias. Te quiero por siempre.

Si se me queda alguien quiero que me disculpe, solo decir que lo más importante es llevarlos a todas en el corazón, que ahí es donde están todos.

A todos estas personas

Muchas gracias.

DEDICATORIA

A mis padres Odalys y Reymundo que siempre cuidaron de mí, me dieron todo para que solo me concentrara en los estudios, para así poder construir mi futuro este que construí con todo mi esfuerzo y dedicación, nunca lo hubiese podido lograr sin su confianza su amor y su gran apoyo en todos los momentos difíciles.

A mi abuela Gladys que siempre tuvo un consejo para cada duda o preocupación, un mimo o una palabra dulce para cada travesura a lo largo de todos estos años. Por su amor inmenso y su dedicación.

RESUMEN

El marco de trabajo Sauxe desarrollado por el Centro de Informatización de la Gestión de Entidades (CEIGE), tiene como objetivo soportar el desarrollo de aplicaciones web de gestión. Entre sus características más importantes se pueden señalar que está orientado a componentes y provee una estructura genérica posibilitando que el desarrollador logre una mayor estandarización, flexibilidad, integración y agilidad en el desarrollo del producto final. Sin embargo, la creación de componentes dentro de Sauxe se realiza actualmente de forma manual, provocando demora en la implementación de los productos de software que en él se desarrollan, así como un incremento de la introducción de errores por parte de los programadores. Este proceso además de resultar lento y engorroso por gran número de elementos de configuración a tener en cuenta, no se logra una estandarización en la creación de los componentes. Con la presente investigación se propone una solución para la generación dinámica de componentes dentro del marco de trabajo Sauxe, de modo que se facilite el proceso de gestión de los mismos.

Palabras claves: componente, marco de trabajo, aplicaciones web, Sauxe.

CONTENIDO

INDICE DE FIGURAS

INTRODUCCIÓN

En la actualidad, gracias al avance tecnológico que se ha venido presentando, la mayoría de las operaciones y actividades son manejadas con la ayuda de las computadoras. Las Tecnologías de la Información y las Comunicaciones (TIC) han jugado un papel importante en este avance.

La Industria Cubana del Software (ICSW) está llamada a convertirse en una significativa fuente de ingresos para el país, como resultado del correcto aprovechamiento de la calidad del capital humano disponible. Por esta razón existen diferentes entidades que se dedican a la producción de software, una de ellas es la Universidad de las Ciencias Informáticas (UCI).

En la UCI existen diferentes centros de producción, entre ellos se encuentra el Centro de Informatización de la Gestión de Entidades (CEIGE) que se dedica al desarrollo de sistemas de gestión para el entorno empresarial utilizando tecnologías web. En dicho centro radica el Departamento de Tecnología donde se desarrolló un framework (*marco de trabajo*) denominado Sauxe, que fue creado con el objetivo de agilizar el proceso de desarrollo y garantizar una base tecnológica sólida que soporte el desarrollo de soluciones de esta envergadura.

Sauxe es un Marco de Trabajo que contiene un conjunto de componentes reutilizables que provee la estructura genérica permitiendo una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo. Está orientado a componentes, siendo esto consecuente con la definición que propone la IEEE según el Instituto de Ingeniería de Software (*SEI, del inglés Software Engineering Institute*), se puede afirmar que un componente, en esencia, es una unidad reutilizable que puede interoperar con otros módulos software por medio de sus interfaces, las cuales define desde donde se puede tener acceso a los servicios que este ofrece a los demás componentes. Un componente puede presentarse en forma de código fuente o código objeto; puede estar escrito en lenguaje funcional, procedural u orientado a objetos. (1)

El marco de trabajo Sauxe está orientado a componentes, además presenta una estructura genérica para aplicaciones web de gestión, que provee una mayor estandarización, flexibilidad, integración y agilidad al proceso de desarrollo. Estos componentes se desarrollan de forma manual, esto implica que se tengan que reescribir códigos en varios ficheros que nunca varían y que requieren de especial atención para su funcionamiento. Esta forma de crear la estructura básica de los componentes genera errores que se traducen en pérdidas de tiempo y atraso en los cronogramas de desarrollo. Además influye de forma negativa en la estandarización, extensión, seguridad y mantenimiento de los componentes que se implementan utilizando Sauxe.

De lo antes planteado se puede definir el siguiente **problema a resolver**:
¿Cómo disminuir el tiempo de desarrollo y evitar la introducción de errores por parte de los programadores en la creación de componentes dentro del marco de trabajo Sauxe?

Se define como **objeto de estudio** la generación dinámica de componentes y como **campo de acción** generación dinámica de componentes en el marco de trabajo Sauxe.

Para dar solución al problema planteado se propone como **objetivo general**:
Desarrollar una solución para la generación dinámica de componentes en el marco de trabajo Sauxe para disminuir el tiempo de desarrollo y la introducción de errores por parte de los programadores.

Objetivos específicos:

- Establecer el Marco Teórico de las soluciones para la generación de componentes, que permita identificar puntos y debilidades de reutilización.
- Realizar el Análisis y Diseño de la solución con el fin de construir una aplicación que cumpla con las necesidades del cliente.
- Implementar la propuesta de solución haciendo uso de herramientas afines al desarrollo de aplicaciones web.
- Aplicar métricas de diseño y pruebas funcionales a la aplicación para

validar la solución propuesta.

Se define como **hipótesis** que si se desarrolla una solución para la generación dinámica de componentes en el marco de trabajo Sauxe, se logrará disminuir el tiempo de desarrollo y la introducción de errores por parte de los programadores a la hora de desarrollar componentes.

Con el cumplimiento de los elementos antes planteados se tiene como **posibles resultados**: Solución para la generación dinámica de componentes dentro del marco de trabajo Sauxe que cumpla con los requisitos que requiere la arquitectura definida en el Departamento de Tecnología.

Se ha determinado además que se utilizarán las estrategias y métodos de investigación siguientes:

- El **método histórico lógico** se utilizará para realizar una revisión histórica del desarrollo de las soluciones para la generación dinámica de componentes para aplicaciones web de gestión y determinar si actualmente se están desarrollando sistemas informáticos similares, tomando una posición al respecto.
- El **método hipotético deductivo** se utilizará para a partir del planteamiento de la hipótesis de la investigación y siguiendo reglas de deducción poder llegar a un nuevo conocimiento.
- El **método empírico** que se utilizará en la presente investigación es la observación.
- El **método de observación** se utilizará para realizar una evaluación de la situación problemática en cuestión.

En la presente tesis el contenido está estructurado por tres capítulos que se muestran a continuación:

CAPÍTULO 1. Fundamentación Teórica: Se describen los conceptos asociados al dominio del problema a resolver, siendo estos esenciales para entender el entorno del mismo. Se presenta el estado del arte de las técnicas implicadas en el objeto de estudio y el conjunto de tecnologías involucradas en el desarrollo de la propuesta.

CAPÍTULO 2. Propuesta de Solución: Se exponen los procesos de negocios y

requisitos a cumplir por la herramienta además de los artefactos generados durante el diseño de la misma.

CAPÍTULO 3. Implementación y Prueba: Se exponen los artefactos generados durante la implementación de la solución así como las métricas y pruebas utilizadas para la validación de la misma.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

El presente capítulo constituye el marco teórico de la investigación a realizar. En él se define una serie de conceptos necesarios para comprender el objetivo fundamental del trabajo, profundizando en temas como la creación de componentes en los marcos de trabajo. Además se ofrece una visión de la metodología, tecnologías y las herramientas que se utilizarán durante el desarrollo de la solución.

1.2. Marco teórico

1.2.1. Aplicaciones Web de Gestión.

Las aplicaciones web son hoy en día uno de los activos principales de cualquier empresa. Debido a que dan servicio a usuarios finales (por ejemplo aplicaciones de *e-banking* o *e-commerce*) así como también a aplicaciones corporativas (*una intranet*), las cuales manejan información cuya confidencialidad, disponibilidad e integridad es fundamental para las empresas. (2)

En otros términos, una aplicación web es un sistema que permite a los usuarios ejecutar lógica de negocio a través de un navegador. El protocolo principal de comunicación es HTTP donde el cliente hace una petición al servidor, este la procesa y le devuelve el resultado, terminando la comunicación entre ellos.

Las ventajas más significativas de las aplicaciones web son:

Compatibilidad Multiplataforma: una misma versión de la aplicación puede correr sin problemas en múltiples plataformas como Windows, Linux, Mac, etc.

Actualización: las aplicaciones web siempre se mantienen actualizadas y no requieren que el usuario deba descargar actualizaciones y realizar tareas de instalación.

Acceso inmediato y desde cualquier lugar: las aplicaciones basadas en tecnologías web no necesitan ser descargadas, instaladas y configuradas. Además pueden ser accedidas desde cualquier computadora conectada a la red en donde se accede a la aplicación.

Seguridad en los datos: los datos se alojan en servidores con sistemas de almacenamiento altamente fiables y se ven libres de problemas que comúnmente

sufren los ordenadores de usuarios comunes como virus y roturas de disco. (3).

El marco de trabajo Sauxe desarrollado por el CEIGE permite crear aplicaciones web con fines empresariales. Esto garantiza agilidad en el proceso de desarrollo de los componentes que se desarrollan dentro del marco de trabajo.

1.3. Sauxe como marco de trabajo para Aplicaciones Web.

Características de Sauxe

Sauxe como cualquier marco de trabajo (en inglés Framework) en el desarrollo de software, brinda una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. (4) Este marco de trabajo, fusionado bajo tecnología totalmente libre (entre ellas PHP, PostgreSql, Apache) posee el desarrollo de tecnologías propias basadas en otros marcos de trabajo como ZendFramework para el manejo de la lógica de negocio, Doctrine para el acceso a datos y ExtJS para la capa de presentación.

Cuenta con una arquitectura en capas como se muestra en la ilustración 1 que a su vez presenta en su capa superior un MVC[1]. Contiene un conjunto de componentes reutilizables que provee la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo. Siguiendo el paradigma de independencia tecnológica por el cual apuesta el país, reutiliza las siguientes tecnologías libres (5):

Ilustración 1. Arquitectura de Sauxe.

[1] **(MVC).Modelo Vista Controlador** es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Con el desarrollo y reutilización de Sauxe se ahorran cuantiosos recursos humanos y materiales. Para Cuba adquirir un marco de trabajo de este tipo en el mercado internacional resultaría costoso y si se decide implementar un sistema que garantice cada uno de los escenarios arquitectónicos que debe soportar cada una de las aplicaciones que se desarrollen implicaría un gasto de millones de dólares innecesariamente. Además no se lograría estandarizar y agilizar el proceso de desarrollo y las aplicaciones no estarían centradas en los requerimientos del usuario sino en la tecnología.

Al iniciar un desarrollo sobre Sauxe, se tiene garantizado aspectos como la seguridad, la multi-entidad, el multi-tema, el multi-idioma, la auditoría, la integración, la interoperabilidad, la concurrencia, la administración de transacciones, entre otros. Una de las características más importantes de Sauxe es que está orientado a componentes. Este enfoque posibilita alcanzar un mayor nivel de reutilización de las soluciones, así como una independencia en la ejecución de las pruebas a los componentes minimizando las afectaciones a otras partes del sistema y el mejoramiento de la calidad de la aplicación pues un componente como una unidad de composición de aplicaciones puede ser construido y perfeccionado continuamente. La programación orientada a componentes permite reducir el costo a lo largo de la etapa de mantenimiento, esto es un factor esencial en la mayoría de los negocios, en los cuales se está extendiendo el uso de la tecnología de componentes. (6).

1.3.1. Componentes en Sauxe.

En Sauxe un componente representa una abstracción de los procesos de negocio que se modelan. Siendo esto consecuente con la definición que propone la IEEE según el Instituto de Ingeniería de Software (*SEI, del inglés Software Engineering Institute*), que define que un componente, en esencia, es una unidad reutilizable que puede interoperar con otros módulos software por medio de sus interfaces, las cuales define desde donde se puede tener acceso a los servicios que este ofrece a los demás componentes. (7).

En la especificación UML (Lenguaje Unificado de Modelado), un componente es una unidad modular con interfaces bien definidas, que es reemplazable dentro del

contexto. Un componente define su comportamiento en términos de interfaces provistas y requeridas; y dicho componente será totalmente reemplazable por otro que cumpla con las interfaces declaradas. (8) A consideración del autor un componente no es más que un software reutilizable que proporciona servicios a otros componentes mediante un conjunto de interfaces. Un componente en Sauxe tiene la siguiente estructura:

Estructura asociada al código fuente de las aplicaciones en Sauxe

En la ilustración 2 se pueden observar las dos primeras carpetas que se crean en la estructura de los directorios, que son necesarias para la creación de un módulo. En la ilustración 3 y 4 se muestran los directorios ubicados en la carpeta Web y Apps respectivamente. Estas carpetas son creadas manualmente donde se debe tener mucho cuidado en la ubicación y nombre de estos componentes.

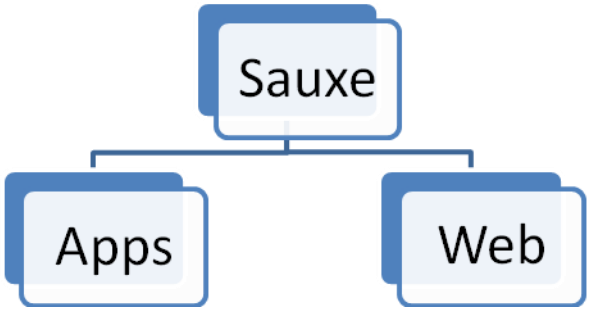

Ilustración 2. Estructura asociada al código fuente de las aplicaciones en Sauxe.

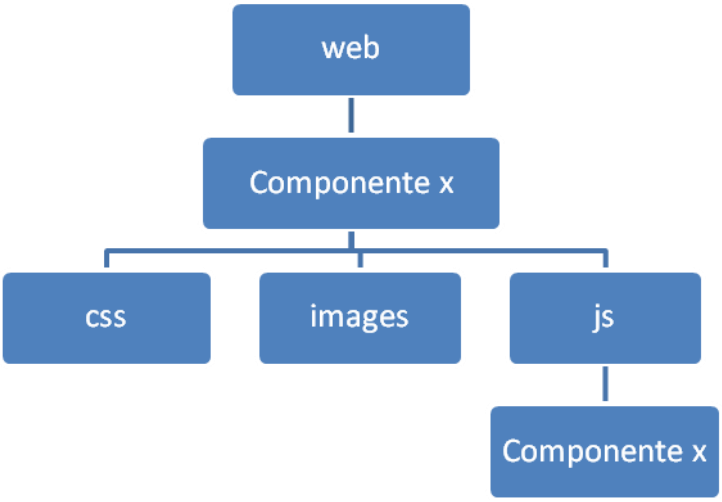

Ilustración 3. Estructura de un componente en el directorio web.

