Universidad de las Ciencias Informáticas

Facultad 1

Biblioteca de componentes para la implementación de la interfaz web del Gestor de Documentos Administrativos eXcriba

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

Autor: Alberto Del Toro Fuentes

Tutores: Ing. Misael Fonseca Mata

Ing. Michel David Suarez

La Habana, Junio 2012

Declaración de autoría

Declaro que soy el único autor de este trabajo y autorizo al Centro de Informatización Universitaria de la Universidad de las Ciencias Informáticas, para que hagan el uso que estimen pertinente con este trabajo.

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

_____

Firma del autor

Alberto Del Toro Fuentes

_____

Firma del tutor

Ing. Misael Fonseca Mata

_____

Firma del tutor

Ing. Michel David Suarez

Agradecimientos

A mi familia por el apoyo, el amor y la ayuda que me han brindado toda mi vida y principalmente durante estos 5 años que estuve lejos de mis padres. Gracias por hacerme sentir siempre como en casa.

A mis tutores en especial a Michel por dedicarme noches sin dormir para que la tesis pudiera estar lista y con buena calidad.

A Lizandra por atenderme cada vez que tenía alguna duda o queja de la tesis. Gracias por defenderme, por ayudarme y por preocuparte por mi trabajo.

A mis amigos de Santiago que aunque están lejos siempre me apoyan y se preocupan por mi. Yaser, Yanet, Yisel, Yunior, Alejandro y Carlos, siempre los llevo presente.

A mis amigos de La Habana por apoyarme y preocuparse por mi en todo momento, por acogerme en sus casas y brindarme su amistad. Humberto, Enrique, Darien y Leonel gracias por ser mis amigos.

A mis amigos de la universidad por ser las personas más lindas y maravillosas que he conocido en toda mi vida.

A Mirley por tantos momentos buenos y malos que hemos pasado juntos. Para mi eres como una hermana, considero que eres la persona que mejor me conoce. Gracias por darme apoyo, amor, cariño, por confiar en mi, por soportarme cuando tengo hambre, por hacerme reir, en fin por estar a mi lado estos 5 años. Para mi es un honor ser tu amigo.

A Liset por sus consejos, por transmitirme energía positiva incluso en los momentos más difíciles, por su sonrisa, por apoyarme y entenderme siempre, por todos los momentos lindos que pasamos juntos en playas, conciertos y hasta tiendas, por atenderme en su casa como si fuera de la familia, por las tantas horas que pasábamos soñando si fuéramos esto o tuviéramos aquello. Gracias a tus padres que se han portado muy bien conmigo, gracias por alegrarme los dias. Estoy orgulloso de ti

A frank por ser mi primer amigo en la escuela, por compartir mis gustos musicales, por las tantas peñas de rock a las que íbamos sin conocer a nadie, por preocuparse por mi y ayudarme a convalidar Física I y II,

por enseñarme que la amistad no tiene límites y por todos los momentos que hemos pasado en estos años. Gracias por ser la persona más inteligente que he conocido en mi vida.

A Susana por dejarme entrar en su vida, por ser la niña más generosa de la UCI, por apoyarme y darme cariño, por darme comida cada vez que tengo hambre, por su humildad, por ser más fuerte de lo que aparenta. Gracias por ser tan especial.

A Aliu por los buenos momentos que vivimos en primero y segundo, el helado en las tardes, las historias del pre y las conversaciones y risas hasta las 3 de la madrugada. Gracias por todo.

A Ismael y Roberto gracias por ofrecerme su amistad y sobre todo por hacer de aliu y liset las mujeres más felices del mundo.

A Reinier Elejalde por ser un excelente profesor, tutor y amigo. Si te pagaran por cada línea de código que has escrito para nosotros fueras millonario y pudieras comprar toda una fábrica de galleticas de vainilla. Sin tu ayuda muchos no hubiésemos llegado tan lejos. Gracias por todo.

A todos los que compartieron conmigo desde el primer día de la universidad, a los que ya no están y a los que quedan. A mis compañeros de apartamento, a los del 10202 y en especial a los miembros del 10108 por ser el mejor grupo en el que he dado clases.

Dedicatoria

A mis padres por el amor, el cariño, el apoyo y la educación que me han dado durante todos estos años.

Gracias por sentirse orgullosos de mi.

A mi hermano por ser un ejemplo a seguir sin ni siquiera saberlo.

A mi abuela por hacerme sentir que soy su único nieto, simplemente por hacerme sentir especial.

Esta tesis va dedicada especialmente a ustedes por ser las personas más importantes de mi vida.

Resumen

En el Gestor de Documentos Administrativos (GDA) eXcriba no existe una vía homogénea para crear los componentes que conforman la interfaz web dificultando la labor de los desarrolladores. Para darle solución a los problemas derivados de lo antes planteado se propone desarrollar una biblioteca de componentes que sirva de infraestructura base para crear nuevos componentes y así incrementar la reutilización de código a la vez que se reduce el tiempo de desarrollo de la interfaz web del sistema.

Para su cumplimiento se realizó un estudio del estado del arte sobre el desarrollo basado en componentes y se seleccionaron las herramientas y tecnologías necesarias para el desarrollo del módulo. Además, se definieron los componentes candidatos para formar parte de la biblioteca y las funcionalidades de la misma. Se diseñó e implementó la biblioteca de componentes evaluándose finalmente su correcto funcionamiento mediante pruebas de caja blanca y caja negra las cuales arrojaron resultados satisfactorios.

Palabras clave: componente, desarrollo basado en componentes.

Índice General

Introducción

La evolución de las Tecnologías de la Información y las Comunicaciones (TIC) ha proporcionado formas más fáciles de interactuar con las computadoras. Dicha interacción se realiza a través de interfaces de usuario que constituyen el punto de contacto entre la persona y el ordenador.

En la actualidad existe un aumento indiscriminado de información y en ocasiones resulta difícil asimilar lo que se está analizando. La principal causa es que los datos se presentan con escaso diseño de interfaces, provocando al usuario una sensación de desorientación. Para una persona es prácticamente imposible extraer conclusiones, tendencias y patrones del contenido consultado. Con el fin de darle solución a este problema se han desarrollado diversas aplicaciones centradas en la visualización de la información, representándola mediante imágenes y gráficos, haciéndola más comprensible, manejable y útil. Sin embargo los usuarios de dichas aplicaciones exigen que estas sean creadas en menores tiempos y con mayor calidad.

Lo antes planteado, es una de las causas por la cual desde hace un tiempo ha estado cambiando la forma en que se programan las aplicaciones. Otro factor importante es que el desarrollo de software continúa siendo una actividad costosa en tiempo y dinero y los errores en el producto final son frecuentes. El nivel de reutilización sigue siendo muy escaso a pesar de la multitud de bibliotecas de código[1] existentes y disponibles.

La tendencia en los últimos años ha sido el desarrollo de software basado en componentes[2]. El mismo permite crear aplicaciones en menor tiempo usando componentes ya diseñados para integrarse con facilidad. Esta nueva forma de programación ha tomado gran auge debido a que resuelve grandes problemas como la

[1] Bibliotecas de código: conjunto de subprogramas utilizados para desarrollar software.

[2] Componente: pieza de software casi independiente que ejecuta funciones y que puede ser modificado en tiempo de diseño.

reutilización de código, la escalabilidad[3] y el mantenimiento a la vez que agiliza el proceso de implementación.

El Gestor de Documentos Administrativos eXcriba (GDA eXcriba) es un software desarrollado en la Universidad de las Ciencias Informáticas que tiene como objetivo principal automatizar los procesos documentales que se ejecutan dentro de cualquier entidad[4], desde la elaboración de un documento en su fase de inicio hasta su conservación o expurgo. Sus principales funcionalidades son: crear, clasificar, describir mediante la norma internacional ISAD(G), versionar, definir tipologías, gestionar flujos documentales, almacenar documentos en diferentes formatos electrónicos, gestionar los trámites de los documentos que se generan o reciben y salvaguardar el patrimonio documental entre otras [1].

A pesar de las características y funcionalidades antes mencionadas, la programación de componentes en el eXcriba se encuentra de forma desagregada. En la confección de cada uno de ellos se usaron los marcos de trabajo[5] jQuery y CodeIgniter. Esta situación provoca que no exista una vía homogénea para la implementación del software trayendo consigo problemas de duplicidad de código, mayores tiempos de producción y mayor complejidad a la hora de crear los componentes que son usados en el diseño de la interfaz web del eXcriba.

A partir de la situación antes mencionada se plantea como **problema a resolver**: ¿Cómo simplificar el proceso de implementación de componentes en la interfaz web del GDA eXcriba?

El **objetivo general** del presente trabajo es: Desarrollar una biblioteca de componentes que permita simplificar la implementación de la interfaz web del GDA eXcriba.

Para darle solución al mismo se definen los siguientes **objetivos específicos**:

[3]Escalabilidad: capacidad de un sistema de cambiar su tamaño o configuración para adaptarse a circunstancias cambiantes sin perder su calidad.

[4]Entidad: colectividad considerada como unidad. Cualquier corporación, compañía o institución tomada como persona jurídica.

[5]Marco de trabajo o *framework*: conjunto de bibliotecas que poseen una documentación y una metodología de uso, que permite diseñar e implementar aplicaciones de forma más uniforme, rápida y con mayor calidad.

- Fundamentar los conceptos teóricos relacionados con el desarrollo basado en componentes.
- Definir los componentes que serán empleados para el diseño de la aplicación.
- Especificar cómo funciona el modelo, la vista y los controladores para cada componente.
- Elaborar una guía para el desarrollo de aplicaciones con el uso de la biblioteca de componentes.

Como **objeto de estudio** se define: el desarrollo de software basado en componentes delimitando el campo de acción en: el desarrollo de software basado en componentes en las aplicaciones web.

Métodos teóricos

Analítico - Sintético: para dividir el objeto de estudio en conceptos que serán examinados por separado, estudiándose rigurosamente cada uno de ellos de manera independiente, confrontando el criterio de varios autores y las correspondencias entre ellos. Igualmente se analizarán las diferentes herramientas y técnicas necesarias para el desarrollo de la solución.

Histórico-Lógico: para la adquisición de conocimientos sobre las actividades de la ingeniería y el desarrollo de software basado en componentes.

Modelación: para modelar los diagramas que se obtengan durante el diseño del módulo, obteniendo una panorámica del flujograma de actividades.

Justificación de la investigación

Actualmente en el GDA eXcriba no existe una vía homogénea para implementar los componentes. Por esta razón se concebió el desarrollo de una biblioteca para la interfaz web que posibilitará el uso y creación de componentes en el sistema. La misma servirá de infraestructura en la cual los componentes podrán relacionarse unos con otros y dar como resultado en ocasiones otros nuevos todos con la misma estructura definida para ello. Cada uno contendrá un manual de uso y desarrollo que facilitará el trabajo con los mismos. El uso de la biblioteca contribuirá a reducir al mínimo las líneas de código utilizadas en el proceso de desarrollo. Todas estas funcionalidades permitirán agilizar en el eXcriba las tareas relacionadas con el diseño e implementación de la interfaz web haciendo uso de la biblioteca implementada.

Estructura del documento

El presente documento consta de una introducción, cuatro capítulos, conclusiones, recomendaciones y anexos. A continuación se explica el contenido de los capítulos.

Capítulo I Fundamentación teórica: se realiza un análisis de los fundamentos teórico-conceptuales de la programación de *software* basado en componentes. Además, se seleccionan las tecnologías y herramientas necesarias para la solución del problema.

Capítulo II Características del sistema: se describen las características del módulo a desarrollar, para ello se detalla la propuesta de solución, se identifican los requisitos funcionales y no funcionales, así como los casos de uso que darán solución a los mismos.

Capítulo III Diseño del sistema: se realizan los diagramas de clases del diseño y se define la arquitectura del módulo.

Capítulo IV Implementación y prueba del sistema: se abordan los aspectos relacionados con la implementación y prueba del software, detallando los diagramas de despliegue y de componentes de la solución y los casos de prueba realizados.

Capítulo 1

Fundamentación teórica

El desarrollo de la informática y las telecomunicaciones ha proporcionado un cambio en la forma en la que se programa actualmente un software. Las causas principales son la demanda de tiempos de desarrollo más cortos así como el costo en tiempo y dinero que implica crear una aplicación. Por tal motivo se hace necesario introducir nuevas estrategias como el uso de componentes de software. La programación basada en componentes se convirtió en un paradigma por su eficiencia y su alto grado de reutilización.

En este capítulo se estudiarán conceptos como componente de software así como aspectos relacionados a la programación basada en componentes. Además se seleccionan las herramientas, tecnologías y lenguajes para la realización del módulo.

1.1. Componente de software

Para Szyperski[1]: *"Un componente de software es una unidad de composición con interfaces especificadas contractualmente y solamente dependencias explícitas de contexto. Un componente de software puede ser desplegado independientemente y está sujeto a composición por terceros"* [citado por [2]].

Según Philippe Krutchen[2]: *"Un componente de software es una parte no trivial, casi independiente y reemplazable de un sistema que cumple una función dentro del contexto de una arquitectura bien definida. Un componente cumple con un conjunto de interfaces y provee la realización física de ellas"* [citado por [2]].

Del análisis de las anteriores definiciones se deduce que un componente de software es: "una pieza de software casi independiente que ejecuta funciones y que puede ser modificado en tiempo de diseño sin tener

[1] Arquitecto de software en la división de sistemas conectados y afiliado con Microsoft Research.

[2] Ingeniero y profesor de ingeniería de software en la Universidad de British Columbia en Vancouver, Canadá. Director de desarrollo de proceso (RUP) en Rational Software.

que conocer absolutamente nada sobre su implementación. Los servicios de los componentes son conocidos mediante sus interfaces y requisitos".

1.1.1. Características de los componentes

En su artículo "Desarrollo de software basado en componentes" A Montilva, Arapé y Colmenares [2] definen las características que debe satisfacer un componente software. Según ellos cada componente debe ser:

- Identificable – Un componente debe tener una identificación clara y consistente que facilite su catalogación y búsqueda en repositorios de componentes.
- Auto-contenido – Un componente no debe requerir de la utilización de otros para ejecutar la función para la cual fue diseñado.
- Genérico – Sus servicios deben servir para varias aplicaciones.
- Mantenido – Un componente debe estar inmerso en un proceso de mejoramiento continuo que le garantice al integrador nuevas versiones que incluyan correctivos, optimizaciones y nuevas características.
- Accesible sólo a través de su interfaz – El componente debe exponer al público únicamente el conjunto de operaciones que lo caracteriza (interfaz) y ocultar sus detalles de implementación. Esta característica permite que un componente sea reemplazado por otro que implemente la misma interfaz.
- Sus servicios son invariantes – Las operaciones que ofrece un componente, a través de su interfaz, no deben variar. La implementación de estos servicios puede ser modificada, pero no debe afectar la interfaz.
- Documentado – Un componente debe tener una documentación adecuada que facilite su búsqueda en repositorios de componentes, adaptación a nuevos entornos e integración con otros componentes.

- Certificado – El componente puede ser certificado por una agencia de software independiente o mediante la aplicación de modelos de auto-certificación que le permiten al comprador del componente determinar la calidad del software adquirido.

Un componente se puede ver como un producto comercial de calidad, realizado por un fabricante especializado. Su distribución es a través de un único paquete instalable el cual contiene todo lo necesario para su funcionamiento. Puede estar implementado y ser utilizado en cualquier lenguaje de programación. La relación entre varios componentes proporciona la funcionalidad del sistema. Las interfaces definen las operaciones y los servicios del mismo. Cuando sólo se conoce la interfaz y la funcionalidad no se le puede realizar ningún tipo de modificación al componente.

Cuando se va a utilizar un componente lo primero que se hace es instanciarlo. La declaración de instancias se realiza de forma similar a la declaración de variables en un lenguaje de programación. El próximo paso es configurar el componente y finalmente realizar la conexión de las interfaces de entrada y salida con otros componentes. Tras la realización de todos los pasos indicados anteriormente se podrán utilizar todos los métodos ofertados por las interfaces de entrada del componente.

1.1.2. Clasificación de componentes

Los componentes de software se pueden clasificar de varias maneras de acuerdo al tamaño o a las acciones que realizan. La clasificación que se enuncia a continuación está orientada a la reutilización y fue tomada de [3]:

- Componentes GUI[3] – Proporcionan las interfaces externas de las aplicaciones y conocen el funcionamiento de la interacción con el usuario. Por si mismos, no representan un sistema funcional, ya que necesitan componentes más complejos para ejecutar las peticiones del usuario.

- Componentes de Servicio – Proveen servicios comunes requeridos por diferentes aplicaciones, como acceso a bases de datos, servicios de mensajería, transacciones e integración con servicios del sistema.

[3]GUI: *Graphic User Interface* o Interfaces Gráficas de Usuario.

Una de las principales características de estos componentes, es que todos necesitan usar sistemas o infraestructura adicional para poder ejecutar su funcionalidad.

- Componentes de Dominio – Cuando son reusables pertenecen sólo a un dominio de aplicación específico y no pueden ser reutilizados fuera de él. Por hacer parte de un dominio de aplicación en particular, deben contar con una infraestructura de aplicación propia.

1.1.3. Técnicas de certificación de componentes

La especialista Rebeca Díaz [4] considera que existen 4 técnicas de certificación de componentes, las cuales se describen a continuación:

Certificación por niveles – Se establecen un conjunto de niveles que satisfacen una serie de requisitos de calidad, de forma que cada componente reutilizable será evaluado según estos criterios y le será asignado un nivel de certificación.

Modelos de certificación estadísticos – Esta técnica se centra en la evaluación de la fiabilidad de un componente. Es necesario definir un modelo del sistema (componente), estados de comportamiento y las probabilidades de transición entre estados. De esta manera se podrá estimar qué trazas son más probables y consecuentemente, cuáles precisarán ser más fiables.

Modelos de certificación local – Se verifica la corrección de cada elemento sin tener en cuenta su relación con otros, de esta manera se logra facilitar las tareas de verificación para la composición de elementos.

Certificación orientada al dominio de aplicación – En este tipo de certificación se realizan los siguientes pasos:

- Determinar los requisitos de calidad.
- Definir la instancia de certificación (propiedades de un componente y técnicas para demostrar que las satisface).
- Desarrollo de una biblioteca que satisfaga los requisitos de calidad.

- Emplear este tipo de certificación para obtener aplicaciones usando componentes reutilizables.

1.1.4. Modelo de componentes

Un modelo de componentes es utilizado como guía de diseño por los desarrolladores. Define los componentes que se van a usar, cómo deben ser y cómo se relacionan entre ellos y con el sistema, el modo en que estos se especifican y el modelo de referencia de ejecución [5].

1.1.4.1. Características de un modelo de componentes

De acuerdo con las valoraciones de Daniel Garrido [6], un modelo de componente se caracteriza por ser:

- Estandarizado – El modelo debería ser aprobado por un amplio grupo de profesionales e investigadores.
- Independiente del lenguaje – Un modelo de componentes no está desarrollado para un lenguaje de programación específico ni para ninguna característica especial de ningún lenguaje.
- Define interfaces – Un modelo de componentes debería proporcionar un mecanismo para definir las interfaces de un componente.
- Facilita el empaquetado – Cómo el componente es suministrado al usuario.
- Permite introspección – Un modelo de componentes permite la introspección de los componentes en varias fases de desarrollo, desde la fase de diseño hasta la de ejecución.
- Soporte para dinamismo – Un modelo de componentes debe permitir al sistema reorganizarse en tiempo de ejecución (los componentes y sus conexiones).

1.1.4.2. Modelos de componentes

Al referirse a los modelos de componentes, el especialista Daniel Garrido [6] considera que los modelos actuales para la programación basada en componentes son:

La plataforma .NET

Es una nueva biblioteca de Microsoft para el desarrollo de aplicaciones. Para .NET un componente es la forma binaria de una clase y un simple ensamblado puede contener muchos componentes de este tipo. Los mismos van a estar contenidos en unidades de configuración y despliegue llamadas ensamblados.

El modelo de componentes de Java

JavaBeans presenta un modelo de componentes independiente de la plataforma, pero dependiente del lenguaje de programación Java, recibiendo los componentes el nombre de *beans*. Además de los JavaBeans, que básicamente son componentes gráficos, Sun ha desarrollado los denominados EnterpriseBeans. Este tipo de componentes se ejecutan dentro de un contenedor Enterprise JavaBeans (EJB) que es un entorno de ejecución dentro de un servidor de aplicaciones.

El modelo de componentes CORBA[4] CCM (*Corba Component Model*)

CCM es el modelo de componentes propuesto por OMG[5]. En vez de definir el componente con interfaces, los componentes se declararán con la palabra *component*. En tiempo de ejecución, un componente es representado por una referencia a él mismo. CCM proporciona un gran número de interfaces para soportar la estructura y funcionalidad de los componentes. La implementación para muchas de estas interfaces puede ser generada automáticamente.

1.1.5. Plataformas de componentes

Una plataforma de componentes es el entorno de desarrollo que permite solucionar algunos de los problemas asociados a la construcción de aplicaciones basadas en componentes.

ActiveX/OLE

Un objeto ActiveX se define como el que se adhiere al Modelo de Objetos de Componentes (*Component Object Model* COM) definido por Microsoft. Las aplicaciones ActiveX están conceptualmente divididas en

[4]CORBA (*Common Object Request Broker Architecture*): estándar abierto que maneja al detalle la interoperabilidad entre componentes y es utilizado para el desarrollo de aplicaciones que se ejecutan en sistemas distribuidos.

[5]OMG (*Object Management Group*): organización que se encarga de la definición de una arquitectura de objetos y principal impulsora de CORBA.

servidores, objetos que hacen que sus métodos y propiedades estén disponibles para los demás y aplicaciones clientes que usan objetos expuestos en el servidor, métodos y propiedades. Vinculación e incrustación de objetos *(Object Linking and Embedding* (OLE)) proporciona un estándar consistente que permite a los objetos, aplicaciones y componentes ActiveX, comunicarse entre sí con la finalidad de usar el código de los demás. Los objetos no necesitan conocer por anticipado en qué objetos se van a comunicar, ni su código necesita estar escrito en el mismo lenguaje [7].

Enterprise Java Beans

Modelo de componentes basado en la arquitectura cliente-servidor. Esta plataforma ofrece una solución multiplataforma, de fácil reutilización, integración con otros componentes además de la máquina virtual de java, seguridad transaccional, entre otras. Estas características la han hecho reconocida en el mundo de los programadores y del desarrollo basado en componentes [8].

1.2. Reutilización en el proceso de desarrollo de software

La ingeniería de software ha evolucionado desde los lenguajes imperativos hasta lenguajes basados en objetos. Esto se debe entre otros factores al aumento de la complejidad de las aplicaciones, el difícil mantenimiento de las mismas y la necesidad de reutilización de código. La programación orientada a objetos parecía la solución para todos estos problemas y permitía abarcar proyectos más complejos y de una forma más segura. A pesar del éxito de este paradigma de programación, surgieron nuevos desafíos como las posibilidades de reutilización en el proceso de desarrollo de software.

"La reutilización de software es un proceso de la ingeniería de software que conlleva al uso recurrente de activos de software en la especificación, análisis, diseño, implementación y pruebas de una aplicación o sistema de software" [2].

La reutilización de componentes de software es similar a la manera en que se ensamblan componentes en la ingeniería de los sistemas físicos. Tiene gran importancia ya que incide directamente en el costo, tiempo y esfuerzo requerido para desarrollar un producto de software. Otros beneficios importantes son el incremento de la calidad del software producido y el aumento de la productividad de los grupos de desarrollo.

1.3. Ingeniería de software basada en componentes

Los sistemas de hoy en día son cada vez más complejos, deben ser construidos en tiempo mínimo y deben cumplir con los estándares más altos de calidad. Para hacer frente a esta problemática surge lo que hoy se conoce como ingeniería de software basada en componentes la cual esta centrada en el diseño y construcción de sistemas de software reutilizables. Define los métodos y herramientas para dar soporte a la construcción de componentes y sistemas que surgen a partir de estos. Además, acompaña a dos actividades de ingeniería paralelas: la ingeniería de dominio y el desarrollo basado en componentes.

1.3.1. Ingeniería de dominio

Su objetivo es identificar, construir, catalogar y diseminar un conjunto de componentes de software que tienen aplicación en el software actual y futuro dentro de un dominio[6] de aplicación en particular [9]. La ingeniería de dominio cuenta con 3 fases las cuales son análisis, diseño e implementación del dominio. En cada una de estas fases se realizan actividades que propician la obtención de la infraestructura que sirve de base para la reutilización. De acuerdo a lo planteado por Omar Gómez [10] dichas actividades son:

- Obtención de requerimientos – Se obtienen los requerimientos y se transforman en diferentes modelos tales como el proceso de negocio, conceptos del negocio, la visión del sistema y los casos de uso.
- Identificación de componentes – Se toman como entradas el modelo de conceptos de negocio y el modelo de casos de uso del flujo de trabajo de requerimientos – Se asume una capa de aplicación que incluye una separación de los componentes del sistema y los componentes de negocio el objetivo de esta etapa es identificar un conjunto inicial de interfaces de negocio para los componentes de negocio y un conjunto inicial de interfaces de sistema para los componentes del sistema y colocarlos juntos dentro de una arquitectura inicial de componentes.
- Interacción de componentes – Se examinan cómo cada una de las operaciones del sistema serán alcanzadas usando la arquitectura de componentes. Se usan modelos de interacción para descubrir

[6]Dominio: es un área o ámbito de aplicación de productos software.

operaciones sobre las interfaces de negocio. Entre más interacciones sean consideradas, patrones y operaciones de uso en común emergen y así las interfaces pueden ser reutilizadas.

- Especificación de componentes – Se definen para una interfaz dada los estados potenciales de los objetos de un componente en un modelo de información de interfaz y entonces se especifican las precondiciones y poscondiciones para las operaciones, finalmente se capturan las reglas del negocio como restricciones.

- Implementación de componentes – Una vez que se cuentan con las especificaciones se procede con la etapa de implementación, en esta etapa se implementan los componentes en alguna plataforma tecnológica.

La ingeniería de dominio proporciona la biblioteca de componentes reutilizables. Los componentes pueden ser obtenidos comercialmente o implementados por el equipo de desarrollo. Para lograr una perfecta integración de estos componentes con un nuevo sistema y una nueva arquitectura se realizan las siguientes actividades especificadas por Roger Pressman [9]:

- Cualificación de componentes – Se verifica que el componente cumple con la calidad requerida, se adapta a la arquitectura del sistema y cumple la función para la que fue creado.

- Adaptación de componentes – Cada componente debe adaptarse a las normas de diseño que define la arquitectura de no ser así se reemplaza por otro que se pueda adaptar. Existe una técnica de adaptación llamada “encubrimiento de componentes” la cual varia de acuerdo al tipo de componente.

- Composición de componentes – Se establece una infraestructura (generalmente una biblioteca de componentes especializados) donde los componentes estén unidos a un sistema operacional. La infraestructura proporciona un modelo para la coordinación de componentes y servicios específicos que hacen posible dicha tarea.

1.3.2. Desarrollo basado en componentes

Este paradigma de programación es una solución eficiente que se adapta a la velocidad de evolución del mercado. Como se muestra en la figura 1.1 (versión extendida del documento) el desarrollo basado en componentes se basa en la construcción de una aplicación a partir de componentes de software comerciales o gratuitos ya existentes, reduciendo al mínimo el desarrollo de código nuevo mediante la reutilización.

La programación orientada a componentes surge con la idea de diseñar y desarrollar aplicaciones distribuidas[7] basadas en componentes de software reutilizables. Mantiene la filosofía de la programación orientada a objetos. Los objetos son instancias de clases y estas a su vez se relacionan entre sí mediante la herencia. Con la llegada de esta nueva programación se crea un mercado global de componentes de software, en el cual los desarrolladores pueden construir aplicaciones de forma más rápida.

1.3.2.1. Conceptos básicos del desarrollo basado en componentes

Lidia Fuentes, Troya J M y Antonio Vallecillo [11] consideran que existe un conjunto de conceptos básicos que intervienen en la programación orientada a componentes y que permiten diferenciarla del resto de los paradigmas de programación. Entre ellos se encuentran los siguientes:

Composición tardía – Es la composición que se realiza en un tiempo posterior al de la compilación del componente, como puede ser durante su enlazado, carga o ejecución y por alguien ajeno a su desarrollo.

Entornos – Conjunto de recursos y componentes que rodean a un objeto o componente dado y que definen las acciones que sobre él se solicitan, así como su comportamiento. Los entornos pueden ser de ejecución y diseño.

Eventos – Mecanismos de comunicación entre componentes de un mismo entorno sobre cambios en su estado o circunstancias especiales como pueden ser las excepciones.

[7] Aplicaciones distribuidas: sistemas diseñados para soportar el desarrollo de aplicaciones y servicios los cuales explotan una arquitectura física que consta de múltiples elementos de procesamiento autónomos que no comparten memoria principal pero cooperan entre sí mediante el envío de mensajes asíncronos sobre una red de comunicaciones.

Reutilización – Habilidad de un componente software de ser utilizado en contextos distintos a los que fue diseñado. Existen 4 modalidades de reutilización: caja blanca, caja de cristal, caja gris y caja negra.

Contratos – Especificación que se añade a la interfaz de un componente y que establece las condiciones de uso e implementación que ligan a los clientes y proveedores del componente. Los contratos cubren aspectos tanto funcionales como no funcionales (calidad de servicio, fiabilidad o seguridad).

Polimorfismo – Habilidad de un mismo componente de mostrarse de diferentes formas, dependiendo del contexto; o bien la capacidad de distintos componentes de mostrar un mismo comportamiento en un contexto dado. La programación orientada a componentes muestra tres nuevas posibilidades:

- La capacidad de un componente de reemplazar a otro en una aplicación, sin romper los contratos con sus clientes (también conocido por polimorfismo de subtipos o de inclusión).
- El polimorfismo paramétrico o implementación genérica de un componente el cual se resuelve en tiempo de ejecución.
- El polimorfismo acotado combina los polimorfismos de inclusión y paramétrico para poder realizar restricciones sobre los tipos en los que se puede parametrizar un componente.

Seguridad – Garantía que debe ofrecer un componente de respetar sus interfaces y contratos.

Reflexión – Habilidad de una entidad software de conocer o modificar su estado.

1.3.2.2. Beneficios del desarrollo basado en componentes

El desarrollo basado en componentes se basa en la reutilización de software lo que proporciona un incremento en la productividad. Con el desarrollo de aplicaciones basadas en componentes, se logra aumentar la capacidad de satisfacción de demanda gracias a la aceleración de los procesos que forman parte del ciclo de vida del producto. Además, simplifica el diseño gracias a que el funcionamiento de cada componente ya es conocido y la implementación se reduce a la configuración e instanciación de estos por lo que el proceso de desarrollo es mucho más rápido. Un componente puede ser construido y luego mejorado continuamente aumentando así la calidad del sistema. Las pruebas de software se les pueden hacer a cada

uno de los componentes individualmente. Además, se pueden actualizar y agregar componentes según sea necesario, sin que se afecte otra parte del sistema.

1.3.2.3. Retos actuales del desarrollo basado en componentes

La programación orientada a componentes es una disciplina relativamente joven que se enfrenta con varios problemas. Los diseñadores de componentes no conocen quién, cómo y dónde se utilizará el componente y esto en ocasiones dificulta el diseño. Cada componente debe adaptarse a las necesidades y requisitos de la aplicación en la que se usará, sin tener que manipular su implementación. La garantía de un componente solo se puede asegurar por una entidad certificadora y no con un simple análisis del mismo.

Las empresas que deciden optar por el desarrollo basado en componentes se encuentran con una falta importante de suministradores de componentes que cumplan con un estándar industrial requerido. En parte se debe a la inmadurez del mercado frente a otros sectores de la industria del software. Cuando se adopta una tecnología de componentes se requiere un nuevo proceso de ingeniería, con nuevas fases y nuevos agentes lo que implica un costo por el cambio que requiere la nueva ingeniería. En muchas ocasiones los componentes que se adquieren comercialmente no están bien documentados provocando que sea difícil adicionar nuevas funcionalidades. Otra desventaja es la poca estabilidad de las tecnologías de componentes debido a que es necesario introducir diferencias en los productos para conseguir ventajas competitivas.

1.4. Aplicaciones web

Las aplicaciones web son software diseñados para funcionar sobre un servidor web y estas se visualizan mediante un navegador. Entre los lenguajes de programación empleados para el desarrollo de aplicaciones web se encuentran PHP, Javascript, Perl y Python. Normalmente una web esta estructurada como una aplicación de tres-capas.

- Presentación: es la encargada de visualizar el contenido.

- Negocio: es la encargada de escribir el codigo fuente mediante una secuencia ordenada y coherente de los requisitos funcionales identificados para la aplicación.
- Datos: está constituida generalmente por una base de datos y es la encargada de almacenar y suministrar información de la capa de negocio [12].

El desarrollo basado en componentes se evidencia por igual en el desarrollo de aplicaciones web. El lenguaje de programación PHP (ideal para este tipo de aplicaciones) y los diferentes marcos de trabajo implementados en dicho lenguaje posibilitan la construcción de componentes y bibliotecas que facilitan y agilizan el diseño y construcción de las interfaces. En el desarrollo web se siguen los mismos pasos y actividades de la ingeniería y el desarrollo basado en componentes abordadas en secciones anteriores de este capítulo.

1.5. Estudio de sistemas homólogos

1.5.1. Alfresco

Enterprise Content Managment o Gestor de Contenido Empresarial (ECM) Alfresco es considerado el líder en soluciones de código abierto para la web y gestión de contenidos. Lo que hace a Alfresco único son sus servicios, estos se pueden ampliar fácilmente con secuencias de comando web a través de servicios REST [13].

Los siguientes componentes se observan en la interfaz web del ECM Alfresco y fueron tomados de [14]:

SortLink: permite al usuario ordenar un conjunto de datos haciendo clic en un enlace HTML.

Progressive panel: permite al usuario ocultar y/o mostrar las áreas de la interfaz de usuario para visualizar áreas más complejas de la interfaz de usuario según sea necesario. También permite al desarrollador rodear una zona de la interfaz de usuario con una frontera gráfica atractiva.

ModeList: muestra una lista gráfica de los elementos, cada uno con una etiqueta y la imagen de icono opcional. Sólo uno de los elementos se pueden seleccionar en cualquier momento. El elemento seleccionado se muestra en un estilo diferente.

Breadcrumb: (también conocido como migas de navegación) permite la manipulación y la visualización de una cadena de ruta separada. Muestra enlaces HTML para permitir al usuario seleccionar inmediatamente cualquier parte de la ruta de navegación.

Además en la interfaz web del software se observan otros componentes como tabla, formulario, acordeón[8] y menú.