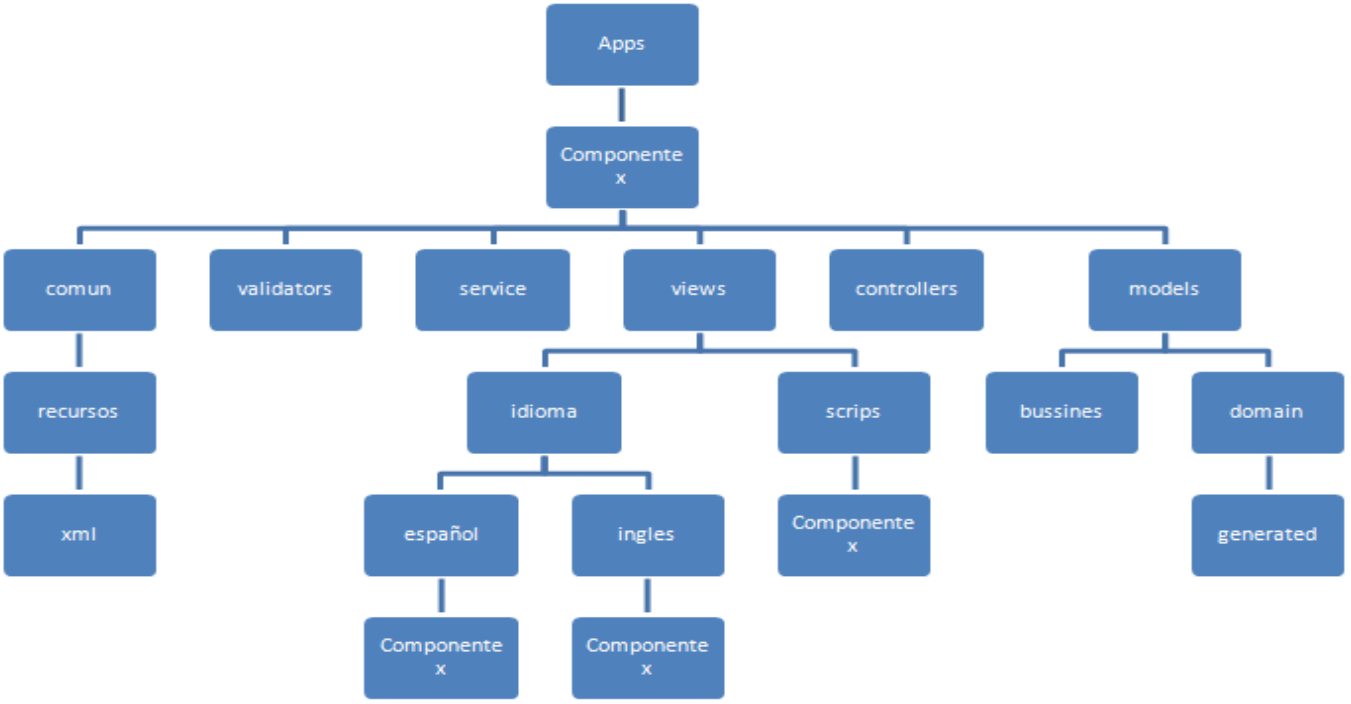

Ilustración 4. Estructura de un componente en el directorio apps.

Estructura de un componente en el directorio apps y web

En este paso son creados y ubicados los ficheros en sus respectivos directorios. En las ilustraciones 4 y 5 se parte desde la carpeta web y apps en el mismo orden en que son mencionadas, perteneciendo al segundo nivel del árbol de directorios en el cual su raíz sería la carpeta Sauxe que no es mostrada. Al crear los ficheros

debemos agregar un código inicial en algunos de ellos, la parte del código sería igual para los ficheros que forman parte del esqueleto del marco de trabajo Sauxe.

Sauxe, a pesar de las ventajas que posee en el desarrollo de aplicaciones web, no cuenta con un generador dinámico de componentes. Esto trae consigo que no se puedan generar componentes dinámicos, haciendo engorroso el trabajo de los desarrolladores. Es por eso que se realizará un estudio de las herramientas afines a la generación dinámica de componentes con el fin de garantizar una estructura para el marco de trabajo Sauxe que permita generar componentes dinámicos.

1.4. Estado del Arte

En este epígrafe se hará un estudio y análisis de alguna de las herramientas existentes que generan código, así como de los marcos de trabajo más usados que igualmente realicen esta función.

1.4.1. Lenguaje para describir Arquitectura

Lenguaje descriptivo de modelado arquitectónico de software que se focaliza en la estructura de alto nivel de la aplicación antes que en los detalles de implementación de sus módulos concretos. (9). Existen varios ADL para describir arquitecturas, dentro de los más importante se pueden señalar los siguientes: ACME, Wright, Jacal, Unicon, Rapide, Darwin, de estos se analizarán Unicon, Rapide y Darwin porque son los que generan código fuente. A continuación se muestran algunas características de estos.

1.4.1.1. Unicon

Se define como un ADL de propósito general, haciendo énfasis en conectores y estilos (Unicon). Permite generar código ejecutable a partir de una descripción, a partir de componentes primitivos adecuados. Se desataca por su capacidad de manejo de métodos, de análisis de tiempo real a través de RMA (*Rate Monotonic Analysis,* análisis de tasa monotónica). Entre sus principales características sobresalen las interfaces y la forma de generar código C mediante el procedimiento de asociar elementos arquitectónicos. Posee un entorno gráfico

para Windows, que además de modelar permite generar código en lenguaje C. Este ADL se destaca además por su capacidad de manejo de métodos de análisis de tiempo real a través de RMA. Es importante señalar este ADL no tiene una base formal que posibilite la validación de propiedades de las especificaciones, por otra parte no es posible la reutilización de componentes o arquitecturas, ni su adaptación a cambios en los requisitos y la descripción de sistemas dinámicos. (10)

1.4.1.2. Rapide

Se define como el diseño de un lenguaje para la producción de prototipos de sistemas, con herramientas de soporte para la simulación y análisis. Su principal objetivo radica en ayudar en el desarrollo de sistemas distribuidos, multilenguaje y de gran tamaño, así como en el análisis riguroso de grandes sistemas. Se basa en los conceptos de interfaz y patrones de conexión. Corre bajo Linux. La estructura de Rapide es sumamente compleja y en realidad articula cinco lenguajes: el lenguaje de tipos describe las interfaces de los componentes; el lenguaje de arquitectura describe el flujo de eventos entre componentes; el lenguaje de especificación describe restricciones abstractas para la conducta de los componentes; el lenguaje ejecutable describe módulos ejecutables; y el lenguaje de patrones describe patrones de los eventos. Los diversos sub-lenguajes comparten la misma visibilidad, scoping (*proceso rápido y abierto para determinar el alcance de las acciones*) y reglas de denominación, así como un único modelo de ejecución. Rapide define tipos de componentes (*llamados interfaces*) en términos de una colección de eventos de comunicación que pueden ser observados (*acciones externas*) o iniciados (*acciones públicas*). Las interfaces de Rapide definen el comportamiento computacional de un componente vinculando la observación de acciones externas con la iniciación de acciones públicas. Rapide puede generar código C, C++ y Ada. (11)

1.4.1.3. Darwin

Se define como un lenguaje de descripción arquitectónica. Su objetivo está orientado al diseño de arquitecturas dinámicas y cambiantes. Darwin describe un

tipo de componente mediante una interfaz consistente en una colección de servicios que son provistos (*declarados por ese componente*) o requeridos (*es decir, que se espera ocurran en el entorno*). Las configuraciones se desarrollan instanciando las declaraciones de componentes y estableciendo vínculos entre ambas clases de servicios. Darwin soporta desarrollos escritos en C++, aunque no presupone que los componentes de un sistema real estén programados en algún lenguaje en particular. Darwin carece de la capacidad de definir nuevos tipos, soportando sólo una amplia variedad de tipos de servicio predefinidos. Darwin presupone que la colección de tipos de servicio es suministrada por la plataforma para la cual se desarrolla una implementación, y confía en la existencia de nombres de tipos de servicio que se utilizan sin interpretación, sólo verificando su compatibilidad. (12)

Después del estudio realizado de los principales ADL, haciendo énfasis en los que generan código fuente, se pudo llegar a la conclusión que el más cercano a la solución que se propone es el ADL Rapide, puesto que define tipos de componentes (*llamados interfaces*) en términos de una colección de eventos de comunicación que pueden ser observados (*acciones externas*) o iniciados (*acciones públicas*). Las interfaces de Rapide definen el comportamiento computacional de un componente vinculando la observación de acciones externas con la iniciación de acciones públicas. Sin embargo el ADL Rapide genera el código fuente asociado a los componentes especificados en los lenguajes C, C++ y Ada, lo que constituye un factor que imposibilita anexarlo como solución para la generación dinámica de componentes al marco de trabajo Sauxe.

1.4.2. Marcos de trabajo

Un marco de trabajo es una estructura de archivos y utilidades que aceleran la programación de una aplicación informática, proveyendo una metodología de trabajo que sistematiza y facilita la generación de formularios, funciones y módulos de uso común, permitiendo al desarrollador dedicar su atención hacia los aspectos específicos de cada aplicación. Dentro de estos marcos de trabajo existentes se

hizo un estudio de los marcos de trabajo ZendFramework y Symfony, a continuación se explican las características de estos marcos de trabajo.

1.4.2.1. ZendFramework

La estructura de los componentes de Zend Framework es algo único; cada componente está construido con una baja dependencia de otros componentes. Esta arquitectura débilmente acoplada permite a los desarrolladores utilizar los componentes por separado.

A menudo se refiere a este tipo de diseño como "use-at-will" (*uso a voluntad*). Aunque se pueden utilizar de forma individual, los componentes de la biblioteca estándar de ZendFramework conforman un potente y extensible marco de trabajo de aplicaciones web al combinarse. (13) A pesar de permitir crear componentes de manera independiente no presenta facilidades para crearlos de manera dinámica.

1.4.2.2. Symfony

Marco de trabajo diseñado para optimizar, gracias a sus características, el desarrollo de las Aplicaciones web. Para empezar, separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. El resultado de todas estas ventajas es que no se debe reinventar la rueda cada vez que se crea una nueva aplicación web. (14). A pesar de sus ventajas no permite que la estructura para la creación dinámica de componentes se monte sobre otra aplicación o marco de trabajo, es por eso que se hace necesario desarrollar una solución sobre el marco de trabajo Sauxe que permita generar componentes de forma dinámica y que pueda incluirse la estructura de estos componentes en otra solución. Para darle solución a este problema se realizó el estudio de varias herramientas que posibilitaron el desarrollo de la solución.

1.5. Herramientas utilizadas

1.5.1. Herramientas CASE: Visual Paradigm

Se empleará Visual Paradigm 3.4 como herramienta CASE. Utiliza UML 6.0 como lenguaje de modelado, con soporte multiplataforma y que proporciona excelentes facilidades de interoperabilidad con otras aplicaciones. Permite generar diagramas, entre ellos, diagramas de clases, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Ayudando a construir aplicaciones de calidad más rápido, mejor y a bajo costo. (15)

1.5.2. Lenguaje Unificado de Modelado (UML)

El Lenguaje de Modelado Unificado (*Unified Modeling Language*) como notación orientada a objetos, se empleará con el fin de especificar y documentar un sistema de software, de un modo estándar incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema. UML implementa un lenguaje de modelado común para todos los desarrollos por lo que se crea una documentación también común, que cualquier desarrollador con conocimientos de UML será capaz de entender, independientemente del lenguaje utilizado para el desarrollo. (16)

1.5.3. ZendStudio Neon

La solución que se propone se desarrollará sobre el IDE, el cual es un completo entorno integrado de desarrollo para el lenguaje de programación PHP. Está escrito en Eclipse, y disponible para las plataformas Microsoft Windows, Mac OS X y GNU/Linux. Está diseñado para usarse con el lenguaje PHP; sin embargo ofrece soporte básico para otros lenguajes Web, como HTML, Javascript y XML (17).

Sus principales características son:

- No requiere la instalación previa de PHP ni del entorno de ejecución de Eclipse.
- Soporte para PHP 4 y PHP 5.

- Resaltado de sintaxis, autocompletado de código, ayuda de código y lista de parámetros de funciones y métodos de clase.
- Inserción automática de paréntesis y corchetes de cierre.
- Emparejamiento de paréntesis y corchetes.
- Detección de errores de sintaxis en tiempo real.
- Funciones de depuración.
- Manual de PHP integrado.
- Soporte para navegación en bases de datos y ejecución de consultas SQL.2.

1.5.4. Servidor web: Apache

Se utilizará como servidor web Apache 2.0 pues es una tecnología gratuita de código abierto compatible con muchos Sistemas Operativos. Tiene todo el soporte que se necesita para tener páginas dinámicas. Permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor. Tiene una alta configurabilidad en la creación y gestión de registros de actividad. Apache permite la creación de ficheros de registro a medida del administrador, de este modo se puede tener un mayor control sobre lo que sucede en el servidor. (18)

1.5.5. Sistema de Control de Versiones: Subversion

La versión 1.6.5 de Subversion (*SVN*) es la herramienta de entorno colaborativo que se utilizará para el control de versiones. Este se encuentra preparado para funcionar en red y se distribuye bajo licencia libre. Mantiene versiones no sólo de archivos, sino también de directorios y versiones de los metadatos asociados a esos directorios. Además de los cambios en el contenido de los documentos, se mantiene el historial de todas las operaciones de cada elemento, incluyendo la copia, cambio de directorio o de nombre. Ofrece mejor uso del ancho de banda, ya que en las transacciones se transmiten sólo las diferencias y no los archivos completos. (19)

1.5.6. Mozilla Firefox

Se utilizará como navegador web Mozilla Firefox 3.6.10. Este navegador

multiplataforma es libre y es compatible con los estándares web (señálese HTML, XML, CSS y JavaScript) que se emplearán para el desarrollo a la solución que se propone. Incluye entre sus funcionalidades un mecanismo para añadir funcionalidades mediante extensiones. Precisamente Firebug es el complemento que se integrará a Mozilla Firefox para ayudar a desarrollar, evaluar y depurar la aplicación, controlando el CSS y HTML en tiempo real, midiendo el tiempo de carga para optimizar la página o corrigiendo los posibles inconvenientes con JavaScript.

1.6. Modelo de desarrollo utilizado.

Para el desarrollo de esta herramienta se utilizó un modelo de desarrollo orientado a componentes, definido por el Centro de Informatización de Gestión de Entidades (CEIGE). Este es un modelo de desarrollo orientado a las necesidades y artefactos generados durante el proceso de desarrollo de cualquier producto en el centro CEIGE. (20) Es una combinación de diferentes metodologías de las cuales se ha tomado lo que sería más conveniente para llevar a término un proyecto. Entre sus características más importantes se tienen las siguientes:

Centrado en la arquitectura

La arquitectura determina la línea base, los elementos de software estructurales a partir de los elementos de la arquitectura de negocio. Interviene en la gestión de cambios y diseña la evolución e integración del producto. La arquitectura orienta las prioridades del desarrollo y resuelve las necesidades tecnológicas y de soporte para el desarrollo.

Orientado a componentes

Las iteraciones son orientadas por el nivel de significancia arquitectónicas de los componentes, los mismos son abstracciones arquitectónicas de los procesos de negocio y requisitos asociados que modelan, el componente es la unidad de medición y ordenamiento de las iteraciones.

Iterativo e incremental

Las iteraciones son planificadas y coordinadas con el equipo de arquitectura, los clientes y la alta gerencia. Cada iteración constituye el desarrollo de componentes, los cuales son integrados al término de la iteración, permitiendo de esta manera la

evolución incremental del producto.

Ágil y adaptable al cambio

El desarrollo de las partes formaliza solamente las características principales de la solución, priorizando los talleres y las comunicaciones entre las personas. Los clientes y funcionales están involucrados en el proyecto y poseen parte de la responsabilidad del éxito del mismo. Los cambios son conciliados semanalmente, discutidos y aprobados.

Este modelo de desarrollo permitirá la generación de artefactos de vital importancia en el análisis y el diseño como son:

- Modelo de proceso de negocio.
- Descripción de procesos de negocio.
- Modelo conceptual.
- Prototipo de interfaz de usuario.
- Especificación de requisitos.
- Modelo de datos.
- Diagrama de clases.
- Descripción del diseño de clases.
- Diagrama de componente.
- Diagrama de despliegue.
- Casos de prueba.

1.7. Conclusiones parciales.

Este capítulo fue significativo para la definición de un conjunto de conceptos fundamentales asociados al dominio del problema. Se analizaron las tendencias y tecnologías actuales para el desarrollo de aplicaciones informáticas permitiendo situar la solución que se propone en el marco teórico actual. Se realizó una presentación de las tecnologías y herramientas, propuestas por la dirección del proyecto para el desarrollo de la solución. Se identificaron los elementos esenciales del marco de trabajo Sauxe haciendo énfasis en su concepción de componente para el posterior diseño e implementación de la solución que se propone. Se identificó el modelo de desarrollo que permitirá crear la solución atendiendo a sus características. Se identificaron las herramientas afines para

usar en el desarrollo del problema planteado dando paso a la propuesta de solución.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

2.1. Introducción

La Arquitectura de software es una práctica joven de apenas unos pocos años de trabajo constante, sin embargo ya se ha convertido en un factor de vital importancia para lograr que los sistemas de software tengan un alto nivel de calidad. Poseer una buena Arquitectura de software es de suma importancia, ya que ésta es el corazón de todo sistema de software y determina cuáles serán los niveles de calidad asociados al sistema. El siguiente capítulo recoge los artefactos generados con la utilización del modelo de desarrollo utilizado. Primeramente son descritos los procesos de negocio relativos al campo de acción, luego los escenarios arquitectónicos con el fin de describir los escenarios con sus requisitos asociados, le continúa el modelo conceptual con el objetivo de esclarecer los conceptos fundamentales con los que se trabajarán en el desarrollo de la aplicación y seguido los requisitos funcionales y no funcionales de la solución para lograr un entendimiento de lo que se quiere lograr. Posteriormente se realiza una descripción del diseño elaborado para alcanzar las metas trazadas, además de la estrategia a seguir durante el proceso de implementación.

2.2. Descripción de los Procesos de Negocio

La descripción de los procesos de negocio facilita las actividades del análisis ya que hace posible el entendimiento de los procesos en cuestión para así definir con mayor facilidad los requisitos que cumplirán las necesidades del usuario.

2.3. Descripción del proceso: Gestionar Componentes y/o Paquete de Componentes.

En este proceso surge por la necesidad de gestionar un nuevo componente o paquete de componente. Cuando se crea un paquete de componente se crean en los *directorios web* y *apps* asociadas a este, así como los archivos y ficheros correspondientes al paquete de componente creado. Este proceso se diferencia del proceso gestionar componentes porque además de los directorios y ficheros que crea el paquete de componentes, el componente tiene más directorios y

ficheros relacionados con los requisitos y funcionalidades.

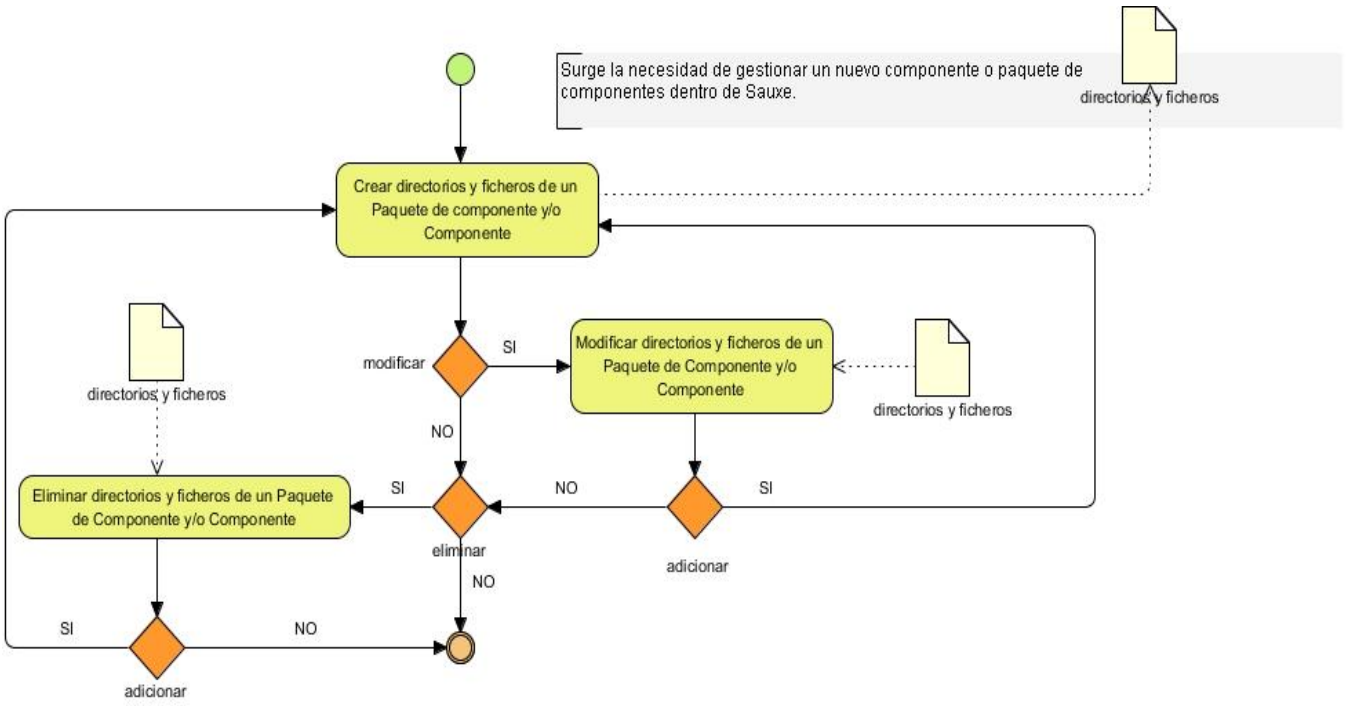

Ilustración 5. Diagrama del proceso de negocio Gestionar Paquetes de Componentes y/o Componentes.

2.3.1.1. Descripción textual del Proceso de Negocio Gestionar Componente y Paquete de Componente.

Objetivo	Crear directorios y ficheros de un Paquete de Componente y/o Componente.
Evento(s) que lo genera(n)	Surge la necesidad de gestionar un nuevo componente y/o paquete de componente dentro del marco de trabajo Sauxe.
Pre condiciones	Realizar la configuración inicial del sistema.
Marco legal	N/A.
Clientes internos	Gestionar Funcionalidad.
Clientes externos	N/A.

Entradas	N/A
Flujo de eventos	
Flujo básico Crear directorios y ficheros de un paquete de componente y/o componente.	
1	Crear directorios y ficheros de un paquete de componente y/o componente. En esta actividad se crean los directorios y ficheros relacionados con el componente o el paquete de componente según lo que se creó. Después de realizar esta operación se puede modificar, eliminar ó salir según lo que se desee.
Flujos alternos	
Flujo alternativo 1.a Modificar directorios y ficheros de un paquete de componente y/o componente.	
1.	Modificar directorios y ficheros de un paquete de componente y/o componente. En esta actividad se modifican los directorios y ficheros asociados al componente o paquete de componente. Después de realizar esta operación se puede eliminar, salir o ir al flujo básico, según lo que se desee.
Flujo alternativo 2. a Eliminar directorios y ficheros de un paquete de componente y/o componente.	
1.	Eliminar directorios y ficheros de un paquete de componente y/o componente. En esta actividad se eliminan los directorios y ficheros asociados al componente o paquete de componente. Después de realizar esta operación se puede ir al flujo básico, o al flujo alternativo 1.a o salir.

2.3.2. Descripción del proceso: Gestionar Funcionalidad.

En este proceso está asociado a los componentes que son los únicos que pueden gestionar funcionalidades. Cuando se gestiona una funcionalidad en la carpeta del componente al cual se le gestionará dicha funcionalidad se crea una carpeta con el nombre de la funcionalidad, además un xml con la información de la funcionalidad, es decir: nombre, abreviatura y descripción.

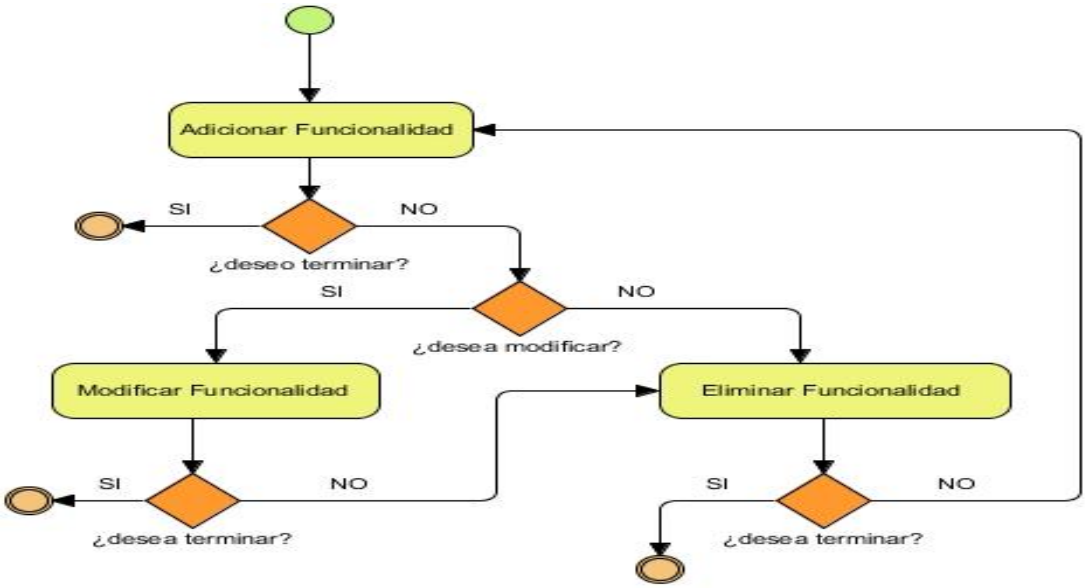

Ilustración 6.Descripción del Proceso Gestionar Funcionalidad

Objetivo	Crear directorios y ficheros de un Paquete de Componente y/o Componente.
Evento(s) que lo genera(n)	Surge la necesidad de gestionar un nuevo componente y/o paquete de componente dentro del marco de trabajo Sauxe.
Pre condiciones	Realizar la configuración inicial del sistema.
Marco legal	N/A.
Clientes internos	Gestionar Componente y Paquete de Componente.
Clientes externos	N/A.
Entradas	N/A
Flujo de eventos	
Flujo básico Adicionar Funcionalidad	
2 Adicionar Funcionalidad. En esta actividad se adiciona una o más funcionalidades relacionadas con el componente. Después de realizar esta operación se puede modificar, eliminar ó salir según lo que se desee.	
Flujos alternos	
Flujo alternativo 1.a Modificar funcionalidad de componente.	
2. Modificar Funcionalidad. En esta actividad se modifican la (las)	

	funcionalidades asociadas al componente. Después de realizar esta operación se puede eliminar, salir o ir al flujo básico según lo que se desee.
Flujo alternativo 2. a Eliminar funcionalidad al componente.	
2.	Eliminar Funcionalidad. En esta actividad se elimina la (las) funcionalidades asociadas al componente. Después de realizar esta operación se puede ir al flujo básico, o al flujo alternativo 1.a o salir.

2.4. Modelo Conceptual

Un paquete de componente está compuesto por uno o muchos componentes. Un componente a su vez está compuesto por uno o muchos requisitos y una o muchas funcionalidades, como se muestra en la siguiente figura.

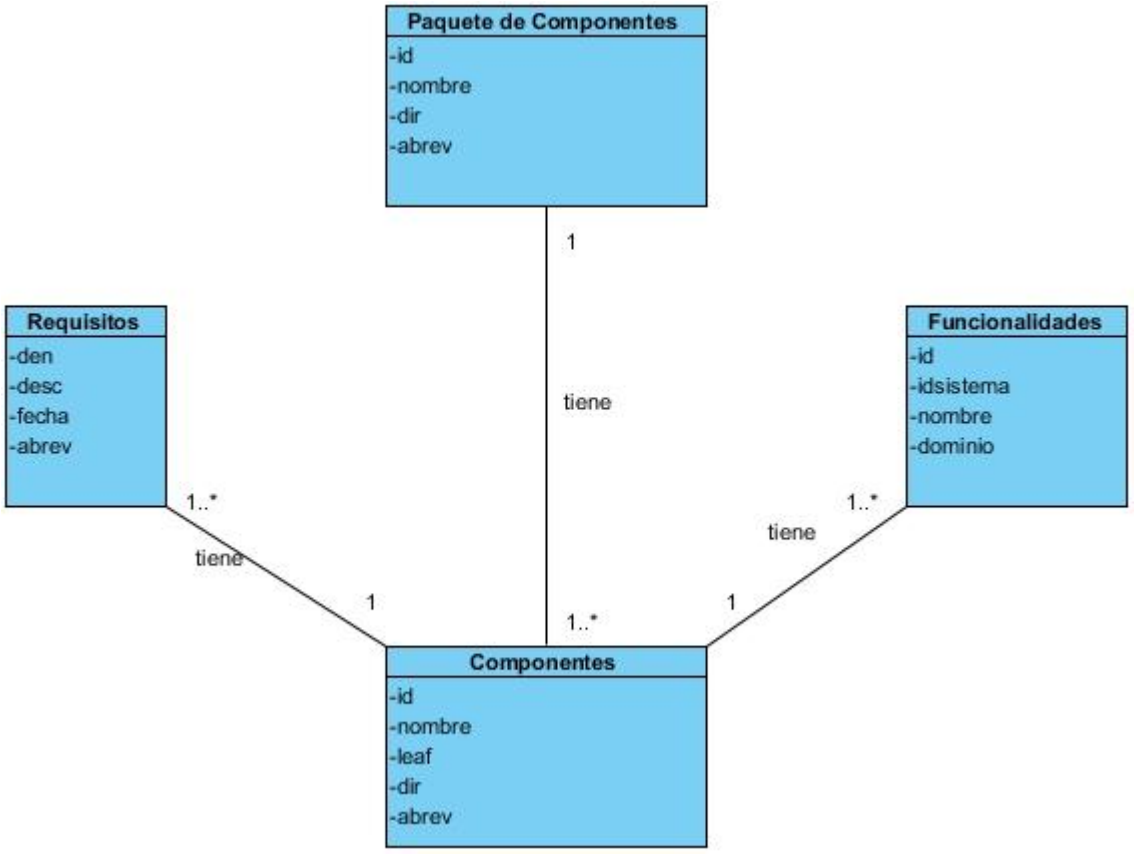

Ilustración 7. Modelo Conceptual

2.5. Requisitos de la Solución Propuesta

Una vez que se realiza el modelado del negocio se puede comenzar a ejecutar el proceso de captura de requisitos del sistema mediante las entrevistas con el cliente, esta es una de las actividades fundamentales que se realiza en el proceso

de desarrollo de software. Un requisito es una condición que tiene que ser alcanzada por un sistema o componente software para satisfacer un contrato, estándar, u otro documento impuesto formalmente. Los mismos pueden dividirse en requisitos funcionales y requisitos no funcionales.

Los requisitos funcionales son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requisitos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer. (21).

2.5.1. Requisitos Funcionales

RF1 Gestionar Componentes o Paquetes de Componentes.

RF1.1 Adicionar componente o paquete de componente.

RF1.2 Modificar componente o paquete de componente.

RF 1.3 Eliminar componente o paquete de componente.

RF 1.4 Listar componente o paquete de componente

RF2 Gestionar requisitos de Componente.

RF 2.1 Adicionar requisitos de Componente.

RF 2.2 Modificar requisitos de Componente.

RF 2.3 Eliminar requisitos de Componente.

RF 2.4 Listar requisitos de Componente

RF3 Gestionar funcionalidades del Componente.

RF 3.1 Adicionar funcionalidad de Componente.

RF 3.2 Modificar funcionalidad de Componente.

RF 3.3 Eliminar funcionalidad de Componente.

RF 3.4 Listar funcionalidad de Componente

2.5.2. Especificación del requisito Adicionar componente.

2.5.2.1. Descripción textual del requisito

Precondiciones	Debe haberse seleccionado un paquete de componentes.
Flujo de eventos	

Flujo básico Adicionar componente	
3	El usuario selecciona el botón adicionar.
4	El usuario introduce la descripción.
5	El usuario selecciona la opción Componente y selecciona el botón aceptar.
6	El sistema crea dentro del paquete de componentes una carpeta con el nombre apps.
7	El sistema crea dentro del paquete de componentes una carpeta con el nombre web.
8	El sistema crea dentro de la carpeta apps del componente la carpeta común la cual contiene los xml del componente.
9	El sistema crea dentro de la carpeta apps del componente la carpeta controllers donde se almacenan todas las clases controladoras del componente.
10	El sistema crea dentro de la carpeta apps la carpeta models que contiene las carpetas domain donde se programan las consultas y business se programan las modificaciones, como por ejemplo los métodos invocados desde la clase controladora.
11	El sistema crea dentro de la carpeta apps la carpeta views que es donde se recopilan los ficheros que van a gestionar la capa de presentación, estos son idioma y scripts. Dentro de la carpeta idioma existirán dos subcarpetas, una "es" donde se almacenan los archivos que gestionan una presentación en español como por ejemplo ficheros de tipo json que recopilan etiquetas para mostrar mensajes en el idioma correspondiente, y otra carpeta llamada "en" donde se gestiona la presentación en inglés. Dentro de la carpeta scripts se incluyen todas las vistas, para ello se crea una carpeta para cada clase controladora y dentro se incluye la vista o script. Estos no son más que archivos de extensión phtml donde se especifica el título de la página que se gestiona y se carga el archivo js que mostrará la presentación como tal.
12	El sistema crea dentro de la carpeta web, la carpeta con el nombre del componente, en este caso dentro de esta estarán css y js.
13	El sistema valida la descripción (ver validación 1).
14	Si el dato es correcto el sistema lo registra.
15	Concluye el requisito.