1.5.2. KnowledgeTree

KnowledgeTree es un sistema comercial de gestión documental de código abierto. Cuenta con una gestión de documentos simple y eficaz, destacándose varios puntos positivos tales como: una interfaz estándar, una búsqueda avanzada que ofrece respuestas satisfactorias a las búsquedas complejas, modos de navegación virtual además de gestionar los metadatos[9] desde aplicaciones ofimáticas [15]. Es fácil configurarlo a través de las herramientas de administración del software. Se puede ejecutar tanto en Windows como en varias distribuciones de Linux. En el software se pueden observar componentes como: breadcrumb, datepicker[10] y tabla.

1.5.3. Nuxeo

Se trata de una solución completa de gestión de contenido empresarial, en entorno Java J2EE: metadatos, tipos de documentos, flujo de trabajo avanzado, gestión de categorías, funciones de colaboración, búsqueda, gestión de contenido complejo (web, multiarchivos, estructurados), gestión multi-bases [15]. La arquitectura flexible del sistema permite gestionar documentos, realizar versiones, publicación remota y búsqueda

[8] Acordeón: componente para mostrar gran cantidad de contenido con una lista de enlaces.

[9] Metadatos: información que permite clasificar los documentos en una estructura predefinida, limitar o expandir el acceso y uso y recoger los eventos que le han sucedido a un documento a lo largo del tiempo.

[10] Datepicker: control que permite seleccionar una fecha en un calendario.

avanzada, entre otras. En el software se pueden observar componentes como: tabla formulario, buscador y breadcrumb.

1.5.4. OpenKM

OpenKM es un sistema de gestión documental de código abierto que debido a sus características, puede usarse tanto en grandes empresas como en las PYMES[11]. Es una herramienta muy útil para gestión del conocimiento, proporcionando una alternativa flexible y con menores costes que otras aplicaciones propietarias [16]. En el software se pueden observar componentes como: formulario, editor de documentos y acordeón

1.5.5. Resultados del estudio

Luego de un análisis de los componentes visuales presentes en cada uno de los sistemas antes mencionados y un estudio de los requerimientos del GDA eXcriba fueron identificados varios componentes que satisfacen algunos de estos requerimientos y por tanto serán candidatos para formar parte de la biblioteca a desarrollar. Algunos de los componentes identificados fueron: tabla, formulario, breadcrumb, acordeón entre otros.

1.6. Metodología de desarrollo de software

Proceso Racional Unificado o *Rational Unified Process* (RUP)

Es un proceso para el desarrollo de un proyecto de software que define claramente quién, cómo, cuándo y qué debe hacerse en el proyecto [17]. Consiste en un conjunto de actividades necesarias para transformar los requerimientos del usuario en el sistema de software. Es orientado a objetos, preparado para desarrollar grandes proyectos, dirigido por casos de uso, iterativo e incremental y centrado en la arquitectura.

[11] PYMES: pequeñas y medianas empresas.

Modelo de Capacidad y Madurez Integrado o *Capability Maturity Model Integration* (CMMI)

Es una colección de mejores prácticas que ayudan a las organizaciones a mejorar sus procesos [18]. Dichas prácticas cubren todo el ciclo de vida del producto de software. CMMI permite identificar debilidades y definir mejoras. La Universidad de las Ciencias Informáticas (UCI) ha alcanzado el nivel 2 de 5 niveles que propone CMMI. Para alcanzar este nivel es necesario implantar procesos tales como: planificación, seguimiento y control de proyectos y aseguramiento de la calidad entre otros.

Para el desarrollo de la biblioteca de componentes se decidió usar la metodología RUP por las siguientes razones:

- Reduce riesgos al dividir los proyectos en pequeños ciclos o iteraciones.
- Se adapta a las necesidades de los proyectos.
- Provee al equipo de desarrollo una amplia documentación de los procesos.
- Definida para el desarrollo y evolución del sistema eXcriba al cual se le integrará el módulo.
- La unión con CMMI (nivel 2) proporcionará mayor productividad y mayor calidad a la solución.

1.7. Lenguajes

Lenguaje Unificado de Modelado o *Unified Modeling Language* (UML)

Es un lenguaje gráfico de modelado para la especificación, visualización, construcción y documentación de sistemas. UML permite modelar diferentes vistas de un sistema por medio de diagramas, aportando diferentes perspectivas y niveles de detalle que facilitan su comprensión. UML define varios tipos de diagramas tales como: diagrama de casos de uso, diagrama de secuencia, diagrama de comunicación, diagrama de paquetes entre otros [19].

Lenguaje de Marcado de Hipertexto o *HyperText Markup Language* (HTML)

Es el lenguaje que se utiliza para crear las páginas web. Este lenguaje indica a los navegadores cómo deben mostrar el contenido de una página *web*. El lenguaje HTML contiene dos partes: el contenido, que es el texto que se verá en la pantalla de un ordenador y las etiquetas y atributos que estructuran el texto de la página web en encabezados, párrafos, listas, enlaces y normalmente no se muestra en pantalla [20].

Procesador de Hipertexto o *HyperText preprocessor* (PHP)

Es un lenguaje sencillo, de sintaxis cómoda y similar a la de otros lenguajes como Perl, C y C++. Es rápido, interpretado, orientado a objetos y multiplataforma. Posee una multitud de bibliotecas que lo hacen un lenguaje ideal tanto para aprender a desarrollar aplicaciones web como para desarrollar aplicaciones complejas [21].

Se decide utilizar como lenguaje de programación PHP, debido a que el módulo será integrado a al GDA eXcriba y en la arquitectura del sistema se definió que PHP sería el utilizado para la implementación de sus componentes.

JavaScript

Es un lenguaje de scripts desarrollado por Netscape para incrementar las funcionalidades del lenguaje HTML. Sus características más importantes se muestran a continuación y fueron tomadas de [22].

- Interpretado: no requiere compilación. El navegador del usuario se encarga de interpretar las sentencias JavaScript contenidas en una página HTML y ejecutarlas adecuadamente.
- Orientado a eventos: cuando un usuario pincha sobre un enlace o mueve el puntero sobre una imagen se produce un evento. Mediante JavaScript se pueden desarrollar *scripts* que ejecuten acciones en respuesta a estos eventos.
- Orientado a objetos: el modelo de objetos de JavaScript está reducido y simplificado, pero incluye los elementos necesarios para que los *scripts* puedan acceder a la información de una página y puedan actuar sobre la interfaz del navegador.

1.8. Herramientas

Visual Paradigm

Forma parte de la familia de las herramientas CASE[12] las cuales son un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de *software* y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software [23]. Es una herramienta profesional que permite crear diagramas UML para el desarrollo de aplicaciones de software en varios lenguajes de programación. Permite reducir la duración del ciclo de vida completo del desarrollo de software y proporciona un lenguaje común para todo el equipo de desarrollo. Es fácil de instalar y actualizar.

Entorno de desarrollo Zend Studio

Es un ambiente integrado de desarrollo (PHP IDE) disponible para los desarrolladores profesionales que ofrece las capacidades necesarias para desarrollar aplicaciones de negocio. Las características como refactorización, la generación de código, asistente de código y el análisis semántico se combinan para permitir el desarrollo rápido de aplicaciones tanto en el lado del servidor (en PHP) y el lado del navegador (en JavaScript) [24].

1.9. Tecnologías

jQuery

Es una biblioteca de JavaScript que simplifica el manejo con HTML, de los eventos, de las animaciones y las interacciones Ajax[13] para el desarrollo, posibilitando agilizar el desarrollo web. jQuery está diseñado para cambiar la forma en que se escribe JavaScript [25]. jQuery es de fácil aprendizaje y es usada por la facilidad de funcionalidades que ya vienen definidas, además de que

[12] *Computer Aided Software Engineering* o Ingeniería de Software Asistida por Ordenador.

[13] *Ajax: grupo de técnicas de desarrollo web que se utilizan en el lado del cliente para crear aplicaciones web asíncronas.*

simplifica código y facilita la implementación. Se utiliza jQuery versión 1.3.2 como marco de trabajo de JavaScript para el desarrollo de la solución ya que posee una amplia variedad de *plugins*, componentes visuales y utilidades que facilitan el trabajo con las interfaces de usuarios.

CodeIgniter

Es un marco de trabajo para el desarrollo de aplicaciones escritas en lenguaje de programación PHP. Utiliza el patrón arquitectónico Modelo-Vista-Controlador (MVC), un estilo de programación en el que la aplicación está separada en tres capas: Modelo: es el que procesa/obtiene los datos. Generalmente, se usará sobre todo para gestionar la entrada y salida de los datos en la base de datos. Vista: llamada desde el controlador, es la que forma los datos para representarlos en pantalla. En CodeIgniter (y cualquier marco de trabajo para web) es la que montará todo el código HTML. Controlador: como su nombre indica, es el que "controla" lo que pasa en la aplicación web. Básicamente y a grandes rasgos, un controlador recibe una petición, obtiene datos de un modelo, los procesa y se los envía a la vista para que los muestre de forma adecuada [26].

En la arquitectura del GDA eXcriba está definido el uso de CodeIgniter versión 1.7.2 para el desarrollo de sus módulos por la variedad de funcionalidades ya existentes y bien documentadas que posee.

En este capítulo se dieron a conocer elementos teóricos del desarrollo basado en componentes necesarios para sustentar la solución del problema. EL análisis de algunos sistemas de gestión documental contribuyó a identificar componentes candidatos para formar parte de la biblioteca a desarrollar. Además, se especificaron las herramientas, lenguajes y metodología que se usarán durante el proceso de desarrollo.

Capítulo 2

Propuesta de solución

En este capítulo se realiza un estudio crítico de la situación actual que presenta el GDA eXcriba referente a los componentes que conforman su interfaz web y se propone una solución para la problemática existente. Además, se definen los requisitos funcionales y no funcionales así como los casos de uso que servirán de base para una futura implementación del módulo.

2.1. Situación actual de los componentes del sistema

A continuación se realiza un análisis del sistema en relación al desarrollo de sus componentes. El GDA eXcriba está compuesto entre otras soluciones por la interfaz de usuario (aplicación web) y el ECM Alfresco como núcleo del sistema. Algunos de los componentes visuales que se observan en él son: explorador (explorer), migas de navegación (breadcrumb), ventana emerjente, acordeón, menú desplegable, notificaciones, formulario, tabla, vista de árbol (treeview) y creador de texto a partir de plantillas[1].

El código de la versión estable del eXcriba web está dividido en los siguientes 15 módulos los cuales hacen uso de los componentes visuales:

1. **excriba-core** – Implementación de la arquitectura base, conjunto de funciones comunes, principales subsistemas y aspectos arquitectónicamente significativos. Este módulo hace uso del componente explorer.

[1]Componente del tipo wysiwyg:*what you see is what you get* (lo que ves es lo que tienes).

2. **x-accordion** – Componente en forma de módulo que gestiona la navegación por carpetas en formas de árbol, así como las distintas zonas de navegación, incluye soporte además para gestión de portapapeles. Este módulo hace uso también del componente vista de árbol.
3. **x-audit** – Implementa los requerimientos de auditoría sobre los documentos y carpetas. Este módulo hace uso de los componentes tabla y formulario.
4. **x-basic-operation** – Implementa los requerimientos de las operaciones comunes que se realizan con la herramienta tales como copiar, mover, editar metadatos de documentos y carpetas, entre otros. Este módulo hace uso de los componentes tabla y formulario.
5. **x-breadcrumb** – Implementa los requerimientos de las migas de navegación, historial de los últimos visitados y navegación por caminos.
6. **x-classification** – Implementa la gestión de cuadro de clasificación funcional. Este módulo hace uso de los componentes tabla y acordeón.
7. **x-global-actions** – Desarrolla los requerimientos de las acciones globales asignadas a la dirección de navegación en que se encuentre el usuario en cada momento.
8. **x-control-version** – Implementa los requerimientos de control de versiones sobre los documentos. Este módulo hace uso del componente tabla.
9. **x-header** – Gestiona la vista de interfaz de usuario referente a la cabecera del software.
10. **x-loader** – Da soporte a la manipulación mediante AJAX, a la comunicación entre la capa de presentación y la lógica de negocio.
11. **x-permissions** – Gestiona el control de acceso de los documentos y carpetas. Este módulo hace uso de los componentes tabla, notificaciones y ventana emerjente.
12. **x-record** – Implementa los requerimientos de gestión de expedientes. Este módulo hace uso de los componentes tabla, formulario, ventana emerjente y migas de navegación.
13. **x-rules** – Implementa los requerimientos de reglas de negocio.

14. **x-search** – Da soporte a los requerimientos de búsqueda y recuperación de información. Este módulo hace uso del componente menú desplegable.
15. **x-toolbar** – Soporta los requerimientos que por su importancia deben existir en una zona de visibilidad alta del sistema.

2.1.1. Análisis de componentes

Luego de identificar los componentes por cada uno de los módulos es necesario realizar un estudio sobre algunos de ellos para identificar como se comportan en el sistema. Se seleccionan explorer, migas de navegación y ventana emerjente por ser de los más usados.

2.1.1.1. Explorer

Este componente permite la navegación en el repositorio documental en forma de directorios y archivos similar a los exploradores en los sistemas operativos. Algunos de los módulos que hacen uso de este componente son: cuadro de clasificación[2], gestión de expediente[3] y gestión de entidades[4] [27].

Entre sus principales características se encuentran:

1. Permite realizar acciones sobre nodos:
   - Mostrar los metadatos asociados.
   - Navegar jerárquicamente.
   - Asociar acciones.
2. Permite configurar:
   - La internacionalización.

[2] x-classification: https://repositorio.cenia.prod.uci.cu/svn/eXcriba/source/trunk/x-classification/.
[3] x-record: https://repositorio.cenia.prod.uci.cu/svn/eXcriba/source/trunk/x-record/.
[4] entity-manager: https://repositorio.cenia.prod.uci.cu/svn/eXcriba/source/trunk/entity-manager/.

- La columna por la cual se hace la navegación de los nodos hijos.
- El directorio raíz sobre el cual se va a usar.
- Metadatos.
- Imágenes configuradas para cada tipo de acción.
- Tipos de nodos que no deben ser visualizados .
- Acciones que se les permite realizar a los nodos basándose en criterios como: lista de permisos, tipos, aspectos y valor de una determinada propiedad.

Como se muestra en la figura 2.1 (Anexo A) el explorer se encuentra dentro del paquete libraries. La clase principal Explorer está compuesta por las clases Explorer_model y Explorer_ui_render. La primera es la encargada de poblar el componente y contiene los datos de configuración, los datos a mostrar, los datos de configuración de acciones y las acciones propias. Mientras que la segunda es la que lo representa en HTML.

Manejo de instancia, configuración y uso.

A modo de conclusión se pudo notar que inicialmente, no se tuvo en cuenta el manejo de múltiples instancias de componentes. Para resolver el problema se creó un subsistema capaz de virtualizar los datos de sesión según un contexto[5], donde cada componente modifica solo la información asociada al contexto al cual pertenece. El subsistema permite separar la sesión en un área particionable. Cada una va a tener un identificador de forma que, la modificación del valor de los datos de un contexto con la misma clave no afecte los datos del otro.

Mediante el subsistema de gestión de eventos se creó un nuevo tipo de evento, (de contexto) con la función de cargar las configuraciones adecuadas de acuerdo al contexto actual. La principal dificultad que se deriva es que por cada instancia se tiene un escuchador distinto.

Para usar el componente explorer se llama a un *helper* (encargado de crear y configurar el contexto) desde una vista y se crea el explorer adecuado según sus propios requisitos.

[5]contexto: abordado anteriormente como forma de manejar las instancias de un componente.

2.1.1.2. Migas de navegación (breadcrumb)

Las migas de navegación es un componente para llevar pistas de los últimos directorios visitados en el sistema. Se actualiza tras cada visita a un nodo llevando registro de los últimos n-nodos por los cuales pasó. La figura 2.3 (Anexo A) muestra la estructura de clases. El controlador Breadcrumb se relaciona con la capa de aplicación y contiene los siguientes métodos: update_id, se encarga de mover el sistema al nodo deseado; location, permite ver la representación en camino de un nodo visitado y refresh, permite obtener los caminos de los nodos hijos del nodo que se esta visitando.

X_breadcrumb_lib_loader crea la vista Breadcrumb y además hace uso de la clase Breadcrumb_utils encargada de cachear los últimos n-nodos visitados de manera que no se repitan.

Manejo de instancia, configuración y uso.

En este caso tampoco existe gestión de múltiples instancias. No tiene un método integrado para combinar la gestión desde la capa de aplicación y la de presentación. Para darle solución a ello se utiliza un parámetro adicional en la comunicación de las capas mencionadas que corresponde con el fichero de configuración del explorer. A su vez esto provoca cierta dependencia por lo que es imposible separar una instancia de ese componente.

El breadcrumb configura los *javascripts* y los estilos con los que se va a inicializar y el directorio raíz donde está inicializado. Su uso se realiza también por medio de una vista que llama a un *helper*.

2.1.1.3. Ventana emerjente

Es un componente para embeber información en forma de ventana emergente, tales como alertas, errores y cuadros de diálogos. Como se muestra en la figura 2.3 (Anexo A) la clase Delete_node crea la vista de igual nombre. Esta a su vez se comunica con el controlador x_basic que es el encargado de pasarle la acción a ejecutar.

En el caso de las ventanas emerjente si se maneja la gestión de instancias. A su vez tiene como limitante que el componente no controla la información que muestra, lo que significa que los elementos en su interior no son instanciables. Se usa mediante una acción disparada por un mecanismo de gestión de acciones.

2.1.1.4. Resultados

A pesar de darle solución a la gestión de instancias y la configuración existen una serie de dificultades referente a los componentes:

1. No se respeta la característica: accesible solo a través de su interfaz, lo que provoca poca reutilización de código.
2. No son identificables provocando que no se puedan:
   - Almacenar en un repositorio.
   - Separarlos de los módulos donde se encuentran.
   - Usar en otro entorno distinto al que fueron concebidos.
3. No existe un mecanismo de comunicación establecido. Esta tarea debe ser realizada por algún desarrollador experto en el tema.
4. No están bien documentados, lo que dificulta la adición de nuevas funcionalidades.