Pos-condiciones	
1	N/A
Flujos alternativos	
Flujo alternativo 3.1 El usuario selecciona la opción Componente	
1	El sistema adiciona un componente.
2	Ir al paso 4.
Pos-condiciones	
1	N/A
Pos-condiciones	
1	N/A
Flujo alternativo *.a El usuario cancela la acción	
1	Concluye el requisito.
Pos-condiciones	
1	No se adiciona el componente.
Validaciones	
1	La descripción del componente debe contener solo caracteres alfabéticos, sin espacios.
Conceptos	**Componente** Visibles en la interfaz: Descripción del componente
Requisitos especiales	N/A
Asuntos pendientes	N/A

Tabla 1. Especificación del requisito Adicionar Componente.

2.5.2.2. Prototipo elemental de interfaz gráfica de usuario

Ilustración 8. Adicionar Componente

2.5.3. Especificación del requisito Adicionar paquete de componente.

2.5.3.1. Descripción textual del requisito

Precondiciones	Debe haberse seleccionado un paquete de componentes.
Flujo de eventos	
Flujo básico Adicionar paquete de componente	
1	El usuario selecciona el botón adicionar.
2	El usuario introduce la descripción.
3	El usuario no selecciona la opción Componente y selecciona el botón aceptar.
4	El sistema crea dentro de la carpeta apps una carpeta con el mismo nombre de la descripción del paquete de componente creado.
5	El sistema crea dentro de la carpeta web una carpeta con el mismo nombre de la descripción del paquete de componente creado.
6	El sistema crea dentro de la carpeta apps del paquete de componente la carpeta común la cual contiene los xml del paquete de componente, las funcionalidades y los requisitos del componente.
7	El sistema crea dentro de la carpeta apps del componente la carpeta controllers donde se almacenan todas las clases controladoras del componente.
8	El sistema crea dentro de la carpeta apps la carpeta models que contiene las carpetas domain donde se programan las consultas y business se programan las modificaciones, como por ejemplo los métodos invocados desde la clase

	controladora.
9	El sistema valida la descripción (ver validación 1).
10	Si el dato es correcto el sistema lo registra.
11	Concluye el requisito.
Pos-condiciones	
2	N/A
Flujos alternativos	
Flujo alternativo 3.1 El usuario no selecciona la opción Componente	
1	El sistema adiciona un paquete de componente.
2	Ir al paso 3.
Pos-condiciones	
1	N/A
Flujo alternativo 3.2 El usuario no selecciona la opción Componente	
1	El sistema adiciona un paquete de componentes.
2	Ir al paso 4.
Pos-condiciones	
1	N/A
Flujo alternativo *.a El usuario cancela la acción	
2	Concluye el requisito.
Pos-condiciones	
2	No se adiciona el paquete de componente.
Validaciones	
2	La descripción del paquete de componente debe contener solo caracteres alfabéticos, sin espacios.
Conceptos	**Componente** Visibles en la interfaz: Descripción del paquete de componente
Requisitos especiales	N/A
Asuntos pendientes	N/A

Tabla 2. Especificación del requisito Adicionar Paquete de Componente

2.4.3.2. Prototipo elemental de interfaz gráfica de usuario

Ilustración 9. Adicionar Paquete de Componente

2.5.4. Especificación de requisito Modificar Componente.

2.5.4.1. Descripción textual del requisito

Precondiciones	Debe haberse seleccionado un componente.
Flujo de eventos	
Flujo básico Modificar componente	
1	El usuario selecciona la opción Modificar.
2	El sistema permite modificar la descripción del componente seleccionado.
3	El usuario introduce la descripción del componente.
4	El sistema valida la descripción del componente (ver validación 1)
5	El sistema modifica dentro de la carpeta apps el nombre del componente que se modificó.
6	El sistema modifica dentro de la carpeta web el nombre del componente que se modificó.
7	Si el dato es correcto el sistema lo registra.
8	Concluye el requisito.
Pos-condiciones	
1	N/A
Flujos alternativos	
Flujo alternativo *.a El usuario cancela la acción	

1	Concluye el requisito.
Pos-condiciones	
1	No se modifica el componente.
Validaciones	
3.	La descripción del componente debe contener solo caracteres alfabéticos, sin espacios.
Conceptos	**Componente** Visibles en la interfaz: Descripción del componente
Requisitos especiales	N/A
Asuntos pendientes	N/A

Tabla 3. Especificación del requisito Modificar Componente

2.5.4.2. Prototipo elemental de interfaz gráfica de usuario

Ilustración 10. Modificar Componente.

2.5.5. Especificación de requisito Modificar Paquete de Componente.

2.5.5.1. Descripción textual del requisito

Precondiciones	Debe haberse seleccionado un paquete de componente.
Flujo de eventos	
Flujo básico Modificar componente	

1	El usuario selecciona la opción Modificar.
2	El sistema permite modificar la descripción del paquete de componente seleccionado.
3	El usuario introduce la descripción del paquete de componente.
4	El sistema valida la descripción del paquete de componente (ver validación 1)
5	El sistema modifica dentro de la carpeta apps el nombre del componente que se modificó.
6	El sistema modifica dentro de la carpeta web el nombre del paquete de componente que se modificó.
7	Si el dato es correcto el sistema lo registra.
8	Concluye el requisito.
Pos-condiciones	
1	N/A
Flujos alternativos	
Flujo alternativo *.a El usuario cancela la acción	
2	Concluye el requisito.
Pos-condiciones	
2	No se modifica el paquete de componente.
Validaciones	
3.	La descripción del paquete de componente debe contener solo caracteres alfabéticos, sin espacios.
Conceptos	**Componente** Visibles en la interfaz: Descripción del paquete de componente
Requisitos especiales	N/A
Asuntos pendientes	N/A

Tabla 4.Especificación del requisito Modificar Paquete de Componente

2.5.5.2. Prototipo elemental de interfaz gráfica de usuario

Ilustración 11. Modificar Paquete de Componente

2.5.6. Especificación de requisito Eliminar Componente.

2.5.6.1. Descripción textual del requisito

Precondiciones	Debe haberse seleccionado un componente.
Flujo de eventos	
Flujo básico Eliminar componente	
1	El sistema elimina el componente.
2	El sistema confirma la eliminación.
3	El sistema elimina dentro de la carpeta apps la carpeta del componente que se eliminó con toda la información que este contenía.
4	El sistema elimina dentro de la carpeta web la carpeta del componente que se eliminó con toda la información que este contenía.
5	Concluye el requisito.
Pos-condiciones	
1	Se eliminó el componente seleccionado.
Flujos alternativos	
Flujo alternativo *.a El usuario cancela la acción	
1	Concluye el requisito.
Pos-condiciones	
1	No se elimina el componente.
Validaciones	

1	N/A	
Conceptos	**Componente**	Visibles en la interfaz: Descripción del componente
Requisitos especiales	N/A	
Asuntos pendientes	N/A.	

Tabla 5.Especificación del requisito Eliminar Componente

2.5.6.2. Prototipo elemental de interfaz gráfica de usuario

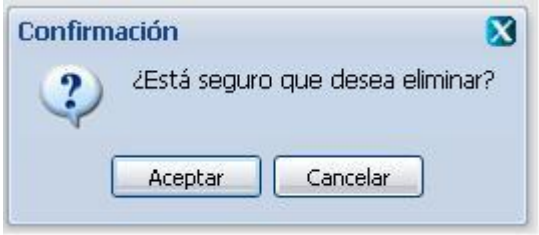

Ilustración 12. Eliminar Componente.

2.5.7. Especificación de requisito Eliminar Paquete de Componente.

2.5.7.1. Descripción textual del requisito

Precondiciones	Debe haberse seleccionado un paquete de componente.
Flujo de eventos	
Flujo básico Eliminar paquete de componente	
6	El sistema elimina el paquete de componente.
7	El sistema confirma la eliminación.
8	El sistema elimina dentro de la carpeta apps la carpeta del paquete de componente que se eliminó con toda la información que este contenía.
9	El sistema elimina dentro de la carpeta web la carpeta del paquete de componente que se eliminó con toda la información que este contenía.
10	Concluye el requisito.
Pos-condiciones	
2	Se eliminó el paquete de componente seleccionado.

Flujos alternativos		
Flujo alternativo *.a El usuario cancela la acción		
2	Concluye el requisito.	
Pos-condiciones		
2	No se elimina el paquete de componente.	
Validaciones		
1	N/A	
Conceptos	**Componente**	Visibles en la interfaz: Descripción del paquete de componente
Requisitos especiales	N/A	
Asuntos pendientes	N/A.	

Tabla 6.Especificación de requisito Eliminar Paquete de Componente

2.5.7.2. Prototipo elemental de interfaz gráfica de usuario

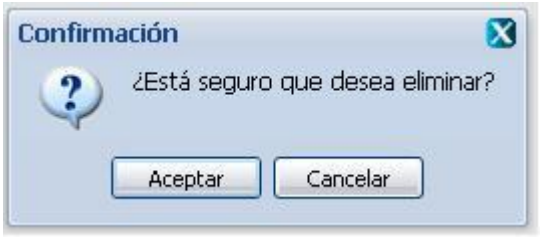

Ilustración 13. Eliminar Paquete de Componente.

2.5.8. Especificación de requisito Adicionar Requisito de Componente.

2.5.8.1. Descripción textual del requisito

Precondiciones	Debe haberse seleccionado un componente.
Flujo de eventos	
Flujo básico Gestionar requisito de componente	
1	El usuario selecciona el botón adicionar requisito.
2	El usuario introduce la denominación del requisito.
3	El usuario introduce la descripción del requisito.

4	El sistema crea dentro de la carpeta apps una carpeta con el mismo nombre del requisito de componente creado y lo registra en el xml del componente.
5	El sistema crea dentro de la carpeta web una carpeta con el mismo nombre del requisito de componente creado y crea el js del componente con el mismo nombre.
6	El sistema valida la descripción (ver validación 1).
7	Si el dato es correcto el sistema lo adiciona.
8	Concluye el requisito.
Pos-condiciones	
3	N/A
Flujos alternativos	
Flujo alternativo 3.1	
1	N/A.
2	
Pos-condiciones	
1	N/A
Flujo alternativo 3.2	
1	N/A
2	
Pos-condiciones	
1	N/A
Flujo alternativo *.a El usuario cancela la acción	
3.	Concluye el requisito.
Pos-condiciones	
3	No se adiciona el requisito al componente.
Validaciones	
4.	La descripción del requisito de componente debe contener solo caracteres alfabéticos, sin espacios.
Conceptos	**Componente** Visibles en la interfaz: Descripción del requisito de componente
Requisitos especiales	N/A

Asuntos pendientes	N/A

Tabla 7.Especificación de requisito Adicionar Requisito de Componente

2.5.8.2. Prototipo elemental de interfaz gráfica de usuario

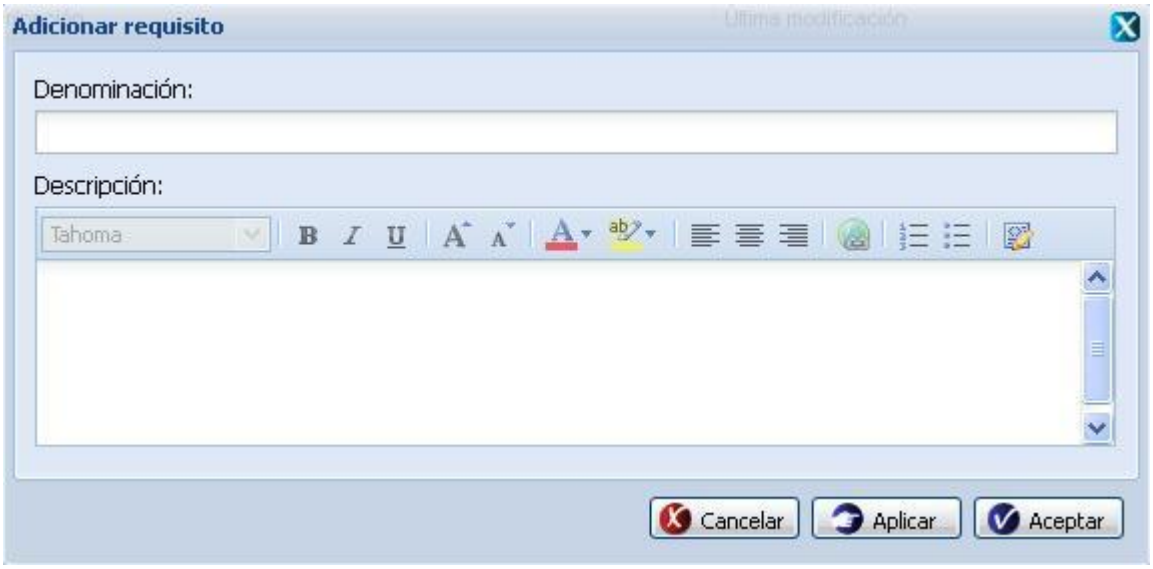

Ilustración 14. Adicionar Requisito.

2.5.9. Especificación de requisito Modificar Requisito de Componente.

2.5.9.1. Descripción textual del requisito

Precondiciones	Debe haberse seleccionado un requisito existente.
Flujo de eventos	
Flujo básico Gestionar requisito de componente	
1	El usuario selecciona el botón modificar requisito.
2	El usuario introduce la nueva denominación del requisito.
3	El usuario introduce la nueva descripción del requisito.
4	El sistema cambia dentro de la carpeta apps el nombre del requisito de componente que existía por el modificado por el usuario, así como el del xml del componente.
5	El sistema cambia dentro de la carpeta web el nombre del requisito de componente que existía por el modificado por el usuario y modifica además el js del componente con el nombre modificado.

6	El sistema valida la descripción (ver validación 1).
7	Si el dato es correcto el sistema lo modifica.
8	Concluye el requisito.
Pos-condiciones	
4	N/A
Flujos alternativos	
Flujo alternativo 3.1	
1	N/A.
2	
Pos-condiciones	
1	N/A
Flujo alternativo 3.2	
1	N/A
2	
Pos-condiciones	
1	N/A
Flujo alternativo *.a El usuario cancela la acción	
3.	Concluye el requisito.
Pos-condiciones	
4	No se modifica el requisito al componente.
Validaciones	
4.	La descripción del requisito de componente debe contener solo caracteres alfabéticos, sin espacios.
Conceptos	**Componente** Visibles en la interfaz: Descripción del requisito de componente
Requisitos especiales	N/A
Asuntos pendientes	N/A

Tabla 8.Especificación de requisito Modificar Requisito de Componente

2.5.9.2. Prototipo elemental de interfaz gráfica de usuario

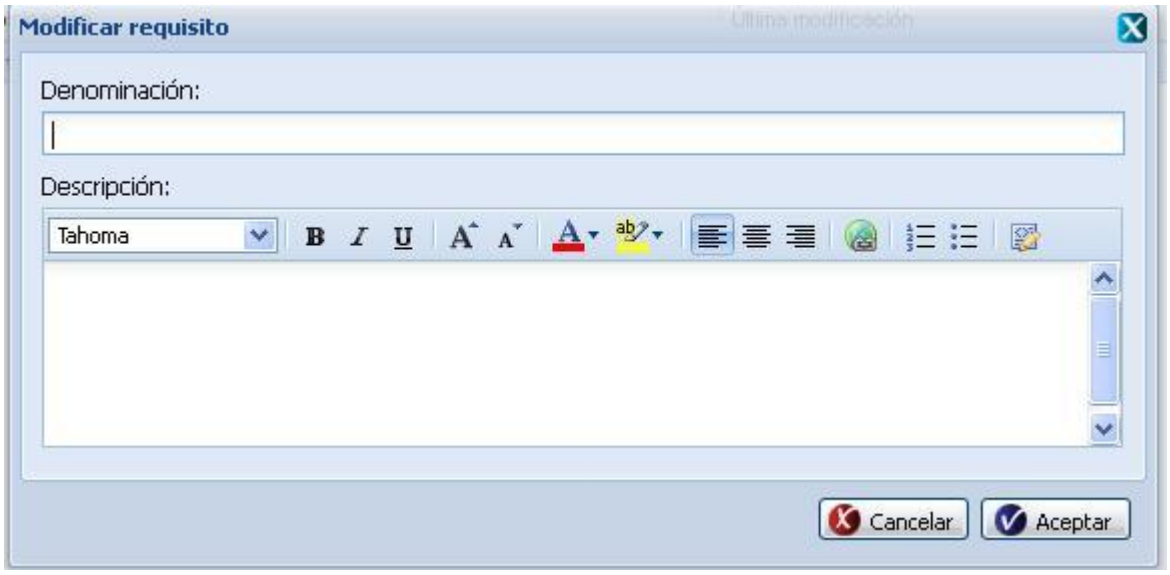

Ilustración 15. Modificar Requisito.

2.5.10. Especificación de requisito Eliminar Requisito de Componente.

2.5.10.1. Descripción textual del requisito

Precondiciones	Debe haberse seleccionado un requisito de componente.
Flujo de eventos	
Flujo básico Eliminar requisito de componente	
11	El sistema elimina el requisito de componente.
12	El sistema confirma la eliminación.
13	El sistema elimina dentro de la carpeta apps la carpeta del requisito de componente que se eliminó con toda la información que este contenía.
14	El sistema elimina dentro de la carpeta web la carpeta del requisito de componente que se eliminó con toda la información que este contenía.
15	Concluye el requisito.
Pos-condiciones	
3.	Se eliminó el requisito de componente seleccionado.
Flujos alternativos	
Flujo alternativo *.a El usuario cancela la acción	

4.	Concluye el requisito.	
Pos-condiciones		
5.	No se elimina el requisito de componente.	
Validaciones		
1	N/A	
Conceptos	**Componente**	Visibles en la interfaz: Descripción del requisito de componente
Requisitos especiales	N/A	
Asuntos pendientes	N/A.	

Tabla 9.Especificación de requisito Eliminar Requisito de Componente

2.5.10.2. Prototipo elemental de interfaz gráfica de usuario

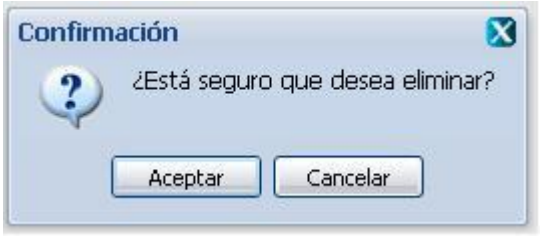

Ilustración 16. Eliminar Requisito.

2.5.11. Especificación de requisito Adicionar Funcionalidades del Componente.

2.5.11.1. Descripción textual del requisito

Precondiciones	Debe haberse seleccionado un componente.
Flujo de eventos	
Flujo básico Gestionar funcionalidades del componente.	
1	El usuario selecciona el botón adicionar funcionalidad.
2	El usuario introduce la denominación de la funcionalidad.
3	El usuario introduce la descripción de la funcionalidad.

4	El sistema crea dentro de la carpeta apps una carpeta con el mismo nombre de la funcionalidad del componente creado y la registra en el xml del componente con el mismo nombre.
5	El sistema crea dentro de la carpeta web una carpeta con el mismo nombre de la funcionalidad del componente creado y lo registra en el js de la funcionalidad con el mismo nombre.
6	El sistema valida la descripción (ver validación 1).
7	Si el dato es correcto el sistema lo adiciona.
8	Concluye el requisito.
Pos-condiciones	
5	N/A
Flujos alternativos	
Flujo alternativo 3.1	
1	N/A.
2	
Pos-condiciones	
1	N/A
Flujo alternativo 3.2	
1	N/A
2	
Pos-condiciones	
1	N/A
Flujo alternativo *.a El usuario cancela la acción	
3.	Concluye el requisito.
Pos-condiciones	
5	No se adiciona la funcionalidad al componente.
Validaciones	
4.	La descripción de la funcionalidad del componente debe contener solo caracteres alfabéticos, sin espacios.
Conceptos	**Componente** Visibles en la interfaz: Descripción de la funcionalidad del componente

Requisitos especiales	N/A
Asuntos pendientes	N/A

Tabla 10.Especificación de requisito Adicionar Funcionalidades del Componente

2.5.11.2. Prototipo elemental de interfaz gráfica de usuario

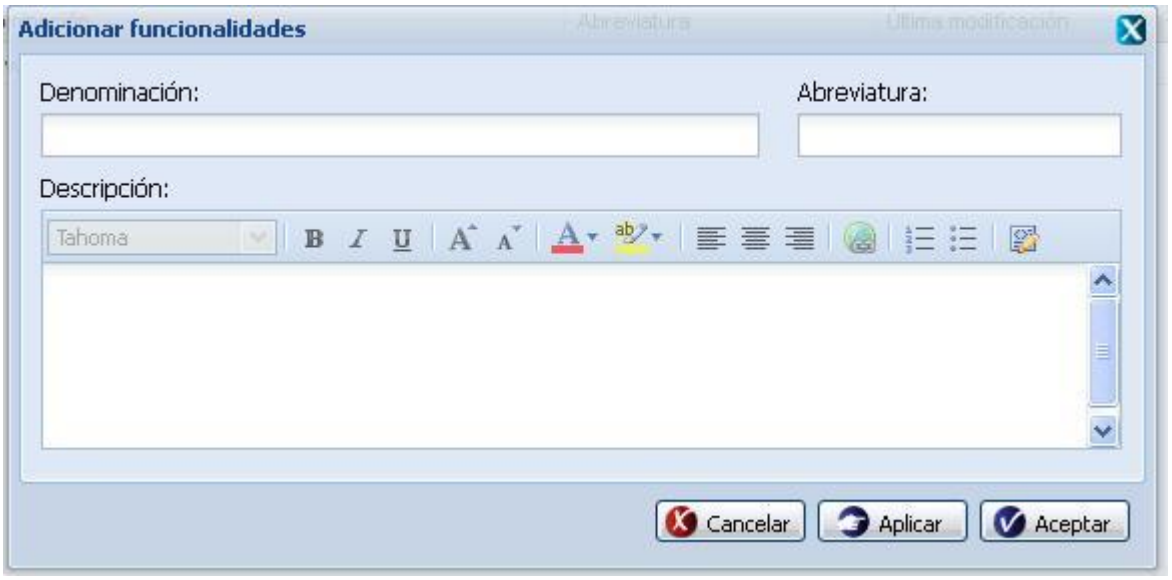

Ilustración 17. Adicionar funcionalidades.

2.5.12. Especificación de requisito Modificar Funcionalidades del Componente.

2.5.12.1. Descripción textual del requisito

Precondiciones	Debe haberse seleccionado una funcionalidad.
Flujo de eventos	
Flujo básico Gestionar funcionalidades del componente	
1	El usuario selecciona el botón modificar funcionalidades.
2	El usuario introduce la nueva denominación de la funcionalidad.
3	El usuario introduce la nueva descripción de la funcionalidad.
4	El sistema cambia dentro de la carpeta apps el nombre de la funcionalidad del componente que existía por el modificado por el usuario, así como el del xml de la funcionalidad.

5	El sistema cambia dentro de la carpeta web el nombre de la funcionalidad del componente que existía por el modificado por el usuario y modifica además el js de la funcionalidad con el nombre modificado.
6	El sistema valida la descripción (ver validación 1).
7	Si el dato es correcto el sistema lo modifica.
8	Concluye el requisito.
Pos-condiciones	
6	N/A
Flujos alternativos	
Flujo alternativo 3.1	
1	N/A.
2	
Pos-condiciones	
1	N/A
Flujo alternativo 3.2	
1	N/A
2	
Pos-condiciones	
1	N/A
Flujo alternativo *.a El usuario cancela la acción	
3.	Concluye el requisito.
Pos-condiciones	
6	No se modifica la funcionalidad del componente.
Validaciones	
4.	La descripción de la funcionalidad de componente debe contener solo caracteres alfabéticos, sin espacios.

Conceptos	**Componente**	Visibles en la interfaz: Descripción de la funcionalidad del componente.
Requisitos especiales	N/A	
Asuntos	N/A	

pendientes

Tabla 11.Especificación de requisito Modificar Funcionalidades del Componente

2.5.12.2. Prototipo elemental de interfaz gráfica de usuario

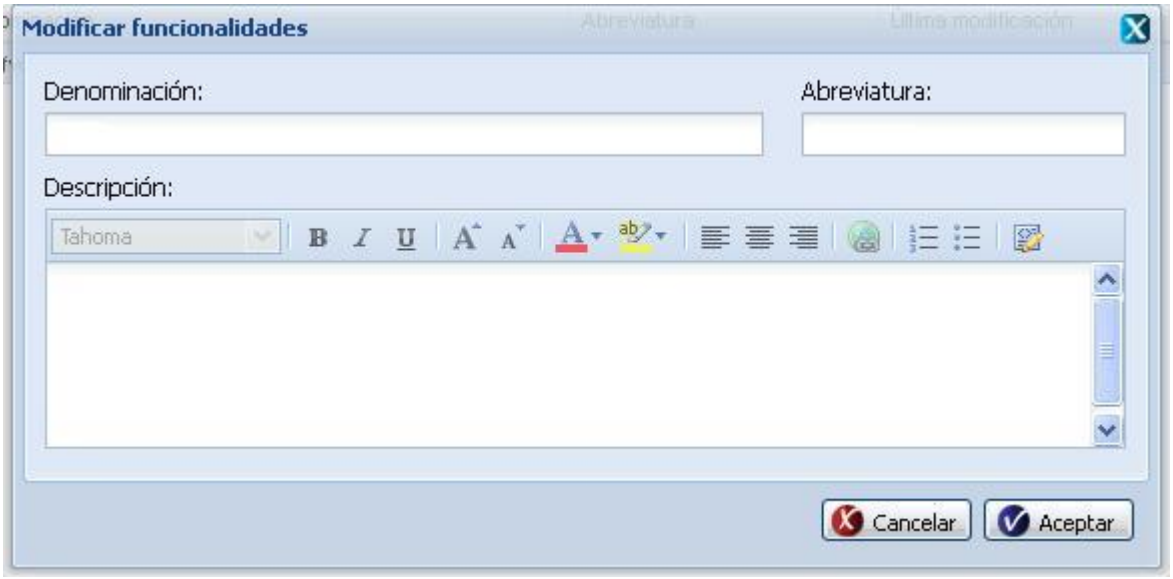

Ilustración 18. Modificar funcionalidades

2.5.13. Especificación de requisito Eliminar Funcionalidades.

2.1.13.1. Descripción textual del requisito

Precondiciones	Debe haberse seleccionado una funcionalidad.
Flujo de eventos	
Flujo básico Eliminar funcionalidad	
16	El sistema elimina la funcionalidad.
17	El sistema confirma la eliminación.
18	El sistema elimina dentro de la carpeta apps la carpeta de la funcionalidad que se eliminó con toda la información que este contenía.
19	El sistema elimina dentro de la carpeta web la carpeta de la funcionalidad que se eliminó con toda la información que este contenía.
20	Concluye el requisito.
Pos-condiciones	
2.	Se eliminó la funcionalidad seleccionada.
Flujos alternativos	

Flujo alternativo *.a El usuario cancela la acción		
3.	Concluye el requisito.	
Pos-condiciones		
4.	No se elimina la funcionalidad.	
Validaciones		
1	N/A	
Conceptos	**Componente**	Visibles en la interfaz: Descripción de la funcionalidad
Requisitos especiales	N/A	
Asuntos pendientes	N/A.	

Tabla 12.Especificación de requisito Eliminar Funcionalidad.

2.1.13.2. Prototipo elemental de interfaz gráfica de usuario

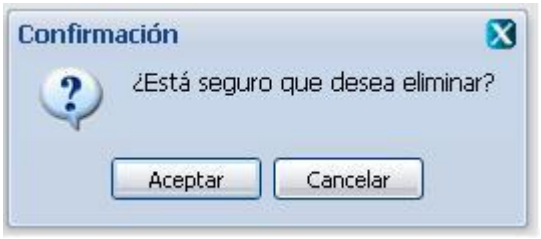

Ilustración 19. Eliminar funcionalidades.

2.6. Requisitos no Funcionales

Son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. Existen múltiples categorías para clasificar a los requisitos no funcionales, siendo las siguientes representativas de un conjunto de aspectos que se deben tener en cuenta, aunque no limitan a la definición de otros.

El presente trabajo forma parte de un proceso productivo iniciado por el CEIGE y los resultados que se obtengan formarán parte del marco de trabajo Sauxe. Por tanto los requisitos no funcionales con los que debe cumplir la aplicación desarrollada fueron establecidos por el centro al inicio del proceso de desarrollo, a

continuación se describen los más importantes. (22)

Usabilidad

RnF: 1 El sistema podrá ser usado por cualquier persona que posea conocimientos básicos en el manejo de la computadora.

Eficiencia

RnF: 2 Los tiempos de respuesta y velocidad de procesamiento de la información serán rápidos, no mayores de 5 segundos para las actualizaciones y 20 para las recuperaciones.

Soporte

RnF: 3 La aplicación contará antes de su puesta en marcha con un período de pruebas, se le dará mantenimiento, configuración y se brindará el servicio de instalación.

RnF: 4 Los desarrolladores deben utilizar para el desarrollo de la solución el estándar de codificación elaborado en el Departamento de Tecnología

Restricciones de diseño

RnF: 5 El lenguaje de programación a utilizar para desarrollar el sistema debe ser php para la lógica del negocio y ExtJS para la capa de presentación.

RnF: 6 Las herramientas a utilizar para desarrollar el sistema deben ser PostgreSql 8.3 como gestor de base de datos y Pgadmin III como cliente, Doctrine para la capa de acceso a datos y ZendExt Framework para la lógica del negocio

RnF: 7 El sistema debe ser multiplataforma, haciendo énfasis en Linux y Windows.

Estándares Aplicables

RnF: 8 El sistema debe utilizar el estándar SAML para la estandarización del proceso de autenticación basado en entornos xml.

RnF: 9 El sistema debe utilizar el estándar RBAC (*Modelo de Control de accesos Basado en Roles*) para la estandarización del proceso de autorización.

RnF: 10 El producto no debe contener palabras en otros idiomas.

RnF: 11 El producto debe respetar los términos empleados normalmente por los especialistas en el tema de la esfera que se automatiza.

Confiabilidad

RnF: 12 El sistema debe utilizar contraseña de acceso para llevar a cabo la

autenticación del usuario.

RnF: 13 El sistema debe garantizar protección contra acciones no autorizadas o que puedan afectar la integridad de los datos.

RnF: 14 La atención al sistema incluyendo, el mantenimiento de las bases de datos así como la salva de la información se realizarán de forma centralizada por el administrador.

RnF: 15 El sistema debe realizar verificaciones sobre las acciones irreversibles (eliminaciones).

Para que el sistema funcione es necesario garantizar los siguientes requisitos de software:

Para el cliente:

RnF: 2 Navegador Mozilla Firefox.

RnF: 3 Sistema operativo Windows 98 o superior o Linux.

Para el servidor:

RnF: 4 Sistema operativo Windows (2000 o superior) o Linux en cualquiera de sus distribuciones.

RnF: 5 Un servidor Apache 2.0 o superior con módulo PHP 5.0 disponible, este debe estar configurado con la extensión “pgsql” incluida.

RnF: 6 Un servidor de base de datos PostgreSQL 8.0 o superior.

Para que el sistema funcione es necesario garantizar los siguientes requisitos de hardware:

RnF: 7 Requerimientos mínimos: Procesador a 1GHz de velocidad de procesamiento y 1Gb de memoria RAM.