2.2. Propuesta del sistema

Se propone desarrollar una biblioteca de componentes para la interfaz web del GDA eXcriba, que facilite a los programadores utilizar, instanciar y crear nuevos componentes, permitiendo mayor rapidez y simplicidad en el flujo de implementación.

2.2.1. Componentes propuestos

Luego de un estudio de la interfaz de algunos sistemas homólogos, un análisis crítico del eXcriba y de los módulos en desarrollo se obtuvo una lista con los componentes candidatos que conformarán la biblioteca. Debido al alcance del presente trabajo solo se implementara el componente tabla. Los restantes serán agregados y documentados en futuras versiones de la biblioteca. Los componentes seleccionados son:

- Tabla – Estructura para guardar datos, organizar, posesionar o dar mejor formato a los textos y gráficos en una página web.
- Ventana emerjente (en inglés Pop-Up) – Ventanas que emergen automáticamente generalmente sin que el usuario lo solicite. Permite visualizar información en forma de mensaje. Existen varios tipos de ventana que muestran distinta información, un ejemplo es una ventana mostrando un mensaje de error.
- Examinar (en inglés filechooser) – Componente para seleccionar y/o grabar cualquier archivo dentro de un ordenador.
- Explorador (en inglés explorer) – Permite visualizar mayormente el contenido principal de un sistema de gestión documental.
- Formulario – Componente usado generalmente para guardar información. Es una ventana de programa en la cual se indica que datos se requieren para ser ingresados en un repositorio o base de datos.
- Paginador – Permite navegar con mayor facilidad dentro de una tabla y/o sitio web, posibilitando que la información que se muestra esté dividida en varias páginas.
- Región de búsqueda (en inglés Searchlist) – Componente para visualizar el resultado de una búsqueda en el sistema.
- Acordeón – Permite mostrar gran cantidad de contenido con una lista de enlaces recomendados a las categorias de un sitio con el objetivo de darle un aspecto más elegante.
- Árbol (en inglés treeview) – Componente para visualizar datos de forma jerárquica.

2.3. Modelo de dominio

"El modelo de dominio captura los tipos más importantes de objetos en el contexto del sistema. Los objetos del dominio representan las cosas que existen o los eventos que suceden en el entorno en el que trabaja el sistema"[17].

El mismo se realiza cuando no se tiene una visión clara del negocio ni de las personas que participan en sus procesos. Se representa en forma de diagrama.

2.3.1. Descripción de las clases del dominio

Infraestructura – Conjunto de clases y bibliotecas para la gestión de los componentes.

Componente – Elemento de un sistema software que ofrece un conjunto de servicios, o funcionalidades, a través de interfaces definidas.

Instancia – Objeto concreto o resultado de una clase.

Identificador – Característica que hace un componente único.

Instalador – Permite instalar y gestionar un componente dentro de la infraestructura.

Documentación – Manual o guía de uso y desarrollo de los componentes, de manera que el usuario o desarrollador pueda aprender su manejo y consultar cualquier duda ante un problema desconocido.

2.3.2. Diagrama del modelo de dominio

Anexo A de la versión extendida del documento.

2.4. Requerimiento de software

Una buena captura de requisitos influye en el éxito de un software, mientras que una mala definición y especificación de los mismos puede provocar que fallen muchos proyectos. *"Un requisito es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de éste. En el otro extremo, es una definición detallada y formal de una función del sistema"* [28].

Requisitos funcionales

"Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir, no alteran la funcionalidad del producto, por lo que se mantienen invariables sin importarle con que propiedades o cualidades se relacionen" [28].

Requisitos no funcionales

"Los requisitos no funcionales son los requerimientos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste, como la fiabilidad y el tiempo de respuesta" [28].

A continuación se relacionan los requerimientos del módulo biblioteca de componentes para la interfaz web del GDA eXcriba.

1. **Infraestructura**

   Requisitos funcionales.

   RF1 Gestionar componentes.

   RF1.1 - Instalar componente.

   Descripción: El sistema deberá instalar los componentes necesarios.

   Entradas: dirección del fichero en el ordenador. (Obligatorio).

   Salidas: mensaje de error si se carga un fichero que no sea un componente y/o si pesa más de lo permitido.

   Prioridad: Alta.

   Complejidad: Alta.

   RF1.2 - Listar los datos de los componentes instalados.

   Descripción: El sistema mostrará la lista de los componentes instalados.

   Entradas: acción sobre el enlace de instalación. El formato de la entrada es la acción del usuario sobre el botón instalar. (Obligatorio).

   Salidas: muestra una tabla con los datos del componente con los campos (ID, Nombre, descripción, título, versión y dependencia.).

   Prioridad: Alta.

Complejidad: Media.

RF1.3 - Acceder al manual de uso del componente instalado.

Descripción: el sistema mostrará la documentación del componente solicitado.

Entradas: acción sobre hipervínculo que representa el ID del componente. (Obligatorio).

Salidas: muestra la interfaz con la documentación del componente solicitado.

Prioridad: Crítico.

Complejidad: Media.

2. **Componente Tabla**

   Requisitos funcionales

   RF2.1 Listar datos.

   Descripción: La tabla mostrará los elementos que contiene.

   Entradas: acción del usuario sobre alguna funcionalidad de la tabla. (Obligatorio).

   Salidas: muestra los elementos de acuerdo a la acción ejecutada.

   Prioridad: Crítico.

   Complejidad: Alta.

   RF2.2 Buscar elemento.

   Descripción: la tabla dará la posibilidad de buscar un elemento a partir del parámetros de búsqueda.

   Entradas: selección del parámetro de búsqueda e introducción del nombre del elemento a buscar. El formato de la entrada es una cadena entre 1 y 225 caracteres (Obligatorio).

   Salidas: muestra en la tabla los elementos que coinciden con el término de búsqueda.

   Prioridad: Crítico.

Complejidad: Alta.

RF2.3 Actualizar datos.

Descripción: la tabla dará la posibilidad de actualizarse tras ocurrido algun cambio.

Entradas: selección de la opción recargar. El formato de la entrada es la acción del usuario sobre el hipervínculo que representa la opción recargar. (Obligatorio).

Salidas: muestra la tabla actualizada.

Prioridad: Crítico.

Complejidad: Alta.

RF2.4 Ordenar columnas.

Descripción: la tabla dará la posibilidad de ordenar la tabla de acuerdo a la columna especificada.

Entradas: selección de la columna por la cual se desea ordenar. (Obligatorio).

Salidas: muestra en la tabla ordenada por la columna especificada.

Prioridad: Media.

Complejidad: Media.

RF2.5 Ocultar columnas.

Descripción: la tabla dará la posibilidad de ocultar las columna no deseadas.

Entradas: selección de la columna que se desea ocultar. (Obligatorio).

Salidas: muestra en la tabla actualizada.

Prioridad: Media.

Complejidad: Media.

RF2.6 Gestionar página de resultado.

RF2.6.1 - Cambiar la cantidad de elementos a mostrar.

Descripción: la tabla dará la posibilidad de mostrar en una página la cantidad de elementos seleccionados por el usuario.

Entradas: seleccion de la cantidad de elementos a mostrar. (Obligatorio).

Salidas: muestra en la tabla la cantidad de elementos según el valor seleccionado.

Prioridad: Media.

Complejidad: Media.

RF2.6.2 - Ir a la página siguiente.

Descripción: la tabla permitirá ir a la siguiente página.

Entradas: acción en la interfaz para ir a la página siguiente. (Obligatorio).

Salidas: muestra en la tabla los elementos de la página solicitada.

Prioridad: Media.

Complejidad: Media.

RF2.6.3 - Regresar a la página anterior.

Descripción: la tabla permitirá regresar a la página anterior.

Entradas: acción en la interfaz para ir a la página anterior. (Obligatorio).

Salidas: muestra en la tabla los elementos de la página solicitada.

Prioridad: Media.

Complejidad: Media.

RF2.6.4 - Ir a la última página.

Descripción: la tabla permitirá ir a la última página.

Entradas: acción en la interfaz para ir a la última página. (Obligatorio).

Salidas: muestra en la tabla los elementos de la página solicitada.

Prioridad: Media.

Complejidad: Media.

RF2.6.5 - Regresar a la primera página.

Descripción: la tabla permitirá regresar a la primera página.

Entradas: acción en la interfaz para ir a la primera página (Obligatorio).

Salidas: muestra en la tabla los elementos de la página solicitada.

Prioridad: Media.

Complejidad: Media.

RF2.6.6 - Ir a una página deseada.

Descripción: la tabla permitirá ir hacia cualquier página.

Entradas: introduce el número de la página que desea consultar. (Obligatorio).

Salidas: muestra la página solicitada. Muestra un mensaje de error si se introduce un número mayor a la cantidad de páginas.

Prioridad: Media.

Complejidad: Media.

Requisitos no funcionales

Usabilidad

1. La tabla debe ser redimensionable.

2. Los elementos deben cambiar de color cuando se estén señalando.

Portabilidad

1. La tabla debe ser instanciable.

Restricciones de diseño del módulo

1. Utilizar el marco de trabajo jQuery 1.3.2 como biblioteca fundamental para la implementación de las interfaces de usuario.

2. Utilizar el marco de trabajo CodeIgniter 1.7.2 para la implementación del módulo.

3. Implementar la biblioteca en el lenguaje PHP 5.3.

4. Los componentes deben estar internacionalizados.

2.5. Definición de casos de uso del sistema

Un caso de uso es una descripción de un conjunto de secuencias de acciones, incluyendo variaciones, que un sistema lleva a cabo y que conduce a un resultado observable de interés para un actor determinado [17].

2.5.1. Actores del sistema

Un actor es un conjunto de roles que los usuarios de casos de uso desempeñan cuando interaccionan con estos casos de uso [17].

Actores	**Justificación**
Administrador	Especialista que define quién puede hacer qué sobre cada uno de los elementos del sistema.
Usuario	Persona que interactúa con los componentes del sistema.

Tabla 2.1: Definición de los actores

2.5.2. Diagrama de casos de uso del sistema

Anexo A de la versión extendida del documento.

2.5.3. Descripción de casos de uso

Anexo B en la versión extendida del documento.

En este capítulo quedó evidenciada la necesidad de incorporar al sistema de una biblioteca de componentes. A partir de la descripción del problema y la definición de los requisitos funcionales y no funcionales, se podrá diseñar el módulo para que cumpla las expectativas del cliente. Además, la elaboración de los diagramas de casos de uso del sistema y sus descripciones permitirá guiar el desarrollo de la solución.

Capítulo 3

Diseño del sistema

El diseño juega un papel importante en el desarrollo de software debido a que permite generar una serie de diagramas o modelos, que dan paso a la implementación del sistema que se desea construir. Es en esta etapa es donde se establece la calidad del software y se dejan plasmadas visualmente las funcionalidades del mismo para que puedan ser convertidas facilmente en código. En este capítulo se realiza el diagrama de clases del diseño, se describen cada una de las clases y se describe la arquitectura entre otras actividades.

3.1. Modelo de diseño

El modelo de diseño es utilizado para modelar los aspectos dinámicos del sistema. Consta de un conjunto de objetos que describen las realizaciones de los casos de uso y sus relaciones. Se centra en los impactos que producen en el sistema los requisitos funcionales y no funcionales. Constituye una abstracción al modelo de implementación y del código fuente [17].

3.2. Diagrama de clases del diseño

Los diagramas de clases son los más utilizados en el modelado de sistemas orientados a objetos. Un diagrama de clases en sí, muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Los mismos se utilizan para reflejar la vista de diseño estática de un sistema. Son la base para los posteriores diagramas de componentes y los de despliegue. Su importancia radica en que permiten construir sistemas ejecutables aplicando ingeniería directa e inversa [17].

La figura 3.1 muestra el diagrama de clases del diseño del caso de uso Instalar componente. Para consultar los restantes diagramas dirigirse al Anexo B de la versión extendida del documento.

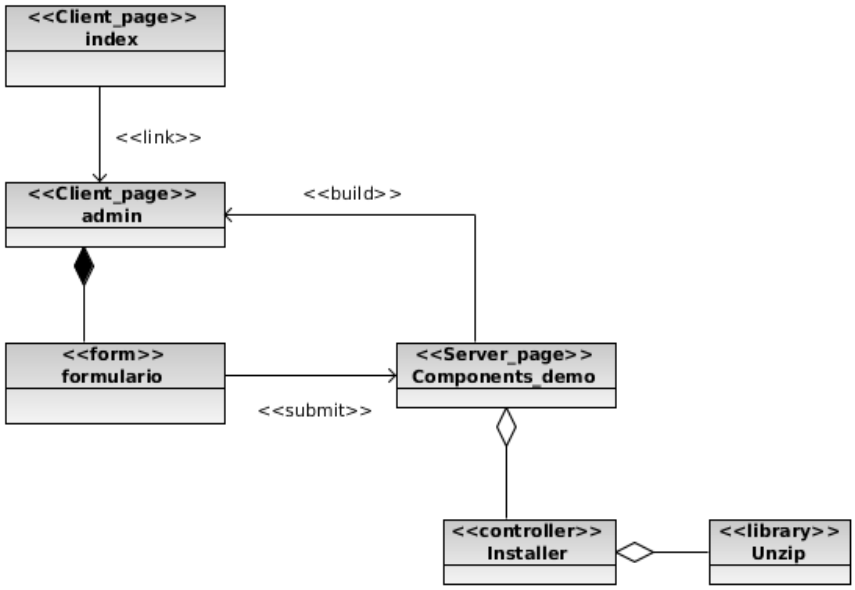


Figura 3.1: Diagrama de clases CU: Instalar componente

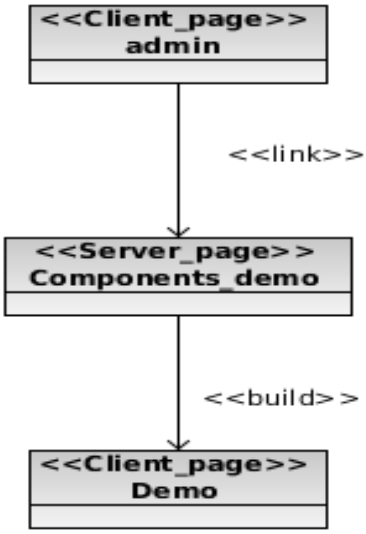


Figura 3.2: Diagrama de clases del CU: Visualizar manual de uso de un componente.

Para modelar los componentes se le asocia el controlador de cada uno al Components_demo y este se encargan de crear su propia vista.

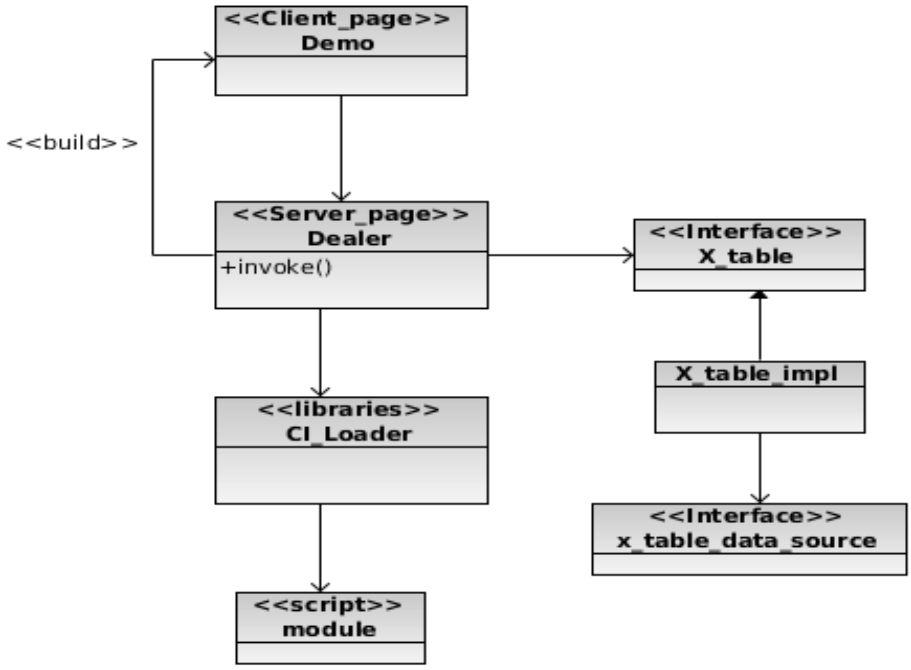


Figura 3.3: Diagrama de clases para el componente tabla.

Todas las funcionalidades de la tabla presentan el diseño de clases que se muestra en la figura C.1. Cada componente seguirá la misma estructura, lo que cambia en cada uno de ellos es la clase que lo implementa y/o la funcionalidad que se esté ejecutando.

3.2.1. Descripción de las clases

A continuación se describen las principales clases controladoras y contenedoras de datos.

Nombre: Installer	
Tipo de clase: Controladora	
Atributo	**Tipo**

$message	
Para cada responsabilidad:	
Nombre:	init()
Descripción:	Inicializa la clase dada la ruta del fichero xmp (paquete de módulos de eXcriba) a instalar.
Nombre:	parse_module_descriptor()
Descripción:	Parsea la descripción del módulo contenida en el fichero module.ini y valida la información.
Nombre:	list_modules()
Descripción:	Lista los módulos instalados
Nombre:	write_install()
Descripción:	Escribe la descripción de un módulo instalado en instaler.ini.
Nombre:	install_component()
Descripción:	Instala componentes en el sistema dada su dirección

Nombre: Components_demo	
Tipo de clase: Controladora	
Atributo	**Tipo**
$message	
Para cada responsabilidad:	
Nombre:	index()
Descripción:	Crea la página inicial.
Nombre:	admin()
Descripción:	Sube el fichero lo instala y lista la información de los componentes instalados.
Nombre:	module()
Descripción:	Redirecciona al controlador de cada módulo.