RnF: 8 Al menos 40Gb de espacio libre en disco duro.

RnF: 9 Tarjeta de red.

Para el cliente:

RnF: 10 Requerimientos mínimos: Procesador a 133Mhz con 128 Mb de memoria RAM.

RnF: 11 Tarjeta de red.

2.7. Diseño de Clases.

A continuación se presentan los diagramas de clases que según la notación UML

son una abstracción del dominio donde a partir de una estructura estática se representan los requisitos a través de las clases del sistema y sus interrelaciones, mostrando en términos generales qué debe hacer el sistema.

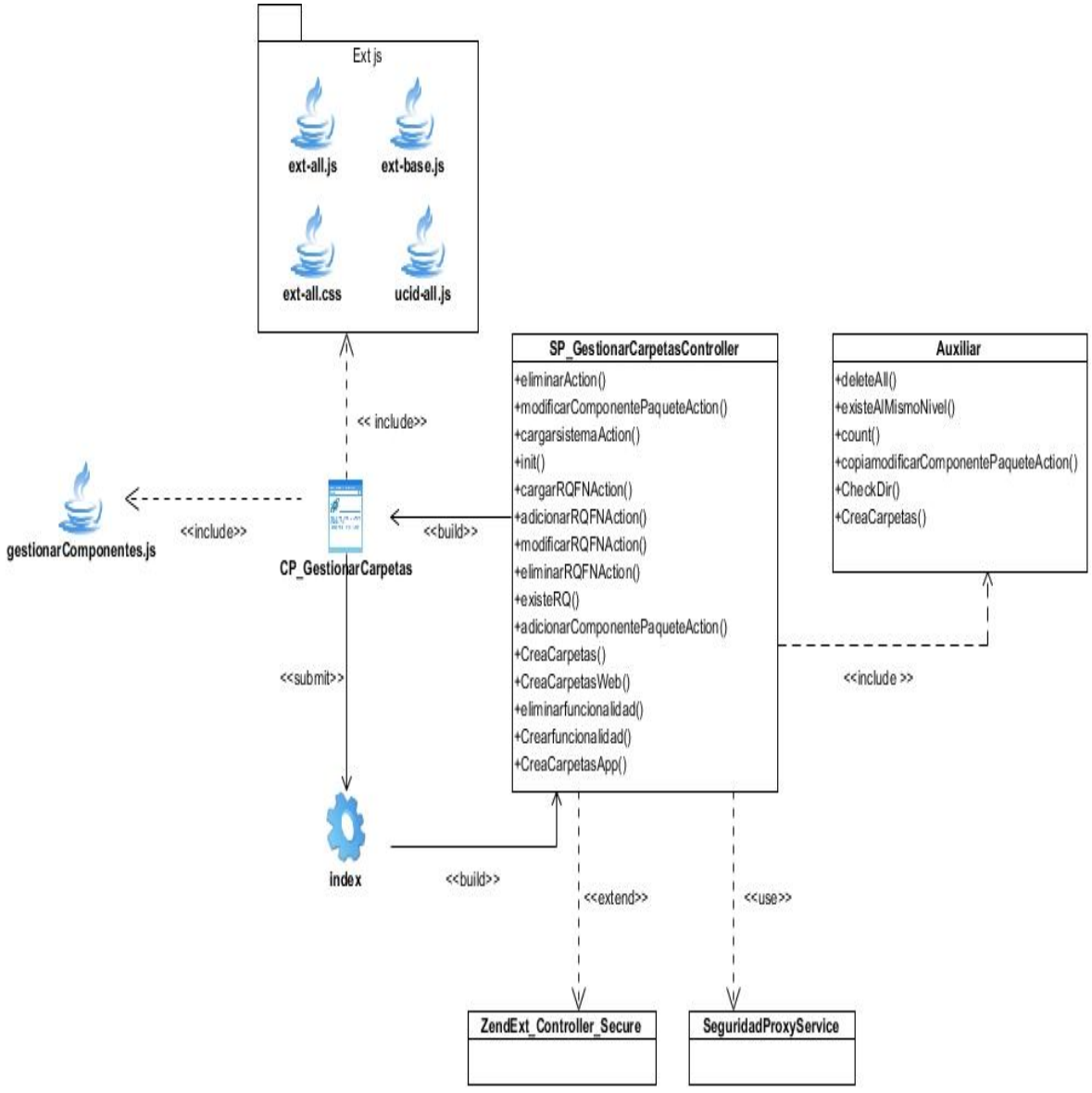

Ilustración 20. Diagrama de Clases del Diseño.

Tabla 4.Descripción del diseño de clases asociadas al componente Gestionar Componentes

Elementos representados	Descripción
Librería ExtJS	Contiene los componentes generados a través de la librería JavaScript ExtJS necesarios para

	visualizar el contenido de la presentación.
CP_GestionarCarpetas	Página encargada de visualizar, a través de los archivos de extensión js que debe incluir, la información relacionada con la gestión de las carpetas en los componentes.
gestionarCarpetas.js	Script encargado de generar de forma dinámica a través del DOM y utilizando la librería ExtJS, los componentes a los usuarios, así como las operaciones que se pueden realizar sobre estos. Debe enviar y recibir los datos de la controladora utilizando tecnología AJAX.
SP_GestionarCarpetasController	Contiene la clase controladora encargada de ejecutar las diferentes funcionalidades para visualizar los elementos relacionados con los componentes, según las peticiones del usuario.
ZendExt_Controller_Secure	Clase encargada de gestionar acciones personalizadas y está integrada a la seguridad.
SeguridadProxyServices	Es donde se crean los servicios del subsistema de seguridad.
Auxiliar	Permite manipular directorios y ficheros asociados a los paquetes de componentes y componentes, además contiene las funciones que permite gestionar los elementos de configuración (en xml) de los mismos.

2.8. Modelo de Datos.

Un modelo de datos es un lenguaje utilizado para la descripción de una base de datos. Por lo general, permite describir las estructuras de los datos de la base de datos, las restricciones de integridad y las operaciones de

manipulación de los datos. (23)

Como en la aplicación no se trabaja con un SGBD si no a través de XML, aquí aparece la estructura del componente.

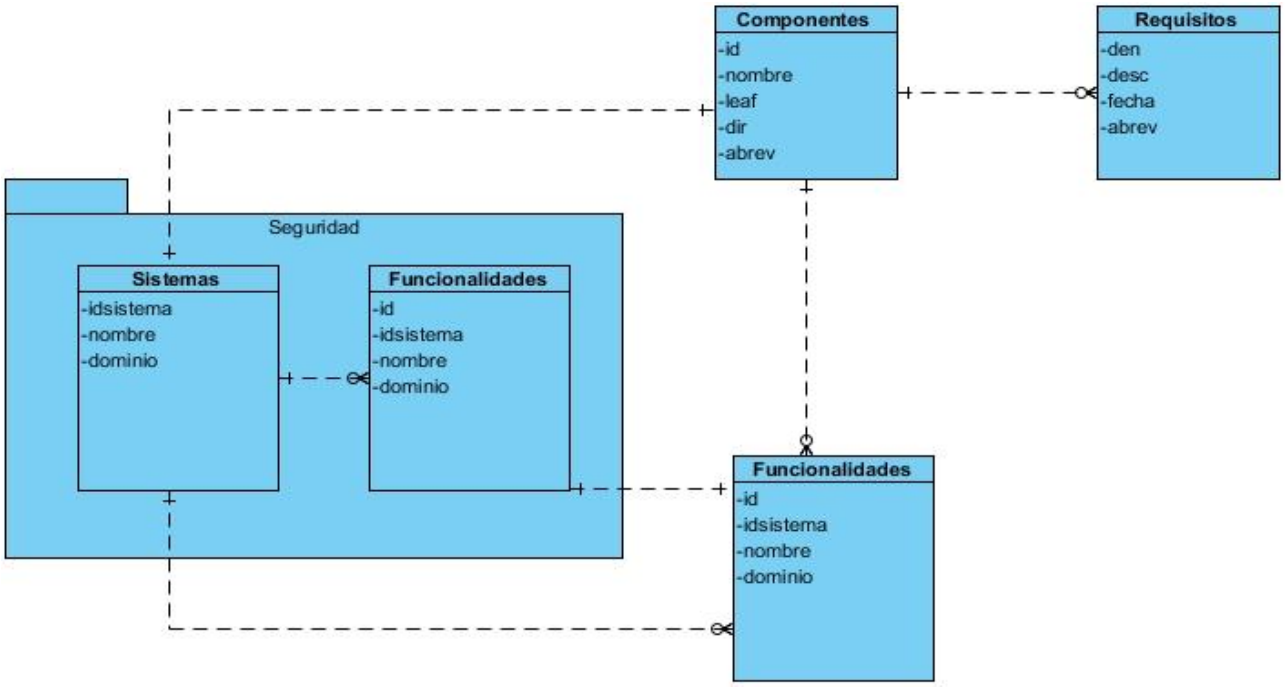

Ilustración 21. Modelo de Datos

2.9. Diagrama de despliegue.

El desarrollador, desde su puesto de trabajo, podrá acceder al sistema el cual estará desplegado en el mismo servidor web donde se encuentre ubicada la aplicación a la cual se le desee generar servicios web. Dicho servidor tendrá un XML en el cual se almacenará la información de interés para la solución. A continuación se muestra el diagrama de despliegue elaborado.

Ilustración 22. Diagrama de Despliegue

2.10. Patrones Utilizados.

Los patrones de diseño especifican la estructura y el comportamiento de una sociedad de clases. Estos son directrices y principios estructurados que describen un problema común y presentan soluciones simples. Están basados en la experiencia y se ha demostrado que funcionan. (24)

2.10. Modelo Vista Controlador (MVC)

• **Modelo**: Está compuesto por: datos, reglas de negocio y las funcionalidades correspondientes para la comunicación con el persistidor de datos Doctrine. Es el responsable de: Acceder a la capa de almacenamiento de datos, definir las reglas de negocio (la funcionalidad del sistema), llevar un registro de las vistas y controladores del sistema.

• **Vista:** Muestra la información del modelo al usuario. Este elemento es el responsable de recibir datos del modelo y mostrarlos al usuario. Tienen un registro de su controlador asociado.

• **Controlador:** Gestiona las entradas del usuario. Es el responsable de controlar todos los eventos generados en la vista y hace función de intermediar entre la vista y el modelo ante las peticiones generadas por el usuario en la interacción con la vista.

EL patrón MVC se aplica en el marco de trabajo Sauxe estableciendo una comunicación entre los elementos de la vista con el controlador y el controlador con el modelo. Para dar una respuesta hacia la vista se obtienen los datos del modelo, de ahí pasan por el controlador y el controlador envía la respuesta hacia la vista.

2.11. Patrones GRASP

GRASP: (*General Responsability Assignment Software Patterns*) se encargan de asignar responsabilidades a los objetos en sentido general, las principales responsabilidades son conocer los atributos y las relaciones con otros objetos y hacer las tareas que debe cumplir cada objeto. (25)

Entre los principales patrones GRASP utilizados se encuentran:

- **Experto**

Este patrón indica cual es la clase que debe asumir una determinada responsabilidad (clase experta), teniendo en cuenta la información que porta para cumplirla. Este se evidencia en la definición de las clases según las funcionalidades que realizan, como por ejemplo la clase Auxiliar que es la encargada de realizar las funcionalidades de insertar, actualizar y eliminar componentes. (26)

- **Alta Cohesión**

Con una alta cohesión dentro de cada clase definida se puede obtener una medida de la fuerza que une a las responsabilidades de una clase, resolviendo el problema de cómo lograr que la complejidad sea lo más manejable posible. Como consecuencia del uso de este patrón se logra un incremento de la claridad y comprensión, lo que simplifica el mantenimiento. Este patrón se aplica de manera general en el diseño del Subsistema PDO, al agrupar las clases en dependencia de los requisitos y los procesos que se necesitaban informatizar, de modo que cada clase implementa las operaciones que se encuentran en su misma área funcional, de esta forma la clase Auxiliar solamente se encarga de la creación y modificación de componentes, así como otras operaciones que se realizan sobre esta área y no de otro tipo de operación. (27)

- **Controlador**

El patrón Controlador define quién debe responder a determinados eventos. Una clase controladora debe ser la encargada de manejar los eventos dentro del componente, así la aplicación del patrón conlleva a separar la lógica de negocios de la capa de presentación, al aplicar estos principios, el controlador no realiza las actividades mencionadas sino que las delega en otras clases, como por ejemplo la clase GestionarcarpetasController en la que se mantiene un modelo de alta cohesión. (28)

- **Creador**

El patrón Creador resuelve el problema de quien debe crear una instancia de alguna clase, recomienda que una clase **B** debe crear una instancia de **A** si (28)

- ✓ B agrega los objetos de A.
- ✓ B contiene a los objetos de A.
- ✓ B registra las instancias de los objetos de A.

- ✓ B utiliza específicamente los objetos de A.
- ✓ B tiene los datos de inicialización que serán transmitidos hasta A cuando este objeto sea creado.

Este patrón se evidencia en la clase GestionarcarpetasController**,** la cual es la responsable de crear instancias de la clase Auxiliar, para así utilizar las funcionalidades de esta.

2.12. Conclusiones Parciales

Los argumentos expuestos en el desarrollo del presente capitulo demuestran lo siguiente:

- ✓ El diseño de la solución propuesta debe tener en cuenta elementos como la escalabilidad del sistema, confiabilidad, usabilidad, funcionalidad y accesibilidad, de modo que cubra las deficiencias presentadas en la ejecución actual del proceso.
- ✓ La elaboración del listado de requisitos funcionales, logró identificar y especificar tanto los aspectos del proceso de negocio como los recursos de aplicación y de soporte, que definirán la estructura del sistema que se pretende desarrollar como propuesta de solución.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA

3.1. Introducción

El trabajo en este capítulo parte de los resultados obtenidos en el análisis y diseño de la solución propuesta, se mostrará el modelo de implementación que pone en práctica el diseño de la solución realizado en el capítulo anterior con el objetivo de definir la estructura y organización de la solución y se especifica el conjunto de validaciones y pruebas que evalúan la calidad de la solución.

3.2. Modelo de Implementación

La Implementación comienza con el resultado obtenido del diseño detallado. Su objetivo principal es desarrollar la arquitectura y el sistema como un todo. De forma más específica, los propósitos de la Implementación son:

Planificar las integraciones de sistema necesarias en cada iteración. Siguiendo para ello un enfoque incremental.

Implementar clases, componentes y subsistemas encontrados durante el diseño.

Integrar componentes.

En los contiguos acápites se presenta la estructura física de la solución propuesta que coincide con la que define el marco de trabajo Sauxe y el diagrama de despliegue asociado a la misma.

3.2.1. Diagrama de componentes

El componente GDC (*Gestor Dinámico de Componentes*) es la solución que se propone para la generación dinámica de componente dentro de Sauxe. Este interactúa con el Subsistema Seguridad pues consume un servicio que posibilita visualizar en la aplicación que se desarrolle sobre Sauxe, los componentes definidos en GDC. A continuación se muestra el diagrama de componentes lógicos elaborado.

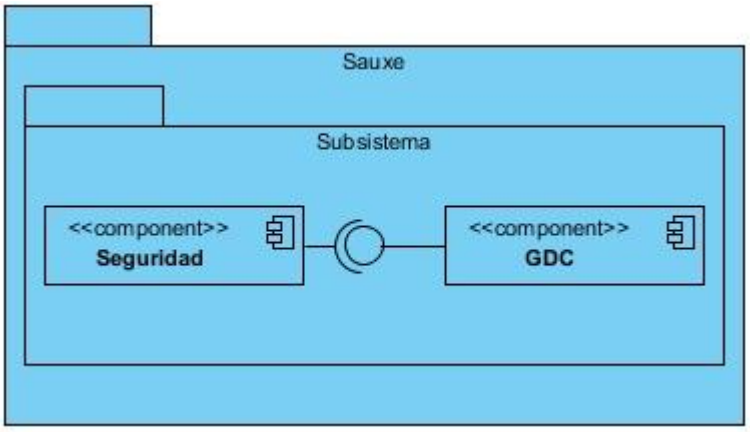

Ilustración 23. Diagrama de Componentes lógicos.