Nombre: Dealer	
Tipo de clase: Controladora	
Atributo	**Tipo**
$message	
Para cada responsabilidad:	
Nombre:	invoke()
Descripción:	Conecta cualquier instancia de un componente con su implementación.

3.3. Arquitectura de software

La arquitectura de software es un aspecto crítico a tener en cuenta en el desarrollo de aplicaciones ya que describe las partes que conforman un sistema. El GDA eXcriba tiene una arquitectura en capas, la cual usará el módulo propuesto. El modelo en 3 capas simplifica la comprensión y organización del software y cada capa solo se relaciona con la superior y/o subyacente.

Capa de presentación – En esta capa se encuentra el conjunto de interfaces de usuario que brindan la información del sistema al cliente. Esta capa envía los datos entrados por el usuario a la capa de aplicación y muestra la información obtenida de esta. Se comunica únicamente con la capa de aplicación mediante el protocolo HTTP[1].

Capa de aplicación – En esta capa se ejecuta la lógica de negocio. Esta capa obtiene la información que le llega desde la capa de presentación y solicita a la capa de acceso al repositorio los datos necesarios que serán procesados y devuelto al cliente. Controla y dirige el flujo de la aplicación en sentido general. La comunicación entre esta capa y la subyacente es mediante el estilo arquitectonico REST[2].

[1]HTTP: *Hypertext Transfer Protocol* o Protocolo de Transferencia de Hipertexto es el protocolo usado en cada transacción de la *World Wide Web*.

[2]REST *Representational State Transfer* o Transferencia de Estado Representacional: es una técnica de arquitectura software para sistemas hipermedia distribuidos como la *World Wide Web*.

Capa de acceso al repositorio – Esta capa recibe las peticiones desde la capa de aplicación, interactúa directamente con el repositorio de contenidos y obtiene los datos solicitados devolviéndolos a la capa superior. La comunicación entre esta capa y la subyacente es mediante bibliotecas de clase.

La siguiente figura muestra como se integra el módulo biblioteca de componentes en la arquitectura del sistema eXcriba.

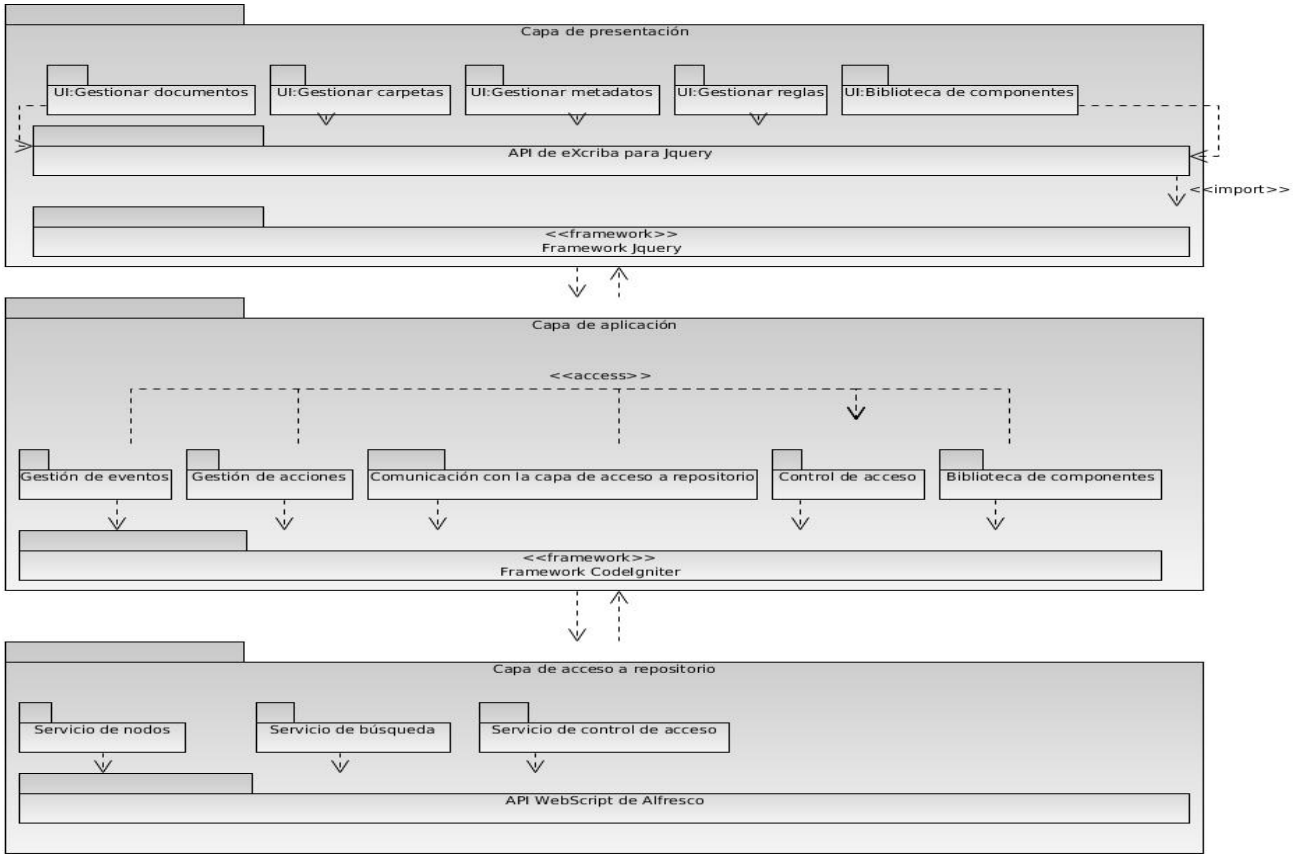


Figura 3.4: Arquitectura del sistema

3.3.1. Arquitectura del módulo biblioteca de componentes

Dentro de la capa de aplicación se creó un subsistema nombrado core que representa la infraestructura encargada de contener las clases y funcionalidades necesarias para que eXcriba soporte componentes. El mismo posee las interfaces que son de interés para la infraestructura, la clase controladora Dealer que se

encarga de conectar la instancia de un componente con su implementación además de las bibliotecas Installer que contiene los métodos que permiten instalar el componente y listar sus datos y Abstract_component la cual conoce el comportamiento de cada componente y gestiona la instancia de los mismos en fichero. Esta lógica está pensada para no cargar la sesión de CodeIgniter ni las *cookies* usando enlaces de id y ficheros de serialización específicos para cada instancia.

Al sistema se le realizaron algunas modificaciones para facilitar el desarrollo del módulo tales como:

- Se redefinió el comportamiento de Matchbox para trabajar con los componentes. Módulo Matchbox es un conjunto de bibliotecas extendidas que permite organizar una aplicación en pequeños componentes o módulos. Los módulos están almacenados dentro de sus propias carpetas y pueden ser usados en otras aplicaciones.

- Se redefinió la biblioteca Template para integrar su mecanismo de gestión con la gestión de los componentes. Template permite múltiples vistas para un solo controlador y embeber vistas dentro de vistas,

- Se agregó la biblioteca Unzip la cual permite extraer archivos en CodeIgniter sin necesidad de instalar cualquier extensión en php.

Los módulos en el sistema eXcriba se construyen usando un arquetipo definido gracias a la inclusión de apache ANT[3] que permite embeber cada módulo dentro de una infraestructura única. Para los componentes que serán creados se hace necesario definir un arquetipo distinto pero usando la misma técnica base debido a que los mismos deben ser independientes y reutilizables.

El arquetipo para los componentes como se muestra en la figura 3.3 se denota con la letra 'c'. La carpeta impl contendrá la clase que define a cada componente. El fichero build.properties es el script de empaquetación, build.xml es el script de compilación mientras que module.php contiene los datos de instalación del componente. Los componentes creados usando este arquetipo se almacenan en el sistema en

[3]Herramienta usada en programación para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción (build).

paquetes de extensión XMP (eXcriba *model package* o modelo de paquetes de eXcriba) los cuales se instalan una vez que se quieran usar los componentes.

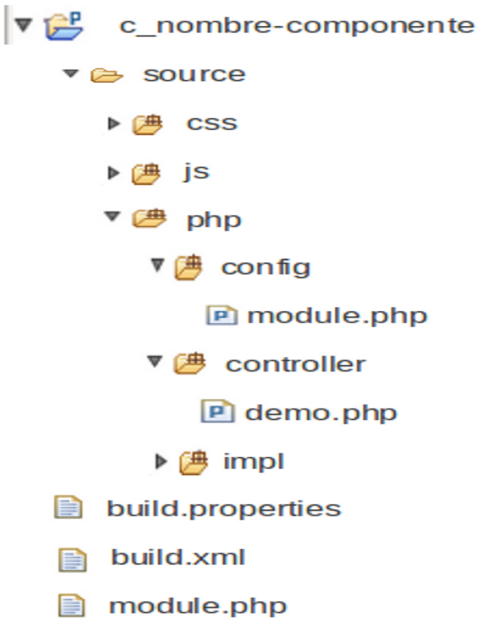


Figura 3.5: Arquetipo de componentes

3.4. Patrones de diseño

"Un patrón es una descripción de un problema y su solución, que recibe un nombre y que puede emplearse en otros contextos; en teoría, indican la manera de utilizarlo en circunstancias diversas" [29].

Patrones GRASP[4]

- Experto asigna una responsabilidad a la clase que tiene la información necesaria para cumplirla.

  Ejemplo: la clase proveedora de datos Static_data_src hace función de experto cuando se desea crear una tabla ya que contiene las siguientes funcionalidades que permiten crear dicho componente: get_datas(), get_path() y get_class_name().

[4] *General Responsibility Asignment Software Patterns*: patrones de los principios generales para asignar responsabilidades.

- Controlador asigna la responsabilidad de las operaciones del sistema a 1os objetos situados en la capa del dominio separándola de la capa de presentación.

  Ejemplo: la clase Demo es el controlador de la vista de ejemplo de como utilizar los componentes de excriba. La misma posee funcionalidades que le permiten realizar dicha acción como: Demo (función que realiza la petición del componente a instalar), index (función que muestra el componente).

- Creador permite identificar quién debe ser el responsable de crear nuevos objetos o clases.

  Ejemplo: la clase Loader es la encargada de crear o instanciar los componentes mediante la función component. También se encarga de verificar si un componente ya fue instanciado anteriormente.

- Alta cohesión define que la información que almacena una clase debe estar relacionada con la misma.

  Ejemplo: este patrón se pone de manifiesto en la clase X_table_impl la cual implementa todas las funcionalidades de la tabla.

- Bajo acoplamiento consiste en tener las clases lo menos ligadas entre sí posible.

  Ejemplo: la clase X_table_impl solo hereda de Abstract_component alcanzando un bajo acoplamiento entre las clases. Las clases que implementan la lógica del negocio y de acceso a datos se encuentran en el modelo, las cuales no tienen asociaciones con las de la vista o el controlador, lo que proporciona que la dependencia en este caso sea baja.

Patrones GoF[5]

- Comand sirve de mediador entre el que realiza la petición y el que le da respuesta.

  Ejemplo: la clase Dealer sirve de comando cuando se desea instalar un componente usando el método invoke (función que invoca al componente que se quiere instalar).

- Factory proporciona un interfaz para crear las familias de objetos relacionados o dependientes sin especificar sus clases concretas.

[5] *Gang of four*: patrones banda de los cuatro.

Ejemplo: la clase Loader funciona como factory si un cliente realiza la petición de crear una tabla pues tiene dentro el método component que es el que instancia el componente.

- Singleton permite manejar distintas sessiones de código y gestionar una sola instancia.

  Ejemplo: este patrón se evidencia en la clase Loader la cual contiene el método ci_load_class que verifica si una instancia ya fue llamada anteriormente.

- Adapter convierte la interfaz de una clase en otra interfaz que otra clase puede usar.

  Ejemplo: se creó Matchmodule que es una adaptación de Matchbox. Matchmodule traduce una petición y busca en module.ini la url del componente.

3.5. Estándares de codificación

Un estándar de codificación es el estilo de codificación que se usa para escribir el código cuando se programa. Cada equipo de desarrollo selecciona el estándar a usar basándose en las características propias del lenguaje de programación, lo que se desea implementar y los estilos propios de los programadores. El estándar que se usará para el desarrollo del módulo es el definido por CodeIgniter y que fue usado en el desarrollo del sistema eXcriba.

Luego de terminado el capítulo se tienen creadas las bases para proceder a una implementación eficaz del módulo biblioteca de componentes. A través de la obtención de los artefactos generados durante el diseño se pudo llegar a una definición más clara de las características que debe poseer el software. Además, teniendo en cuenta la arquitectura MVC que establece CodeIgniter, fue posible obtener una mayor comprensión de la organización del módulo a desarrollar.

Capítulo 4

Implementación y prueba

En el presente capítulo se presenta el diagrama de despliegue y de componentes correspondientes a la propuesta de solución, así como las pruebas que se le debe realizar al producto para garantizar su correcto funcionamiento durante todo su ciclo de vida.

4.1. Diagrama de despliegue

Este diagrama muestra como se despliega físicamente el sistema y en qué hardware se ubicarán sus componentes.

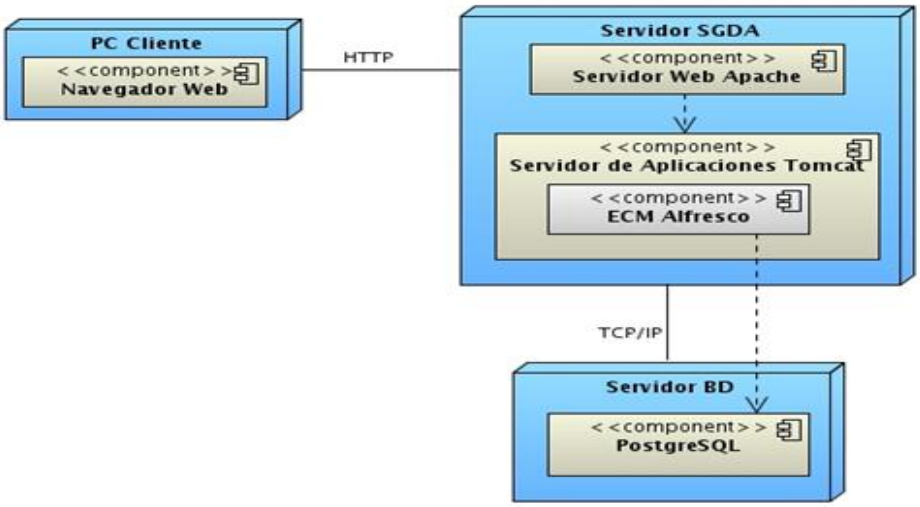


Figura 4.1: Diagrama de despliegue del módulo.

El nodo ordenador cliente hace referencia a las estaciones de trabajo que el usuario utilizará para acceder a la aplicación. Debe tener (obligatorio) un navegador web. El nodo gestión documental contiene el sistema eXcriba que contiene la aplicación y su núcleo el ECM Alfresco. Mientras que el tercer nodo contiene el servidor de base de datos que usa el Alfresco.

4.2. Diagrama de componentes

Este diagrama permite representar un sistema dividido en componentes y mostrando la relación entre ellos. Puede incluir librerías, tablas, archivos, ejecutables y documentos que forman parte del sistema.

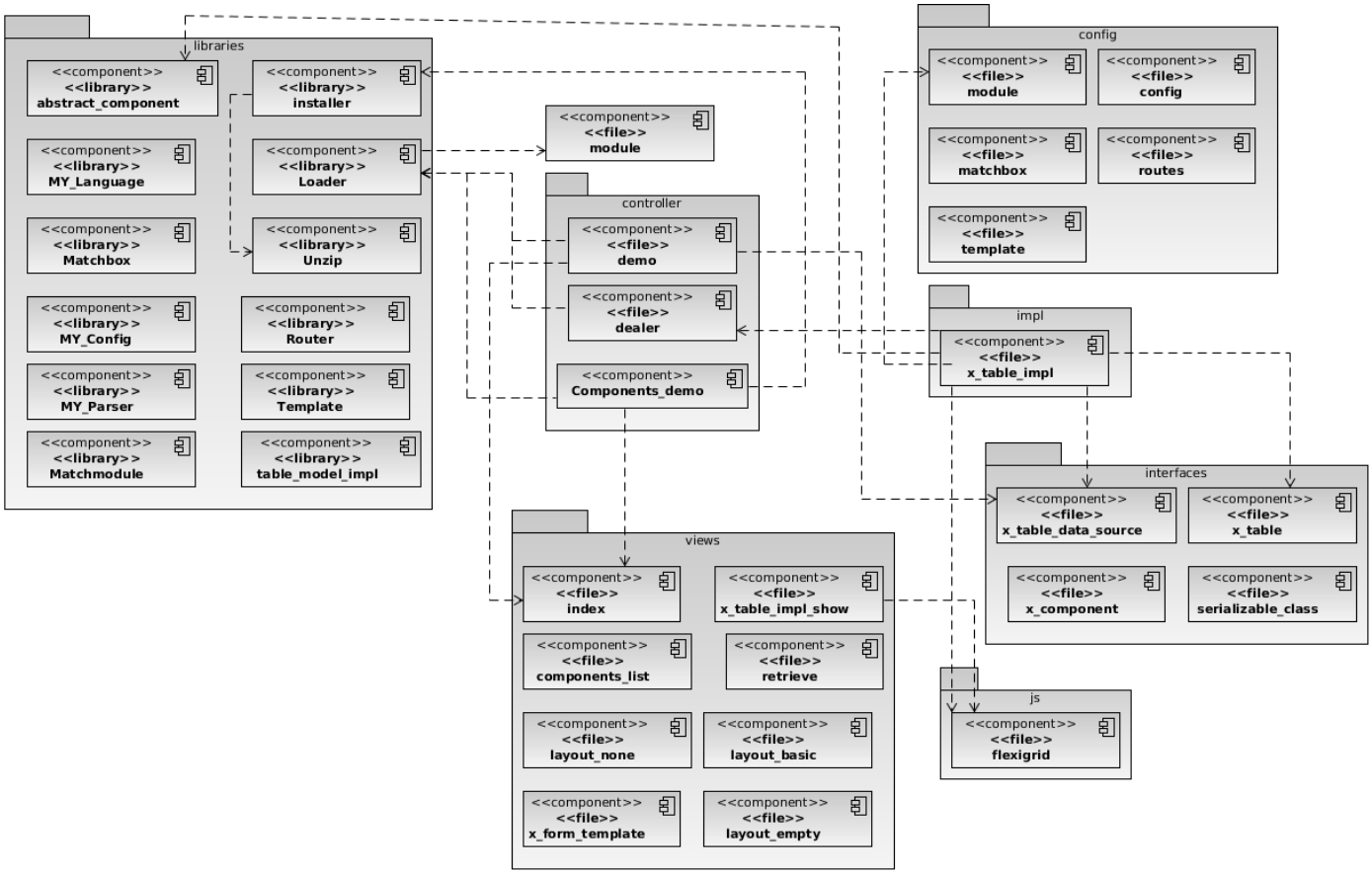


Figura 4.2: Diagrama de componentes del módulo.

4.3. Prueba de software

Las pruebas de software son un elemento fundamental para garantizar la calidad del sistema, estas pruebas representan una revisión final de las especificaciones del diseño y de la codificación, ya que verifican que el software funcione como se diseñó y que los requerimientos se han cumplido, además de documentar los defectos del sistema [9].

4.3.1. Prueba de caja blanca

"La prueba de caja blanca es un método de diseño de casos de prueba que usa la estructura de control de diseño procedimental para obtener los casos de prueba" [9].

Al software implementado se le realizaron pruebas de caja blanca utilizando la técnica del camino básico la cual permitió obtener una medida de la complejidad lógica del diseño.

```
function admin()
{
    $this->load->module_library('core', 'installer');

    if (@is_uploaded_file ( $_FILES ['xmp_file'] ['tmp_name'] ))
    {
        try
        {
            $this->installer->install_component($_FILES ['xmp_file']);
        }
        catch (Exception $ex)
        {
            echo $ex->getMessage();
        }
    }

/       listando todos los components instalados
    $this->load->library('parser');
    $parsed_view = $this->parser->parse(
        'x_components/components_list',
        array(
            'modules'=>$this->installer->get_installed_components()
        ),
        TRUE
    );
    $this->template->render_string($parsed_view);
}
```

Figura 4.3: Código del algoritmo admin.

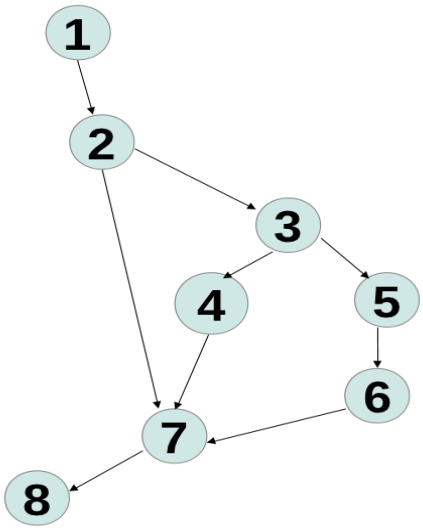


Figura 4.4: Diagrama de flujo asociado al algoritmo admin.

Fórmulas para calcular complejidad ciclomática (para cada formula V(G) representa el valor del cálculo):

1- V(G) = (A – N) + 2

V(G) = (9 - 8) + 2

V(G) = 3

A: cantidad total de aristas.

N: cantidad total de nodos.

2- V(G) = P + 1

V(G) = 2 + 1

V(G) = 3

P: cantidad total de nodos predicados (de los cuales parten dos o más aristas).

3- V(G) = R

V(G) = 3

R: cantidad total de regiones.

La complejidad ciclomática del código es de 3, por lo tanto existen a lo sumo tres posibles caminos por donde el flujo puede circular. Este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

Camino 1	1, 2, 7, 8
Camino 2	1, 2, 3, 4, 7, 8
Camino 3	1, 2, 3, 5, 6, 7, 8

Casos de prueba 1

Camino 1: [1,2,7,8]

Descripción: se selecciona el componente que se va a instalar.

Entrada: fichero de tipo 'xmp_file'(componente).

Resultados esperados: si el fichero no existe se muestran los datos de los componentes instalados.

Casos de prueba 2

Camino 1: [1,2,3,4,7,8]

Descripción: se selecciona el componente que se va a instalar.

Entrada: fichero de tipo 'xmp_file'(componente).

Resultados esperados: instala el componente y se muestran los datos de los componentes instalados.

Casos de prueba 3

Camino 1: [1,2,3,5,6,7,8]

Descripción: se selecciona el componente que se va a instalar.

Entrada: fichero de tipo ’xmp_file’(componente).

Resultados esperados: mensaje de error y se muestran los datos de los componentes instalados.

Las pruebas de caja blanca realizadas al módulo permitieron comprobar que el código está correctamente estructurado y no posee ciclos infinitos contribuyendo a la reducción de errores internos.

4.3.2. Prueba de caja negra

La prueba de caja negra se centra principalmente en los requisitos funcionales del software intentando encontrar errores de las siguientes categorías: funciones incorrectas o ausentes, errores de interfaz, en estructuras de datos o en acceso a bases de datos externas, errores de rendimiento, de inicialización y de terminación [9].

Al software implementado se le realizaron pruebas de caja negra utilizando la técnica de partición de equivalencia obteniéndose los valores válidos y no válidos de las entradas existentes en el software.

4.3.2.1. Casos de prueba de caja negra

CU Instalar componente.

Descripción de la funcionalidad: el caso de uso inicia cuando el administrador desea instalar un componente e introduce la dirección física del mismo en el sistema y termina cuando se ejecuta la acción.

Entrada	**Resultado**	**Condiciones**
El administrador introduce la dirección del componente a instalar. media/Tesis/component/c_table.xmp	Se instala el componente y se muestran los datos asociados al mismo.	
Continúa en la próxima página		

El administrador introduce la dirección del componente a instalar. media/Tesis/component/c_table.xmp Fallo en el sistema.	Se muestra un mensaje de error: “No se ha podido desplegar el módulo. Fallo al crear el directorio temporal”.	
El administrador introduce la dirección de un fichero que no corresponde a un componente. media/Tesis/excriba/fichero.zip	Se muestra un mensaje de error: “El nombre del fichero no es válido para un módulo xmp del eXcriba”.	
El administrador introduce una url incorrecta. media/component/Tesis/c_table.xmp	Se muestra un mensaje de error: “La url del fichero no es válida”.	
El administrador introduce un fichero que excede el peso permitido. media/Tesis/component/c.zip	Se muestra un mensaje de error: “Verifique el tamaño del fichero”.	

CU Visualizar manual de uso de un componente.

Descripción de la funcionalidad: el caso de uso inicia cuando el usuario desea conocer el funcionamiento del componente instalado y termina cuando se ejecuta la acción.

Entrada	**Resultado**	**Condiciones**
Continúa en la próxima página		

El usuario selecciona el identificador del componente que desea consultar. module.id cu.uci.excriba.components.basic_table	Se muestra el manual de uso y desarrollo del componente seleccionado.	El componente debe estar instalado en el sistema.

CU Buscar elementos en la tabla.

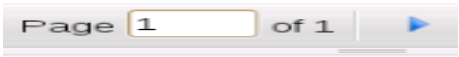
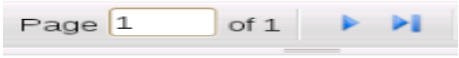
Descripción de la funcionalidad: el caso de uso inicia cuando el usuario desea buscar un elemento y termina cuando se ejecuta la acción.

Entrada	**Resultado**	**Condiciones**
El usuario selecciona el criterio de búsqueda e introduce el valor. Entrada: alberto Find Nombre	Se muestran los datos asociados al nombre entrado por parámetro.	El componente debe estar instalado en el sistema. La tabla debe contener elementos.
El usuario selecciona el criterio de búsqueda e introduce el valor. Entrada: 1254ki Find Nombre	Se muestra un mensaje de error: "Parámetro no válido".	El componente debe estar instalado en el sistema. La tabla debe contener elementos.
Continúa en la próxima página		

El usuario selecciona el criterio de búsqueda e introduce el valor. Entrada: ” ” 	Se muestra un mensaje de error: “Campo vacío“.	El componente debe estar instalado en el sistema. La tabla debe contener elementos.

CU Gestionar página de resultado.

Descripción de la funcionalidad: el caso de uso inicia cuando el usuario desea cambiar de página y termina cuando se ejecuta la acción.

Entrada	**Resultado**	**Condiciones**
El usuario selecciona la opción siguiente página. 	Se muestra la página siguiente de la tabla.	
El usuario selecciona la opción página anterior. 	Se muestra la página anterior de la tabla.	
El usuario selecciona la opción última página. 	Se muestra la última página de la tabla.	
El usuario selecciona la opción primera página. 	Se muestra la primera página de la tabla.	
Continúa en la próxima página		

El usuario modifica la cantidad de elementos a mostrar en la tabla. 	Se muestra en la página la cantidad de elementos seleccionados.	Los elementos de la tabla deben ser mayor o igual al número seleccionado.

CU Ordenar columnas de la tabla.

Descripción de la funcionalidad: el caso de uso inicia cuando el usuario desea ordenar las columnas de la tabla y termina cuando se muestran los datos.

Entrada	**Resultado**	**Condiciones**
El usuario selecciona la columna y la mueve hacia otra posición.	Se muestra la tabla actualizada.	El componente debe estar instalado en el sistema.

CU Actualizar la tabla.

Descripción de la funcionalidad: el caso de uso inicia cuando el usuario desea actualizar los datos de la tabla y termina cuando se ejecuta la acción.

Entrada	**Resultado**	**Condiciones**
El usuario selecciona la opción recargar página.	Se muestra la tabla actualizada.	El componente debe estar instalado en el sistema.

CU Ocultar columnas de la tabla.

Descripción de la funcionalidad: el caso de uso inicia cuando el usuario desea ocultar una columna y termina cuando se ejecuta la acción.

Entrada	**Resultado**	**Condiciones**
El usuario selecciona la columna que desea ocultar o mostrar.	Se realiza la acción seleccionada Se muestra la tabla actualizada.	El componente debe estar instalado en el sistema.

Al módulo se le realizaron 3 iteraciones de pruebas de caja negra (figura 4.5) detectándose en la primera 4 no conformidades, de ellas 2 significativas. En la segunda se habían resuelto 3 de los errores anteriores y se detectaron otros 3 no significativas. En la tercera iteración fueron corregidas todas las no conformidades.

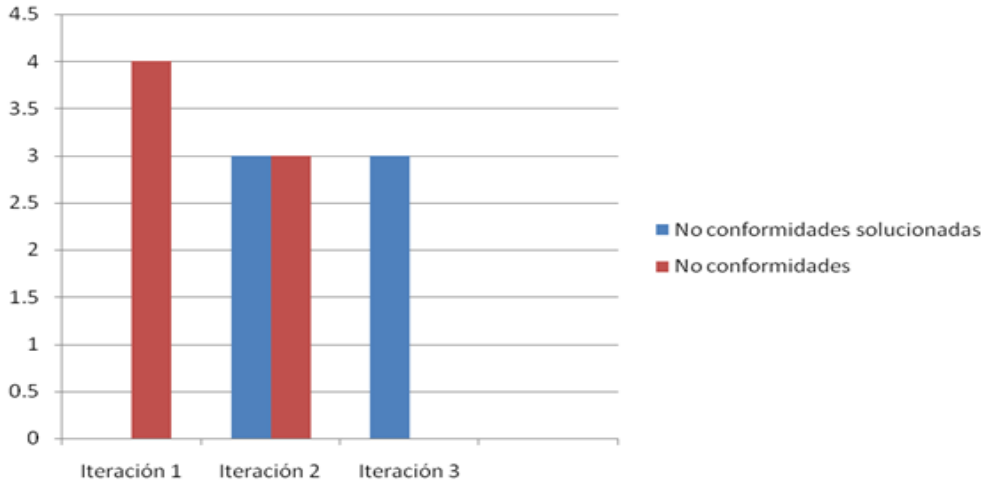


Figura 4.5: Prueba de caja negra

Las pruebas de caja negra permitieron verificar que el comportamiento observado se apega a las especificaciones del producto y a las expectativas del usuario concluyendo que el módulo posee la calidad requerida.

Luego de haber transitado por el flujo de implementación y prueba el módulo biblioteca de componentes quedó desarrollado. En los diagramas generados pudo ilustrarse la relación entre lo principales componentes del software mientras que las pruebas realizadas arrojaron algunos errores que fueron corregidos y permitieron aumentar la calidad final de la solución.

Conclusiones

Durante el desarrollo del trabajo de diploma se expuso la necesidad de implementar una biblioteca siguiendo un desarrollo basado en componentes, arribando a las siguientes conclusiones:

1. El paradigma de componentes es una solución eficiente y adaptable a las necesidades del mercado que permite construir aplicaciones a partir de componentes comerciales o gratuitos ya existentes.
2. El diseño del módulo biblioteca de componentes permitió establecer un punto de partida para la actividades de implementación y facilitó la descomposición en partes de las funcionalidades a desarrollar.
3. La implementación del módulo permitió obtener la infraestructura necesaria para crear nuevos componentes.
4. La utilización de las técnicas partición de equivalencia y camino básico permitieron validar el correcto funcionamiento del módulo.

Recomendaciones

- Implementar y documentar los restantes componentes identificados.
- Actualmente el alcance que tienen las instancias de los componentes es de sessiones. Se recomienda las instancias de *request* y las de aplicación.
- Desarrollar la gestión de componentes por peticiones AJAX.

Referencias bibliográficas

[1] FONSECA, Misael, [et al] *eXcriba, gestor de documentos administrativos*. En II Taller de Sistemas de Gestión de la Información y el Conocimiento. UCIENCIA 2012. Universidad de las Ciencias Informáticas, 20-22 febrero 2012. ISBN: 987-959-286-019-3.

[2] MONTILVA, Jonas, ARAPÉ, Nelson y COLMENARES, Juan. *Desarrollo de Software Basado en Componentes. En: Actas del IV Congreso de Automatización y Control*, Mérida, nov 2003. 9p.

[3] ARIZA, Rosa, Maribel, MOLINA, Juan C. *Jerarquía y Granularidad de componentes de software para Pymes en Bogotá*. Proyecto de Investigación, Bogotá D.C.: Pontificia Universidad Javeriana, junio 2005. 75p.

[4] DÍAZ, Redondo, Rebeca. *Reutilización de requisitos funcionales de sistemas distribuidos utilizando técnicas de descripción formal*. Tesis Doctoral, Vigo, España: Universidade de Vigo, febrero 2002. 284p.

[5] LÓPEZ, Martinez, Patricia. *Desarrollo de sistemas de tiempo real basados en componentes utilizando modelos de comportamiento reactivos*. Tesis Doctoral, Santander, España: Universidad de Cantabria, junio 2010. 316p.

[6] GARRIDO, Márquez, Daniel. *Componentes Software para Sistemas Distribuidos de Tiempo Real*. Tesis Doctoral, España: Universidad de Málaga, diciembre 2005. 265p.

[7] MERELO, Juan Julián. Página personal. [Consultado: abril 2012] Disponible en: http://geneura.ugr.es/~jmerelo/DegaX/activex.html.

[8] ARIZA, Rosa, Maribel, MOLINA, Juan C. *Introducción y principios básicos del desarrollo de software basado en componentes*. [en línea] septiembre 2004. 12p [Consultado: febrero 2012] Disponible en: http://www.ecured.cu/.

[9] PRESSMAN, Roger S. *Ingeniería del Software Un enfoque práctico*. 5ta. ed. La Habana: Félix Varela, 2005. 601 p. ISBN: 970-105-473-3.

[10] GÓMEZ, Omar. *Especificación e implementación de software basado en componentes con la tecnología EJB 3.0*, [en línea] enero 2008. 33p [Consultado: febrero 2012] Disponible en: www.osgg.net/omarsite/resources/papers/cbse_ejb.pdf.

[11] FUENTES, Lidia, TROYA, JM, VALLECILLO, Antonio. *Desarrollo de Software Basado en Componentes*, [en línea] [Consultado: marzo 2012] Disponible en: www.lcc.uma.es/~av/Docencia/Doctorado/tema1.pdf.

[12] GARCÍA, Miguel, Daniel. *Aplicaciones Web para el desarrollo sostenible a través de ONG*. Tesis Doctoral. Departamento de Arquitectura de Computadoras. 2008. 238p.

[13] CARUANA, David [et al]. *Professional Alfresco: Practical Solutions for Enterprise Content Management*. [en línea]. 576p. Wiley Publishing, 2010. [Consultado: mayo 2012] Disponible en: http://www.ebook3000.com/ ISBN 0470571047.