3.2.2. Estándares de código.

Un estándar de código se basa en la estructura y apariencia física del código de un programa con el fin de facilitar la lectura, comprensión, mantenimiento del código, reutilización a lo largo del proceso de desarrollo de un software y no en la lógica del programa. Su objetivo fundamental es definir la nomenclatura de las variables, objetos, métodos y funciones, así como también tiene que ver con el orden y legibilidad del código escrito.

Partiendo de esto, se definen 3 partes principales dentro de un estándar de programación:

- ✓ Convención de nomenclatura: Es la forma de nombrar las variables, funciones y clases.
- ✓ Convenciones de legibilidad de código: Es la forma de organizar el código.
- ✓ Convenciones de documentación: Es la manera de establecer comentarios, la ayuda, entre otros.

- **Nomenclatura de las funciones:**

El identificativo a emplear para las funciones o métodos se escribe con la primera palabra en minúscula utilizando la notación **CamelCasing** que define que la letra inicial del identificador comienza con minúscula y usando nombres que deduzcan

su propósito. Ejemplo: cargarMetodos. Los denominadores de las acciones de las clases controladoras tienen la peculiaridad de ir seguidos por la palabra “Action”. Ejemplo: cargarCMPAction.

- **Nomenclatura de los atributos**

Las variables se nombran convenientemente de acuerdo con el estándar CamelCasing. Ejemplo: idsubsistema y nombreclase.

3.3. Validación de la Solución Propuesta.

A continuación se exponen los procedimientos empleados para la validación de la solución propuesta. Primeramente se presentan las métricas del diseño en los componentes lo que permite juzgar la calidad del diseño de los mismos. Luego se describen las pruebas de aplicación realizadas. Estas fueron ejecutadas con el objetivo de detectar y corregir el máximo de errores posibles.

3.3.1. Validación de los objetivos

Para la validación de los objetivos se tomaron en cuenta las variables que se plantean en el problema a resolver, llegando a la siguiente conclusión:

Variables:

- Disminución del tiempo de desarrollo de los componentes.
- Introducción de errores por parte de los programadores.

Se seleccionaron 10 desarrolladores para verificar si la solución resuelve los problemas con que cuenta el marco de trabajo Sauxe. Estos 10 desarrolladores se dividieron en 2 grupos, un grupo de 5 desarrolladores creaban los componentes de forma manual y el otro grupo lo hacía mediante la solución. Esto arrojó los siguientes datos:

Grupo de desarrolladores que crearon los componentes de forma manual:

- 1 desarrollador creó el componente en 1 hora y 30 minutos y tuvo 5 errores.
- 1 desarrollador creó el componente en 44 minutos y tuvo 6 errores.
- 1 desarrollador creó el componente en 26 minutos y tuvo 4 errores.
- 1 desarrollador creó el componente en 17 minutos y tuvo 5 errores.
- 1 desarrollador creó el componente en 10 minutos y tuvo 3 errores.

Grupo de desarrolladores que crearon los componentes haciendo uso de la solución:

- 2 desarrolladores crearon el componente en 14 segundos y no tuvieron errores.
- 2 desarrolladores crearon el componente en 11 segundos y no tuvieron errores.
- 1 desarrollador creó el componente en 10 segundos y no tuvo errores.

Comparando las variables presentes en el problema con las mismas variables haciendo uso de la solución se demostró que la solución disminuye en gran medida el tiempo de creación de los componentes, así como de las funcionalidades que presenta la solución, además se eliminan los errores que cometían los desarrolladores en la creación de los componentes, logrando una estandarización en la gestión de los componentes. Señalar además que el tiempo de creación de los componentes utilizando la solución está en dependencia de la agilidad con la que cuente el desarrollador ó la persona que utilice la solución.

3.3.2. Validación del diseño propuesto.

Las métricas para aplicaciones informáticas, no son perfectas; muchos expertos argumentan que se necesita más experimentación hasta que se puedan emplear bien las métricas de diseño. Sin embargo el diseño sin medición, en opinión de la Licenciada en Ingeniería en Sistemas Computacionales Heidi González Doria, es una alternativa inaceptable. A continuación se presentarán las métricas Tamaño operacional de clase (TOC) y Relaciones entre clases (RC), necesarias para evaluar la calidad del diseño propuesto a nivel de componentes. Ambas métricas no fueron aplicadas durante el diseño procedimental, sino que fueron retrasadas hasta tener disponible el código fuente, pues la complejidad del negocio exigía la incorporación de diversas subrutinas necesarias para la implementación de cada procedimiento descrito en el diseño de clases. Las pruebas que se le hicieron a la aplicación arrojaron un resultado bueno, liberándose la aplicación y demostrando que esta cumple con los estándares de calidad definidos en el CEIGE.

Las métricas TOC y RC incluyen medidas de los siguientes atributos de calidad:

Responsabilidad: Responsabilidad que posee una clase en un marco conceptual correspondiente al modelado de la solución propuesta.

Complejidad del mantenimiento: Nivel de esfuerzo necesario para sustentar, mejorar o corregir el diseño de software propuesto. Puede influir significativamente en los costes y la planificación del proyecto.

Complejidad de implementación: Grado de dificultad que tiene implementar un diseño de clases determinado.

Reutilización: Significa cuán reutilizada es una clase o estructura de clase dentro de un diseño de software.

Acoplamiento: Dependencia o interconexión de una clase o estructura de clase respecto a otras.

Cantidad de pruebas: Número o grado de esfuerzo necesario para realizar las pruebas de calidad al producto (componente) diseñado.

3.3.3. Métrica TOC: Tamaño operacional de clase.

Se refiere al número de procedimientos existentes en una clase. Determina una relación directa entre los atributos Responsabilidad y Complejidad de implementación, sin embargo establece una relación inversa entre estos últimos y el atributo Reutilización.

Como resultado de la evaluación de la métrica TOC se obtuvo lo siguiente:

Atributo de calidad	Modo en que lo afecta
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.

Tabla 6. Atributos de calidad evaluados por la métrica TOC.

Para los cuales están definidos los siguientes criterios y categorías de evaluación:

Atributo	Categoría	Criterio

Responsabilidad	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Complejidad implementación	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Reutilización	Baja	>2*Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	<=Promedio

Tabla 7. Criterios de evaluación para la métrica TOC.

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:

Atributo de calidad	Modo en que lo afecta
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.

Tabla 8. Atributos de calidad evaluados por la métrica RC

Para los cuales están definidos los siguientes criterios y categorías de evaluación:

Atributo	Categoría	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2
Complejidad de	Baja	<=Promedio
	Media	Entre Promedio y

mantenimiento		2*Promedio
	Alta	>2*Promedio
Reutilización	Baja	>2*Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	<=Promedio
Cantidad de pruebas	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio

Tabla 9. Criterios de evaluación para la métrica RC.

Resultados obtenidos de la aplicación de la métrica TOC

Una vez obtenidos los resultados de la evaluación del instrumento de medición de la métrica TOC, se puede concluir que el diseño propuesto tiene una calidad aceptable ya que las clases empleadas en la solución poseen 6 operaciones o más lo que conlleva a evaluaciones positivas de los atributos de calidad involucrados (responsabilidad, complejidad de implementación y reutilización).

A continuación se muestran los resultados obtenidos.

Clases	**Cantidad de Métodos**	**Responsabilidad**	**Complejidad**	**Reutilización**
Auxiliar	9	Baja	Baja	Alta
GestionarCarpetasController	20	Media	Media	Media
SeguridadProxyServices	7	Baja	Baja	Alta

Tabla 13.Instrumento de evaluación de la métrica TOC

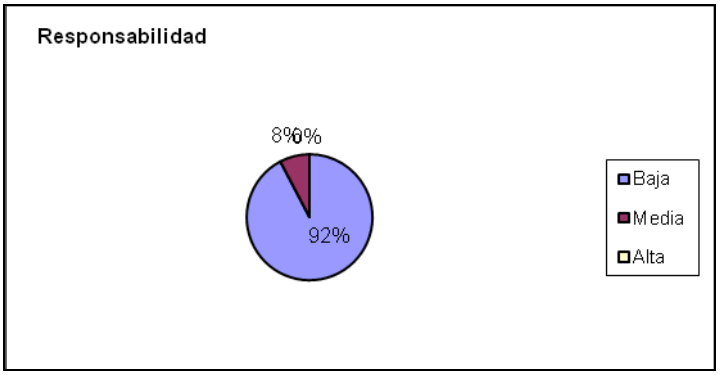

Ilustración 24. Resultados de la evaluación de la métrica TOC para el atributo responsabilidad.

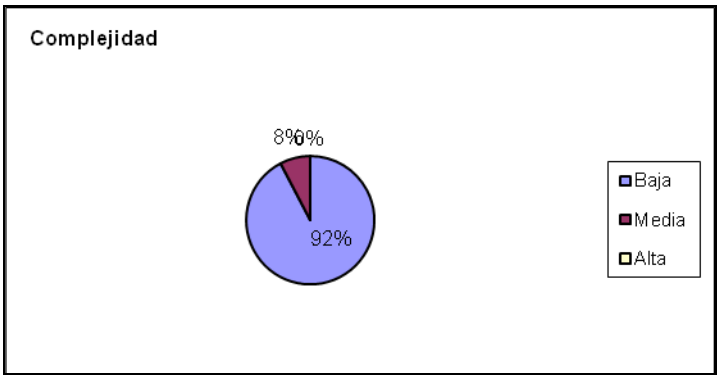

Ilustración 25. Resultados de la evaluación de la métrica TOC para el atributo complejidad.

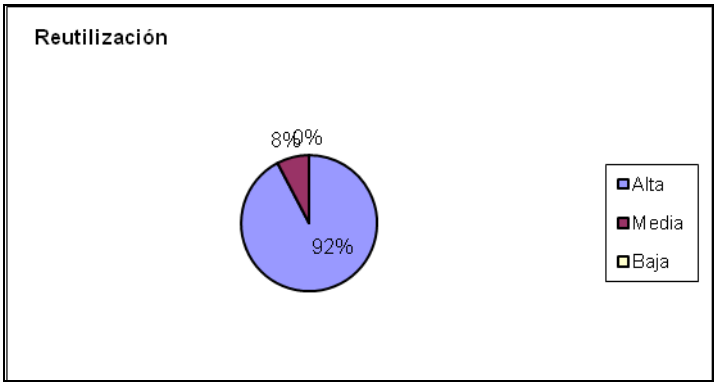

Ilustración 26. Resultados de la evaluación de la métrica TOC para el atributo reutilización.

Resultados obtenidos de la aplicación de la métrica RC

Una vez obtenidos los resultados de la evaluación del instrumento de medición de

la métrica RC, se puede concluir que el diseño propuesto tiene una calidad aceptable ya que las clases empleadas poseen menos de 3 dependencias de otras clases lo que conlleva a evaluaciones positivas de los atributos de calidad involucrados (acoplamiento, complejidad de mantenimiento, cantidad de pruebas y reutilización).

A continuación se muestran los resultados obtenidos.

Clases	Cantidad de Relaciones de Uso	Acoplamiento	Complejidad Mant.	Reutilización	Cantidad de Pruebas
Librería ExtJS	2	Medio	Baja	Alta	Baja
GestionarCarpetasController	3	Alto	Media	Media	Media
SeguridadProxyServices	2	Medio	Baja	Alta	Baja

Tabla 14.Instrumento de evaluación de la métrica RC

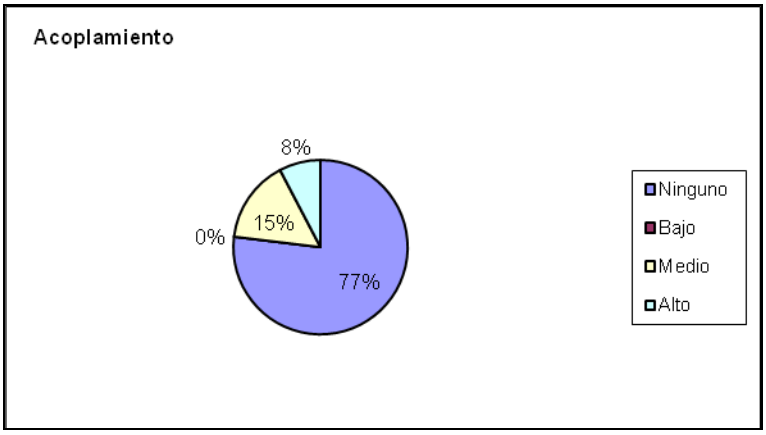

Ilustración 27. Resultados de la evaluación de la métrica RC para el atributo acoplamiento

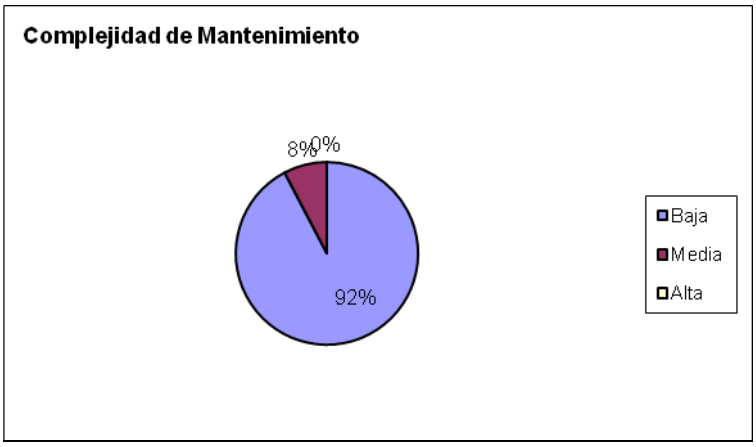

Ilustración 28. Resultados de la evaluación de la métrica RC para el atributo complejidad de mantenimiento

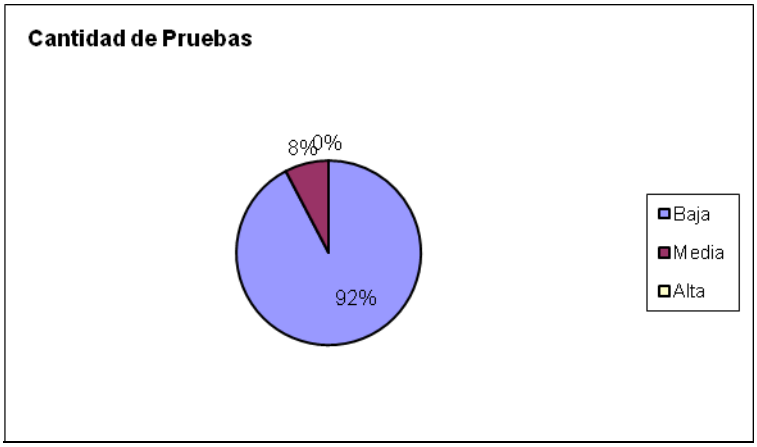

Ilustración 29. Resultados de la evaluación de la métrica RC para el atributo cantidad de pruebas

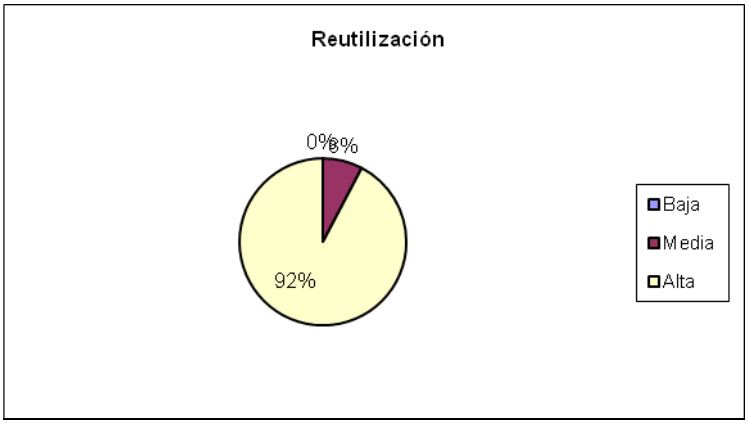

Ilustración 30. Resultados de la evaluación de la métrica RC para el atributo reutilización

Matriz de inferencia de indicadores de calidad

La matriz inferencia de indicadores de calidad, también llamada matriz de

cubrimiento, es una representación estructurada de los atributos de calidad y métricas utilizadas para evaluar la calidad del diseño propuesto. Dicha matriz permite conocer si los resultados obtenidos de las relaciones atributo/métrica es positivo o no, llevando estos resultados a una escalabilidad numérica donde, si los resultados son positivos se le asigna el valor de 1, si son negativos toma valor 0 y si no existe relación es considerada como nula y es representada con un guión simple (-). Luego se puede obtener un resultado general para cada atributo promediando todas sus relaciones no nulas.