[14] Alfresco.org. [en línea]. 2012, [consultado: junio 2012]. Disponible en: http://wiki.alfresco.com/wiki/Component_Library.

[15] DE_ÁVILA, Sancho. Libro Blanco Gestión Documental Open Source. 2008 [en línea]. Barcelona. [Consultado: mayo 2012]. Disponible en: http://www.smile-iberia.com.

[16] OpenKM Document Management System. [en línea].2012,[citado mayo 2012]. Disponible en: http://www.openkm.com/es.

[17] JACOBSON, Ivar, BOOCH, Grady y RUMBAUGH, James. *El Proceso Unificado de Desarrollo de Software*. La Habana: Félix Varela, 2004. ISBN: 84-7829-036-2.

[18] CHISSIS, Mary, KONRAD, Mike, SHRUM, Sandy. *CMMI for Development. Guidelines for Process Integration and Product Improvement*. 3a ed. [en línea]. Addison-Wesley, 2011. [Consultado: marzo 2012] Disponible en: calisoft.uci.cu/attachments/026_Addison.Wesley.CMMI.for.Development.3rd.Edition.Mar.2011.pdf.

[19] MÚNERA, Danny [et al]. *UML2SC: Herramienta para el diseño de sistemas electrónicos complejos utilizando los lenguajes UML y SystemC*, Revista Facultad de Ingeniería Universidad de Antioquia [en línea]. junio, 2009. [Consultado: febrero 2012] Disponible en: http://www.scielo.org.co/scielo.php?pid=S0120-62302009000200016&script=sci_arttext issn: 0120-6230.

[20] Masadelante.com. 1999 [Consultado: marzo 2012] Disponible en: http://www.masadelante.com/faqs/html.

[21] MATEU, Carles. *Desarrollo de aplicaciones web*. [en línea]. 1a ed. 377p. Barcelona: Eureca Media, SL 2004, [Consultado:junio 2012] Disponible en: http://books.openlibra.com/pdf/desarrollo_aplicaciones_web.pdf ISBN 84-9788-118-4.

[22] Javascripts Comunidad Astalaweb. Gabriel Chova. 2003 [Consultado: marzo 2012] Disponible en: http://javascripts.astalaweb.com.

[23] Scribd [en línea] 2012 [Consultado: febrero 2012] Disponible en: http://es.scribd.com/doc/3062020/Capitulo-I-HERRAMIENTAS-CASE.

[24] Zend Studio. The PHP IDE for professionals. [en línea]. 2012, [consultado: febrero 2012]. Disponible en: http://www.zend.com/en/products/studio/?src=hpbt.

[25] The jQuery Foundation, JavaScript library. [en línea]. 2012, [consultado: febrero 2012]. Disponible en: http://jquery.com/.

[26] YEPES, Jesus. *Introducción a CodeIgniter Blog and Web*. julio 2008 [Consultado: Marzo 2012] Disponible en: http://blogandweb.com/php/introduccion-a-code-igniter-i/.

[27] DÍAZ, Padrón, Enrique. *Módulo de gestión de roles y entidades para el gestor de documentos administrativos excriba*. Tesis, La Habana: Universidad de las Ciencias Informáticas, junio 2011. 148p.

[28] SOMMERVILLE, Ian. *Ingeniería de Software*. La Habana: Félix Varela, enero 2005. 687 p. ISBN: 84-7829-074-5.

[29] LARMAN, Craig. *UML y Patrones Introducción al análisis y diseño orientado a objetos*. La Habana: Félix Varela, 2004. 492 p. ISBN: 842-053-438-2.

Bibliografía

- Alfresco.org. [en línea]. 2012, [Consultado: junio 2012]. Disponible en: http://wiki.alfresco.com/wiki/Component_Library.
- ARIZA, Rosa, Maribel, MOLINA, Juan C. *Jerarquía y Granularidad de componentes de software para Pymes en Bogotá*. Proyecto de Investigación, Bogotá D.C.: Pontificia Universidad Javeriana, junio 2005. 75p.
- ARIZA, Rosa, Maribel, MOLINA, Juan C. *Introducción y principios básicos del desarrollo de software basado en componentes*. [en línea] septiembre 2004. 12p Disponible en: http://www.ecured.cu/index.php?title=Especial93N+Y+PRINCIPIOS+B+Maribel+Ariza+Rojas
- CARUANA, David [et al]. *Professional Alfresco: Practical Solutions for Enterprise Content Management*. [en línea]. 576p. Wiley Publishing, 2010. [Consultado: mayo 2012] Disponible en: http://www.ebook3000.com/ISBN0470571047.
- CERNUDA, Agustín. *Sistema de verificación de componentes software.* Tesis Doctoral. Oviedo, Febrero 2002.
- CHISSIS, Mary, KONRAD, Mike, SHRUM, Sandy. *CMMI for Development. Guidelines for Process Integration and Product Improvement*. 3a ed. [en línea]. Addison-Wesley, 2011. [Consultado: marzo 2012] Disponible en: calisoft.uci.cu/attachments/026_Addison.Wesley.CMMI.for.Development.3rd.Edition.Mar.2011.pdf.

- DE_ÁVILA, Sancho. *Libro Blanco Gestión Documental Open Source.* 2008 [en línea]. Barcelona. [Consultado: mayo 2012]. Disponible en: http://www.smile-iberia.com.
- DÍAZ, Padrón, Enrique. *Módulo de gestión de roles y entidades para el gestor de documentos administrativos excriba*. Tesis, La Habana: Universidad de las Ciencias Informáticas, junio 2011. 148p.
- DÍAZ, Redondo, Rebeca. *Reutilización de requisitos funcionales de sistemas distribuidos utilizando técnicas de descripción formal*. Tesis Doctoral, Vigo, España: Universidade de Vigo, febrero 2002. 284p.
- FONSECA, Misael, [et al] *eXcriba, gestor de documentos administrativos*. En II Taller de Sistemas de Gestión de la Información y el Conocimiento. UCIENCIA 2012. Universidad de las Ciencias Informáticas, 20-22 febrero 2012. ISBN: 987-959-286-019-3.
- FUENTES, Lidia, TROYA JM, VALLECILLO, Antonio. *Desarrollo de Software Basado en Componentes*, [en línea] [Consultado: marzo 2012] Disponible en: www.lcc.uma.es/av/Docencia/Doctorado/tema1.pdf.
- GARCÍA, Miguel, Daniel. *Aplicaciones Web para el desarrollo sostenible a través de ONG.* Tesis Doctoral. Departamento de Arquitectura de Computadoras. 2008. 238p.
- GARRIDO, Márquez, Daniel. *Componentes Software para Sistemas Distribuidos de Tiempo Real*. Tesis Doctoral, España: Universidad de Málaga, diciembre 2005. 265p.
- GÓMEZ, Omar. *Especificación e implementación de software basado en componentes con la tecnología EJB 3.0*, [en línea] Enero 2008. 33p [Consultado: febrero 2012] Disponible en: www.osgg.net/omarsite/resources/papers/cbse_ejb.pdf.
- HASSAN-MONTERO, Yusuff. *Visualización de información persona-ordenador: Propuesta algorítmica para la ordenación espacial de grafos.* Tesis Doctoral. Granada, 2010. 395P
- IRIBARNE, Luis F. *Un Modelo de Mediación para el Desarrollo de Software basado en Componentes COTS.* Tesis Doctoral. Almería, Julio de 2003. 274p

- JACOBSON, Ivar, BOOCH, Grady y RUMBAUGH, James. *El Proceso Unificado de Desarrollo de Software*. La Habana: Félix Varela, 2004. ISBN: 84-7829-036-2.
- Javascripts Comunidad Astalaweb. Gabriel Chova. 2003 [Consultado: marzo 2012] Disponible en: http://http://javascripts.astalaweb.com.
- JIMÉNEZ, López, Rafael. *Análisis y Diseño Orientado a Objetos de un Framework para el Modelado Estadístico con MLG.* Tesis doctoral. Palma de Mallorca, 2003. 241p.
- LARMAN, Craig. *UML y Patrones Introducción al análisis y diseño orientado a objetos.* La Habana: Félix Varela, 2004. 492 p. ISBN: 842-053-438-2.
- LÓPEZ, Martinez, Patricia. *Desarrollo de sistemas de tiempo real basados en componentes utilizando modelos de comportamiento reactivos.* Tesis Doctoral, Santander, España: Universidad de Cantabria, junio 2010. 316p.
- Masadelante.com. 1999 [Consultado: marzo 2012] Disponible en http://http://www.masadelante.com/faqs/html.
- MATEU, Carles. *Desarrollo de aplicaciones web*. [en línea]. 1A ed. 377p. Barcelona: Eureca Media, SL 2004, [Consultado: junio 2012] Disponible en: http://books.openlibra.com/pdf/desarrollo_aplicaciones_web.pdfISBN84-9788-118-4.
- MERELO, Juan Julián. Página personal. [Consultado: Abril 2012] Disponible en: http://geneura.ugr.es/jmerelo/DegaX/activex.html.
- MONTILVA, Jonas, ARAPÉ, Nelson y COLMENARES, Juan. *Desarrollo de Software Basado en Componentes*. En: Actas del IV Congreso de Automatización y Control, Mérida, nov 2003. 9p.
- MÚNERA, Danny [et al]. *UML2SC: Herramienta para el diseño de sistemas electrónicos complejos utilizando los lenguajes UML y SystemC*, Revista Facultad de Ingeniería Universidad de Antioquia [en línea]. junio, 2009. [Consultado: febrero 2012] Disponible en: http://www.scielo.org.co/scielo.php?pid=S0120-62302009000200016script=sci_arttext issn: 0120-6230.

- OpenKM Document Management System. [en línea]. 2012, [Consultado mayo 2012]. Disponible en: http://www.openkm.com/es.
- PRESSMAN, Roger S. *Ingeniería del Software Un enfoque práctico*. 5ta. ed. La Habana: Félix Varela, 2005. 601 p. ISBN: 970-105-473-3.
- RODRÍGUEZ, Juan, GONZÁLEZ Sebastián. *Líneas de Productos de Software Investigación y Aplicación de un nuevo Paradigma de Desarrollo*. Tesis Universidad de la República. Uruguay, Junio de 2007.
- Scribd [en línea] [Consultado: febrero 2012] Disponible en: http://es.scribd.com/doc/3062020/Capitulo-I-HERRAMIENTAS-CASE.
- SOMMERVILLE, Ian. *Ingeniería de Software.* La Habana: Félix Varela, enero 2005. 687 p. ISBN: 84-7829-074-5.
- The jQuery Foundation, JavaScript library. [en línea]. 2012, [consultado: febrero 2012]. Disponible en: http://jquery.com/.
- YEPES, Jesus. *Introducción a CodeIgniter Blog and Web*, julio 2008 [Consultado: marzo 2012] Disponible en: http://blogandweb.com/php/introduccion-a-code-igniter-i/.
- Zend Studio. The PHP IDE for professionals. [en línea]. 2012, [consultado: febrero 2012]. Disponible en: http://www.zend.com/en/products/studio/?src=hpbt.