A continuación se muestran los resultados obtenidos.

Atributos\Métricas	TOC	RC	Promedio
Responsabilidad	1	(-)	1
Complejidad de Implementación	1	(-)	1
Reutilización	1	1	1
Acoplamiento	(-)	1	1
Complejidad de Mantenimiento	(-)	1	1
Cantidad de pruebas	(-)	1	1

Tabla 15. Resultados de la evaluación de la relación atributo/métrica

Categoría	Rango de valores
Malo	<=0.4
Regular	>0.4 y <0.7
Bueno	>=0.7

Tabla 16. Rango de valores para la evaluación de la relación atributo/métrica

3.4. Pruebas de Software

Las pruebas constituyen una etapa imprescindible durante el proceso de desarrollo del software. Su objetivo principal es asegurar que el software cumpla con las especificaciones requeridas y eliminar los posibles defectos que este pudiera tener. De forma más específica los propósitos de las pruebas son:

- Realizar la validación del software desarrollado, entendiendo como validación el proceso que determina si el software satisface los requisitos.

- Realizar la verificación del software desarrollado, entendiendo como verificación el proceso que determina si los productos de una fase satisfacen las condiciones de la misma.

Es importante considerar que las pruebas de software no garantizan que un sistema esté libre de errores, sino que se detecte la mayor cantidad de defectos posibles para su debida corrección. (29)

3.4.1. Niveles de Prueba:

A la hora de evaluar dinámicamente un sistema se debe comenzar por los componentes más simples y pequeños e ir avanzando progresivamente hasta probar todo el software en su conjunto. Las pruebas se aplican en distintos niveles de trabajo, dentro de estos se distinguen:

1. Pruebas de Unidad: Prueba individual a las unidades separadas de un sistema de software.

2. Pruebas de Integración: Los componentes individuales son combinados con otros componentes para asegurar que la comunicación, enlaces y los datos compartidos ocurran apropiadamente.

3. Pruebas del Sistema: Son usualmente conducidas para asegurar que todos los módulos trabajan como sistema sin error. Es similar a las pruebas de integración pero con un alcance mucho más amplio.

4. Pruebas de Aceptación: Son realizadas principalmente por los usuarios con el apoyo del equipo del proyecto. El propósito es confirmar que el sistema está terminado, que desarrolla puntualmente las necesidades de la organización y que es aceptado por los usuarios finales.

3.4.2. Métodos de Prueba

Un método de prueba es un enfoque sistemático, independiente del nivel en que se enmarque la prueba, que ayuda a encontrar buenos conjuntos de casos de prueba para detectar diferentes tipos de errores. (29)

1. Pruebas de Caja Blanca: Se comprueban los componentes internos. Comprueba los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado.

2. Pruebas de Caja Negra: Se comprueban las funcionalidades sin tener en cuenta la estructura interna. Se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. O sea, a través de los casos de prueba se demuestra que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene.

Este tipo de pruebas permite encontrar:

- ✓ Funciones incorrectas o ausentes.
- ✓ Errores de interfaz.
- ✓ Errores en estructuras de datos o en accesos a las bases de datos externas.
- ✓ Errores de rendimiento.
- ✓ Errores de inicialización y terminación.

El tipo de pruebas realizadas a la solución fueron pruebas de caja negra, para de esta forma comprobar las funcionalidades, además se hicieron a través de la interfaz de la solución, nunca en la parte interna, es decir en el código de la solución.

3.4.3. Casos de prueba

Los casos de prueba asociados a las pruebas de caja negra especifican una forma de probar el sistema, incluyendo las entradas con las que se ha de probar, las condiciones bajo las que ha de probarse, así como los resultados esperados. Se elaboraron 11 casos de pruebas a la solución. A continuación se muestra el DCP relacionado con el requisito Adicionar Paquete de Componente. El resto se encuentra en la carpeta de entregables.

3.4.3.1. Diseño de casos de prueba

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1: Adicionar Paquete de Componente.	Se adiciona un componente ó un paquete de	EP 1.1: Adicionar un componente ó un paquete de componente	➢ Se presiona el botón **Adicionar**. ➢ Se insertan todos

	componente, con el atributo: Denominación y abreviatura.	introduciendo datos válidos presionando el botón **Aceptar**.	los datos. ➢ Se presiona el botón **Aceptar**. ➢ Se presiona el botón **Aceptar** del mensaje de información.
		EP 1.2: Adicionar un componente ó un paquete de componente dejando campos requeridos en blanco presionando el botón **Aceptar**.	➢ Se presiona el botón **Adicionar**. ➢ Se introducen los datos dejando campos requeridos en blanco. ➢ Se presiona el botón **Aceptar**. ➢ Se presiona el botón **Aceptar** del mensaje de información.
		EP 1.3: Adicionar un componente ó un paquete de componente introduciendo datos válidos presionando el botón **Aplicar**.	➢ Se presiona el botón **Adicionar**. ➢ Se insertan todos los datos. ➢ Se presiona el botón **Aplicar** para mantener activa la interfaz. – Se presiona el botón **Aceptar** del mensaje de error.
		EP 1.4: Adicionar un componente ó un paquete de componente dejando campos requeridos en blanco presionando el botón **Aplicar**.	➢ Se presiona el botón **Adicionar**. ➢ Se introducen los datos dejando campos requeridos en blanco. ➢ Se presiona el

		botón **Aplicar** para mantener activa la interfaz. ➢ Se presiona el botón **Aceptar** del mensaje de error.
	EP 1.5: Cancelar	– Se presiona el botón **Adicionar**. – Se introducen o no los datos en el formulario. – Se presiona el botón **Cancelar**.

Tabla 17. DCP Adicionar Paquete de Componente.

Conclusiones Parciales

El presente capítulo fue crucial para definir algunos elementos que intervienen en el desarrollo de la solución propuesta. En ese sentido se presentó la estructura física de componente Solución para la Generación Dinámica de Componentes, se definieron los estándares de código para su implementación y los componentes necesarios para su despliegue. Se aplicaron además las métricas: Tamaño Operacional de la Clase y Relaciones entre Clases para evaluar la calidad el diseño propuesto. Los resultados arrojados por dichas métricas permitieron valorar principalmente la complejidad en la implementación de la solución que se presenta con la actual investigación.

También se efectuaron pruebas de caja negra al componente Solución para la generación dinámica de componentes para verificar su correcto funcionamiento, aplicándose pruebas de partición de equivalencia. Tras 2 iteraciones de realización de pruebas funcionales se detectaron un total de 6 no conformidades. Estas fueron resultas, permitiendo cumplir con los requisitos capturados en la primera etapa de desarrollo.

Las pruebas de aceptación realizadas hasta el momento por los usuarios con el apoyo del equipo del proyecto han permitido verificar y validar los requerimientos

identificados. Dichas pruebas arrojan que la solución propuesta, en opinión de los desarrolladores, proporciona un avance en la gestión de componentes, ya que se hacía de forma manual y con esta solución dinámica se ahorra tiempo y se evita la introducción de errores por parte de los desarrolladores, logrando de esta forma la estandarización en la gestión de los componentes.

Conclusiones

El desarrollo de la presente investigación y, en consecuencia, la obtención de los resultados generados por la misma, han permitido arribar a las siguientes conclusiones:

- Con el análisis del curso actual de la gestión de componentes en el marco de trabajo Sauxe se pautaron los aspectos significativos que frenan la agilidad de su desarrollo. Esto permitió desarrollar una solución informática que agilice el proceso de gestión de los componentes en el marco de trabajo Sauxe.
- La Modelación y Construcción de la solución propuesta viabiliza la creación, modificación y eliminación de componentes de forma dinámica, posibilitando la gestión dinámica de las mismas. Esto resuelve el problema de estandarización y tiempo de desarrollo de los componentes.
- Proporciona un soporte confiable para el almacenamiento centralizado de la información que se genera durante la ejecución del proceso de Gestión de componentes de forma dinámica.
- A partir de la implementación de los requisitos capturados para la primera etapa de desarrollo se obtuvo una solución integrada, probada y estable.
- Se destaca con la solución propuesta, la incorporación de principios por los que se mide la factibilidad del diseño de una aplicación informática. La utilización de patrones y métricas es un ejemplo fehaciente de ello.
- Por consiguiente, el desarrollo de la Solución para la generación dinámica de componentes en el marco de trabajo Sauxe constituye una mejora en la realización del proceso de Gestión de componentes de forma dinámica en el marco de trabajo Sauxe.

Recomendaciones

Se recomienda considerando cumplidos todos los objetivos trazados en el presente trabajo de diploma lo siguiente:

- Extender las funcionalidades relacionadas con la Gestión de Componentes, en aras de ampliar las posibilidades de trabajo con la solución desarrollada.

Referencias bibliográficas

1- Gómez Baryolo, Oiner, Morejón Borbón, Yoandry y García Tejo, Darien. ARQUITECTURA TECNOLÓGICA PARA EL DESARROLLO DE SOFTWARE. Habana : UCI.

2- Joomla. Syswoody. Syswoody. [En línea] [Citado el: 12 de Febrero de 2011.] http://www.syswoody.com/utilidades/analisis-de-seguridad-en-aplicaciones-web

3- Microsoft. msdn. msdn. [En línea] [Citado el: 22 de Marzo de 2011.] http://www.msdn.microsoft.com/es-es/library/aa291820%28VS.71%29.aspx

4- Baryolo, Gómez, Baryolo, Tenrero, Cabrera, Marianela y Silega, Martínez, Nemuris. 2008. Plantilla Registro de la Propiedad intelectual (Sauxe). 2008.

5- ARQUITECTURA TECNOLÓGICA PARA EL DESARROLLO DE SOFTWARE Ing. Oiner Gómez Baryolo

6- Formalización y estandarización de la documentación técnica de la arquitectura tecnológica del Marco de Trabajo Sauxe versión 2.0. La Habana: s.n., 2010 Cañete, Yanisleydi. Libro de Ayuda del Marco de Trabajo Sauxe. La Habana: s.n., 2010.

7- Clemens Szyperski, “Component Software Component Software Beyond Object – Oriented Programming”. P 20 -22. 1998.

8- Philippe Krutchen. “Rational Software”

9- http://www.ecured.cu/index.php/Lenguajes_de_Descripcion_Arquitectonica.

10- http://www.ecured.cu/index.php/UniCon.

11- http://www.ecured.cu/index.php/Rapide.

12- http://www.ecured.cu/index.php/Darwin.

13- http://www.ecured.cu/index.php/Zend_Framework

14- http://www.ecured.cu/index.php/Symfony

15- 2005.Visual-paradigm. [En línea] 2005. [Citado el: 28 de 09 de 2010.] http://www.visual-paradigm.com.

16- Ferré, Grau Xavier. 2011. “Tutorial UML, Desarrollo Orientado a Objetos con UML.”*Clikear.com*, 2011. [Citado el: 14 de 11 de 2010.] http://www.clikear.com/manuales/uml/index.aspx.

17- 2010.ZEND FRAMEWORK.[En línea] 2010. [Citado el: 27 de 01 de 2011.] http://framework.zend.com/manual/en/.
18- 2009.Documentación del Servidor HTTP Apache 2.0. [En línea] 2009. [Citado el: 28 de 10 de 2010.] http://httpd.apache.org/docs/2.0/es/.
19- Sommerville, Ian. 2005. Ingeniería de Software. Septima edición. Madrid : Pearson Educación S.A., 2005. 84-7829-074-5.
20- Sarduy Pérez, Mileidy Magalys y Hernández Cisneros, Sergio. Propuesta de modelo de desarrollo de software tecnológico del Centro de Soluciones de Gestión. Ciudad de la Habana: s.n., 2009.
21- Sommerville, Ian. 2005**.** Ingeniería de Software. Septima edición. Madrid : Pearson Educación S.A., 2005. 84-7829-074-5.
22- Plantilla de especificación de requisitos de software de CEIGE. 2012.
23- 2008. Definicion.de. [En línea] 2008. http://definicion.de/modelo-de-datos/.
24- Salazar, Ing. Tte. Leonel. 2008. Curso de Capacitación por Roles: Generalidades de los Patrones de Diseño. Habana,Cuba : s.n., 2008.
25- El mundo informático. [En línea] [Citado el: 8 de enero de 2010.] http://jorgesaavedra.wordpress.com/2006/08/17/patrones-grasp-craig-larman/.
26- Prieto, Félix. 2009. Patrones de diseño.
27- Jará, O.H. “Evolución y Orientaciones de Patrones” [última visita 01-10-2007].
28- Pressman, Roger. 1998. Ingeniería de Software. Un enfoque práctico. 1998.
29- Ramírez, Jaime. Unidad de Programación. Métodos de Prueba del Software.

Bibliografía Consultada

1. Definición.org. [En línea] [Citado el: 6 de Octubre de 2010.] http://www.definicion.org.
2. Alvarez, Miguel Angel. Desarrollo Web. [En línea] [Citado el: 6 de 10 de 2010.] http://www.desarrolloweb.com/articulos/que-es-html.html.
3. Corzo, Giancarlo. Desarrollo en Web. [En línea] 22 de 10 de 2008. [Citado el: 5 de 09 de 2010.] http://blogs.antartec.com/desarrolloweb/2008/10/extjs-lo-bueno-lo-malo-y-lo-feo/.
4. Superuser. 2003. Tienda Linux. [En línea] 31 de 05 de 2003. [Citado el: 5 de 10 de 2010.] http://soporte.tiendalinux.com/portal/Portfolio/postgresql_ventajas_html.
5. Lage Dávila, Carlos. 2007. Discurso en la clausura del Seminario Nacional con los directores de empresas que aplican el perfeccionamiento empresarial. 29 de 08 de 2007.
6. Gómez Baryolo, Oiner, Morejón Borbón, Yoandry y García Tejo, Darien. ARQUITECTURA TECNOLÓGICA PARA EL DESARROLLO DE SOFTWARE. Habana : UCI.
7. Pressman, Roger. 1998. Ingeniería de Software. Un enfoque práctico. 1998.
8. González, Dianelys Brito. 2011.ARQUITECTURA DE SOFTWARE. Habana : CEIGE, 2011. CEIGE-N-00-i001.
9. Chaviano Gómez, Enrique y Carrascoso Puebla, Yoan Arlet. 2009.Propuesta de arquitectura orientada a servicios para el módulo de inventario del ERP cubano. Habana : UCI, 2009.
10. UCID. 2009. Proceso de desarrollo y gestión de proyectos de software. Habana : UCID, 2009.
11. Fernández, Osmar Leyet. 2010. Documento Línea Base de proyecto-CEDRUX-1.0. 2010.
12. González Doria, Heidi. 2010. Métricas en el desarrollo del Software. Capitulo 4. México : Universidad de las Américas Puebla.. ISBN/ISSN.
13. Visconti, Marcelo y Astudillo, Hernán. 2010. Fundamentos de Ingeniería de Software. Valparaiso.España. : Departamento de Informática. Universidad Técnica Federico Santa María.
14. 2010. Los datos de objetos de PHP. [En línea] 2010. [Citado el: 28 de 1 de 2010.] http://php.net/manual/es/intro.pdo.php

15. Centro de Informatización y Gestión de Entidades. 2010.Cedrux.Desarrollo Tecnológico. Habana. : UCI.
16. IEEE-SA. 2010. “IEEE SA - 1471-2000 - IEEE Recommended Practice for Architectural Description for Software-Intensive Systems”. [En línea]. http://standards.ieee.org/findstds/standard/1471-2000.html.
17. Microsoft. 2011. “Capítulo 3: Patrones de arquitectura y estilos.” [En línea]. *MSDN Magazine*. http://msdn.microsoft.com/en-us/library/ee658117.aspx.
18. UCID. 2008.Vista de datos de la Arquitectura de ERP. Habana : UCID.