Anexo A

Primer apéndice

A.1. Figuras

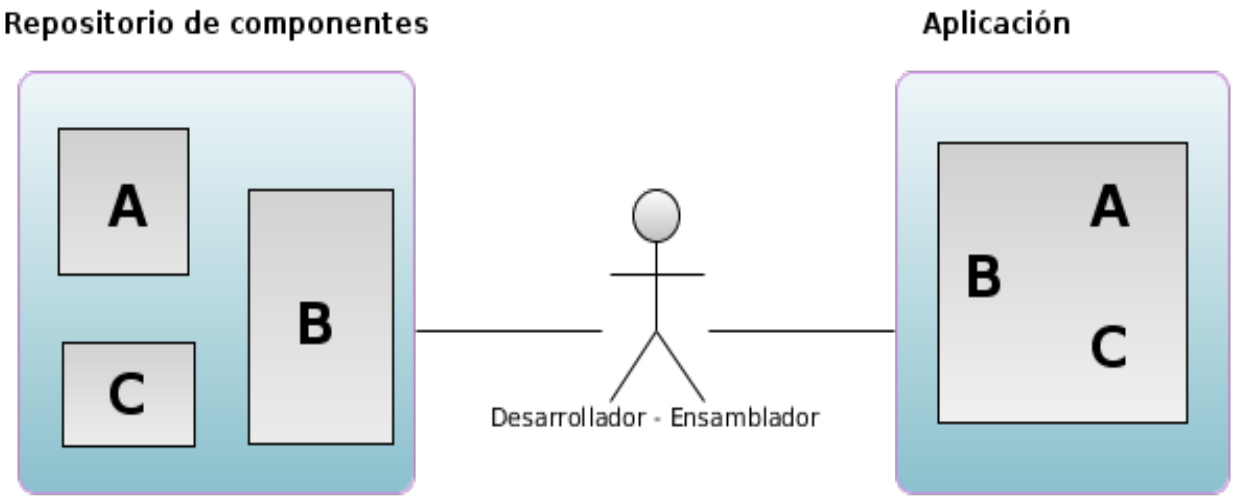


Figura A.1: Desarrollo basado en componente

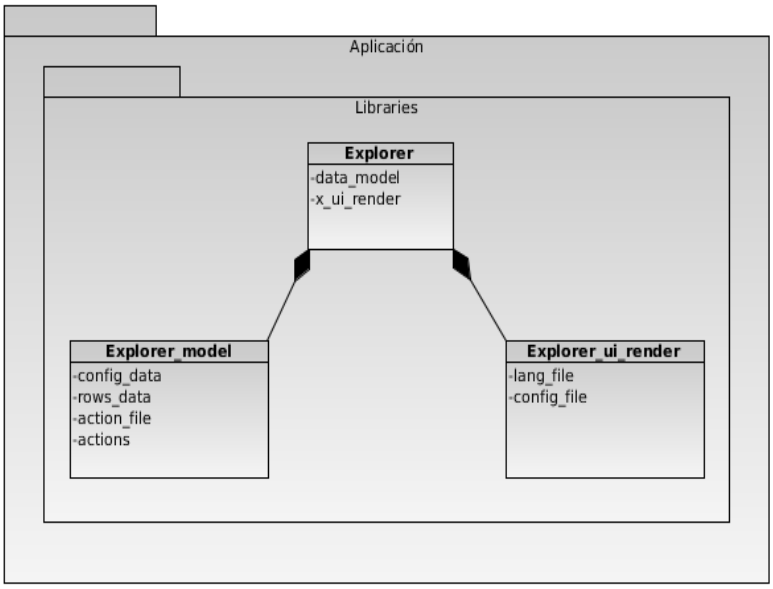


Figura A.2: Diagrama de clases del componente Explorer

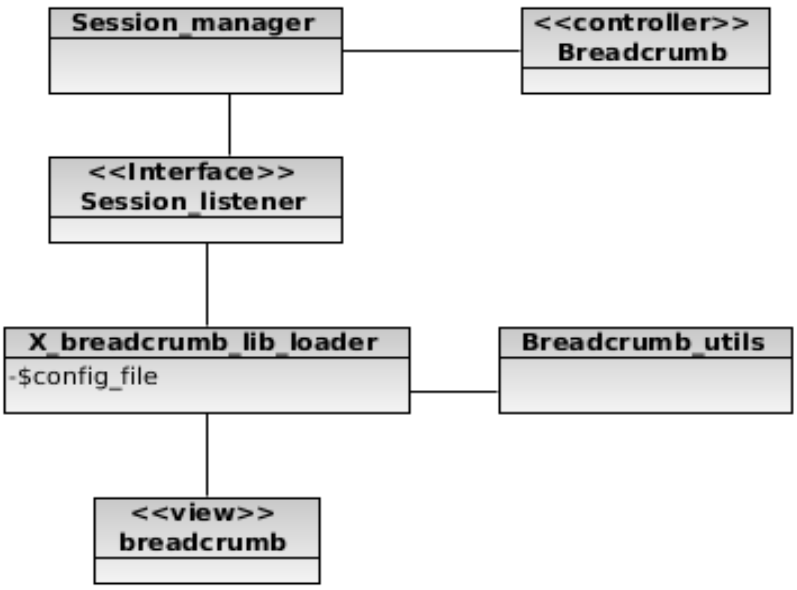


Figura A.3: Diagrama de clases del componente breadcrumb

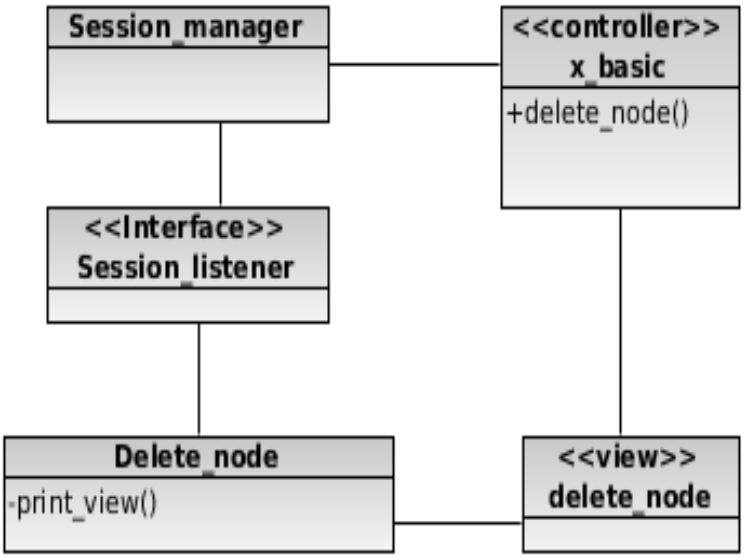


Figura A.4: Diagrama de clases de una ventana emerjente

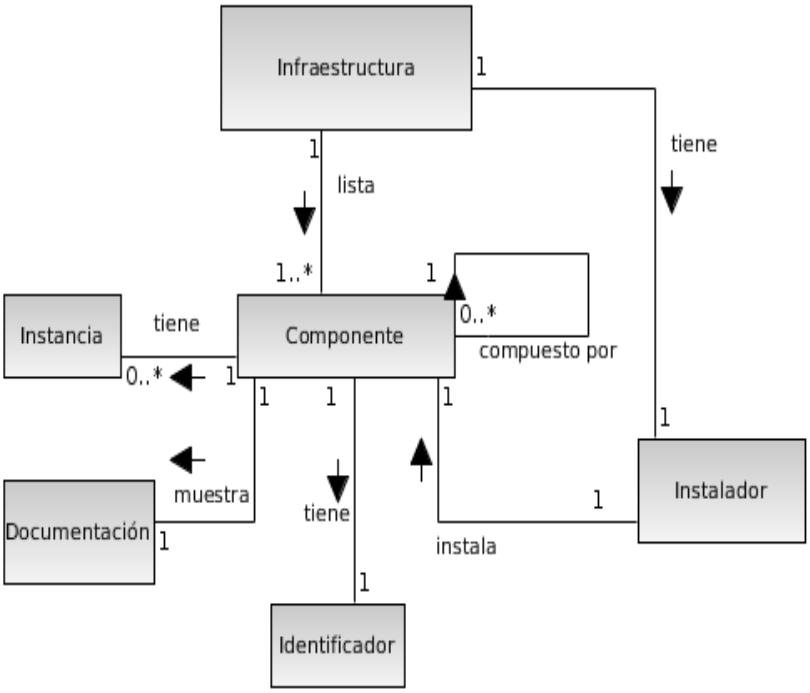


Figura A.5: Modelo de dominio

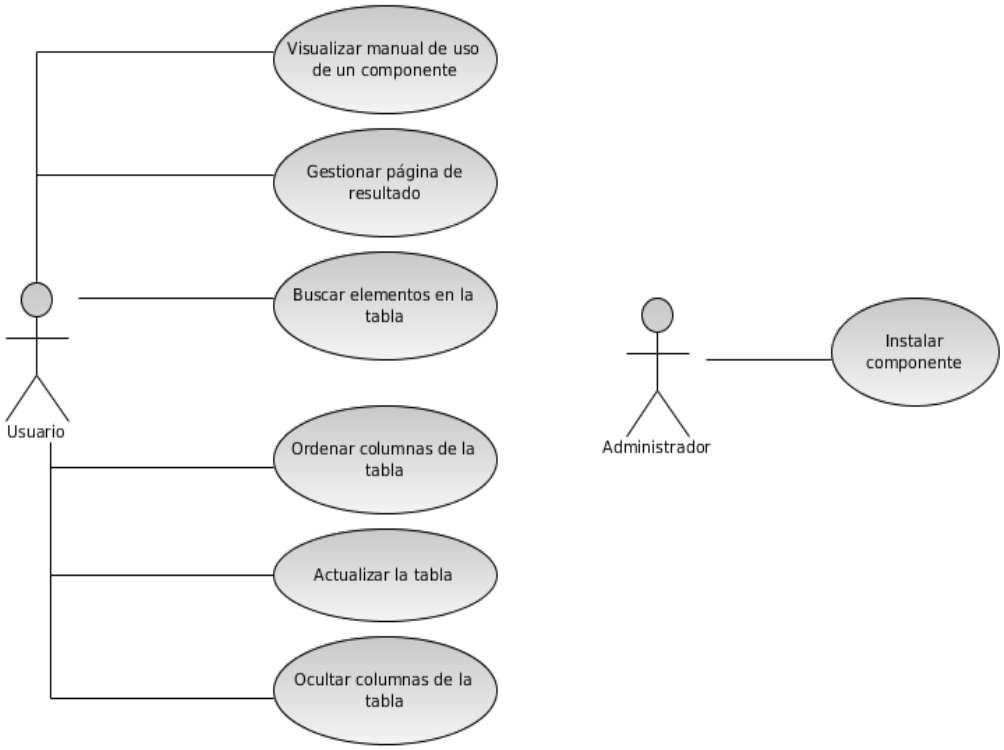


Figura A.6: Diagrama de casos de uso del sistema

Anexo B

Segundo apéndice

B.1. Descripción de casos de uso

Caso de uso	**Instalar componente**
Actor	Administrador
Resumen	El caso de uso inicia cuando el administrador introduce la dirección del componente y termina cuando se ejecuta la instalación.
Prioridad	Crítico
Complejidad	Alta
Referencias	RF1.1, RF1.2
Precondiciones	
Poscondiciones	
Flujo de eventos	
Flujo básico: Instalar componente.	
Actor	**Sistema**
1.Introduce la dirección del componente en el ordenador	
	2. Muestra la interfaz para instalar el componente
3. Selecciona el componente a instalar	
4. Selecciona la opción aceptar	
5. Selecciona la opción instalar	
Continúa en la próxima página	

	6. Instala el componente
	7. Muestra las propiedades del componente instalado Termina el caso de uso
Flujos alternos	
4.a. Selecciona la opción cancelar	
	5.a. Termina el caso de uso

Tabla B.1: Descripción textual del CU: Instalar componente.

Figura B.1: Prototipo de interfaz del CU Instalar componente

Caso de uso	**Visualizar manual de uso de un componente.**
Actor	Usuario
Resumen	El caso de uso inicia cuando el usuario desea conocer el funcionamiento del componente y termina cuando se ejecuta la acción.
Prioridad	Media
Complejidad	Baja
Referencias	RF1.1, RF1.2, RF1.3
Precondiciones	El componente debe estar instalado en el sistema.
Poscondiciones	
Flujo de eventos	
Flujo básico: Visualizar manual de uso de un componente.	
Actor	**Sistema**
1. Accede a la interfaz que muestra los datos de cada componente instalado	
2. Selecciona el identificador del componente que se desea consultar	
	3. Muestra la interfaz con la información del componente Termina el caso de uso
Flujos alternos	

Tabla B.2: Descripción textual del CU: Visualizar manual de uso de un componente.

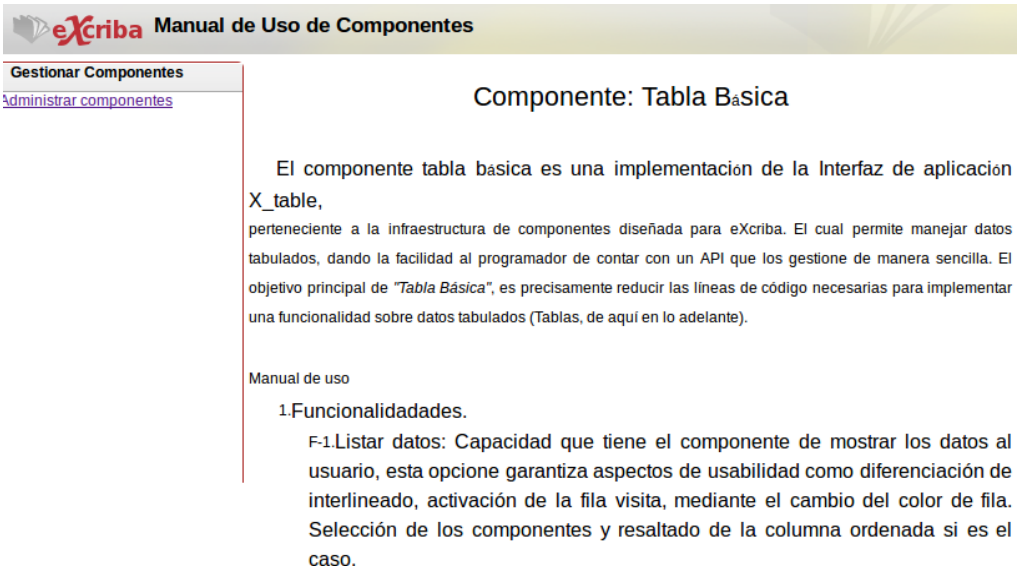


Figura B.2: Prototipo de interfaz del CU Visualizar manual de uso de un componente

Caso de uso	**Buscar elementos en la tabla**
Actor	Usuario
Resumen	El caso de uso inicia cuando el usuario desea buscar un elemento y termina cuando se ejecuta la acción.
Prioridad	Media
Complejidad	Media
Referencias	RF2.1 RF2.2
Precondiciones	
Poscondiciones	
Flujo de eventos	
Flujo básico: Buscar elementos en la tabla	
Continúa en la próxima página	

Actor	Usuario
1. Selecciona la opción de buscar	
	2. Muestra la interfaz para realizar una búsqueda
3. Selecciona el parámetro de búsqueda	
4. Introduce el nombre del elemento a buscar	
	5. Realiza la búsqueda por el parámetro especificado
	6. Muestra el resultado Termina el caso de uso
Flujos alternos	
4.a. Introduce un nombre no válido	
	5.a. Muestra un mensaje de error: “El parámetro no es válido“. Termina el caso de uso

Tabla B.3: Descripción textual del CU: Buscar elementos en la tabla.

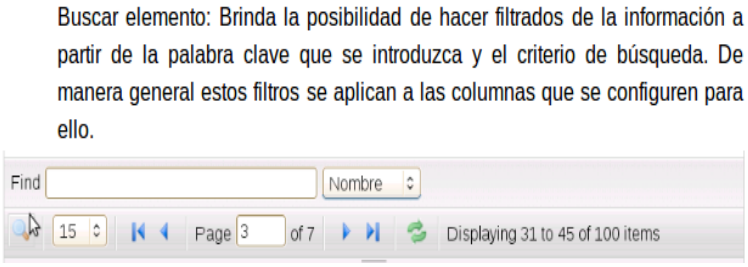


Figura B.3: Prototipo de interfaz del CU Buscar elementos en la tabla

Caso de uso	**Ordenar columnas de la tabla**
Actor	Usuario
Resumen	El caso de uso inicia cuando el usuario desea ordenar las columnas de la tabla y termina cuando se muestran los datos.
Prioridad	Media
Complejidad	Media
Referencias	RF2.1 RF2.4
Precondiciones	
Poscondiciones	
Flujo de eventos	
Flujo básico: Ordenar columnas de la tabla.	
Actor	**Sistema**
1. Selecciona la columna que desea mover	
2. Arrastra la columna hacia el lugar deseado	
	3. Realiza el cambio
	4. Muestra la tabla actualizada Termina el caso de uso

Tabla B.4: Descripción textual del CU: Ordenar columnas de la tabla.

Figura B.4: Prototipo de interfaz del CU Ordenar columnas de la tabla

Caso de uso	**Gestionar página de resultado.**
Actor	Usuario
Resumen	El caso de uso inicia cuando el usuario desea cambiar de página y termina cuando se ejecuta la acción.
Prioridad	Alta
Complejidad	Media
Referencias	RF2.1, RF2.6.1, RF2.6.2, RF2.6.3, RF2.6.4, RF2.6.5, RF2.6.6
Precondiciones	
Poscondiciones	
Flujo de eventos	
Flujo básico: Gestionar página de resultado.	
Actor	**Sistema**
Continúa en la próxima página	

	1. Muestra la interfaz de paginado
2. Ejecuta una de las acciones mostradas	
	3. Muestra las siguientes opciones • Cambiar cantidad de elementos a mostrar. Ver Sección 1: “Cambiar cantidad de elementos a mostrar”. • Ir a la página siguiente. Ver Sección 2: “Ir a la página siguiente”. • Regresar a la página anterior. Ver Sección 3: “Regresar a la página anterior”. • Ir a la última página. Ver Sección 4: “Ir a la última página”. • Regresar a la primera página. Ver Sección 5: “Regresar a la primera página”. • Ir a una página deseada. Ver Sección 6: “Ir a una página deseada”.
Sección 1: “Cambiar cantidad de elementos a mostrar.”	
Flujo básico: Cambiar cantidad de elementos a mostrar	
Actor	**Sistema**
1. Selecciona la opción Cambiar cantidad de elementos a mostrar	
Continúa en la próxima página	

	2. Muestra en la tabla la cantidad de elementos seleccionados
	3. Muestra la cantidad de elementos de esa página y la cantidad total Termina el caso de uso
Sección 2: "Ir a la página siguiente."	
Flujo básico: Ir a la página siguiente.	
Actor	**Sistema**
1. Selecciona la opción Ir a la página siguiente.	
	2. Muestra la página seleccionada
	3. Muestra la cantidad de elementos de esa página y la cantidad total Termina el caso de uso
Sección 3: "Regresar a la página anterior."	
Flujo básico: Regresar a la página anterior.	
Actor	**Sistema**
1. Selecciona la opción Regresar a la página anterior.	
	2. Muestra la página seleccionada
	3. Muestra la cantidad de elementos de esa página y la cantidad total Termina el caso de uso
Sección 4: "Ir a la última página."	
Flujo básico: Ir a la última página.	
Continúa en la próxima página	

Actor	**Sistema**
1. Selecciona la opción Ir a la última página.	
	2. Muestra la página seleccionada
	3. Muestra la cantidad de elementos de esa página y la cantidad total Termina el caso de uso
Sección 5: "Regresar a la primera página."	
Flujo básico: Regresar a la primera página.	
Actor	**Sistema**
1. Selecciona la opción Regresar a la primera página	
	2. Muestra la página seleccionada
	3. Muestra la cantidad de elementos de esa página y la cantidad total Termina el caso de uso
Sección 6: "Ir a una página deseada."	
Flujo básico: Ir a una página deseada.	
Actor	**Sistema**
1. Introduce el número de la página a consultar	
	2. Muestra la página seleccionada
	3. Muestra la cantidad de elementos de esa página y la cantidad total Termina el caso de uso
Flujos alternos	
Continúa en la próxima página	

1.a Introduce un valor mayor a la cantidad de páginas de la tabla	
	1.a.1 Muestra el mensaje de error Termina el caso de uso

Tabla B.5: Descripción textual del CU: Gestionar página de resultado.

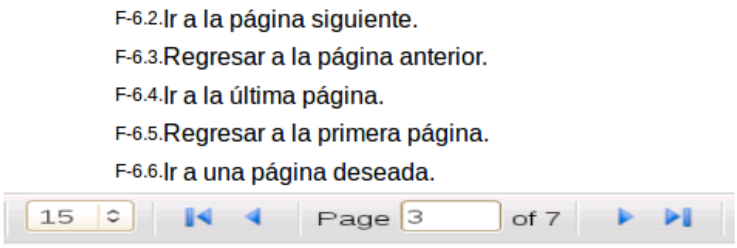


Figura B.5: Pototipo de interfaz del CU Gestionar página de resultado

Caso de uso	**Actualizar la tabla**
Actor	Usuario
Resumen	El caso de uso inicia cuando el usuario desea actualizar los datos y termina cuando se ejecuta la acción.
Prioridad	Alta
Complejidad	Media
Referencias	RF2.1 RF2.3
Precondiciones	
Poscondiciones	
	Continúa en la próxima página

Flujo de eventos	
Flujo básico: Actualizar la tabla.	
Actor	**Sistema**
1. Selecciona la opción actualizar.	
	2. Actualiza los cambios
	3. Muestra la tabla actualizada Termina el caso de uso

Tabla B.6: Descripción textual del CU: Actualizar la tabla.

F-3.Actualizar datos: Actualiza los datos sin recargar la página dónde el cliente visualiza la tabla. Esto facilita sobre todo la consulta de información a la vez que se opere sobre otras regiones de la página, como lo pueden ser formularios, menúes y otros elementos editables.

Figura B.6: CU Actualizar la tabla

Caso de uso	**Ocultar columna de la tabla**
Actor	Usuario
	Continúa en la próxima página

Resumen	El caso de uso inicia cuando el usuario desea ocultar una columna y termina cuando se ejecuta la acción.
Prioridad	Media
Complejidad	Media
Referencias	RF2.1, RF2.5
Precondiciones	
Poscondiciones	
Flujo de eventos	
Flujo básico: Ocultar columnas de la tabla	
Actor	**Sistema**
1. Accede a la interfaz de ocultar columna	
	2. Muestra la interfaz para ocultar una columna de la tabla
3. Selecciona la columna a ocultar	
	4. Ejecuta la acción
	5. Muestra la tabla actualizada Termina el caso de uso

Tabla B.7: Descripción textual del CU: Ocultar columnas de la tabla.

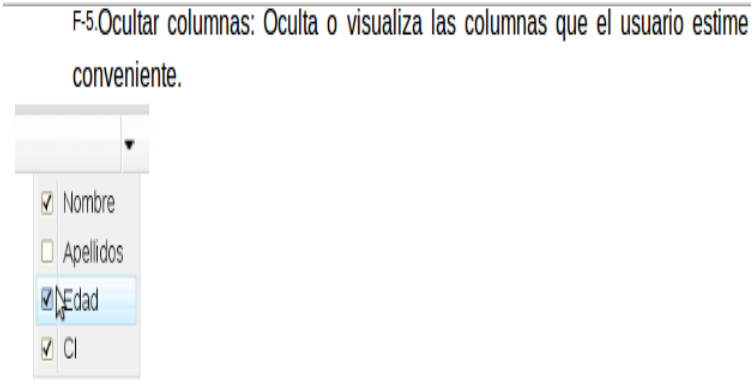


Figura B.7: CU Ocultar columnas de la tabla

Anexo C

Tercer apéndice

C.1. Diagramas de clases del diseño

Todas las funcionalidades de la tabla presentan el diseño de clases que se muestra en la figura C.1. Cada componente seguirá la misma estructura, lo que cambia en cada uno de ellos es la clase que lo implementa y/o la funcionalidad que se esté ejecutando.

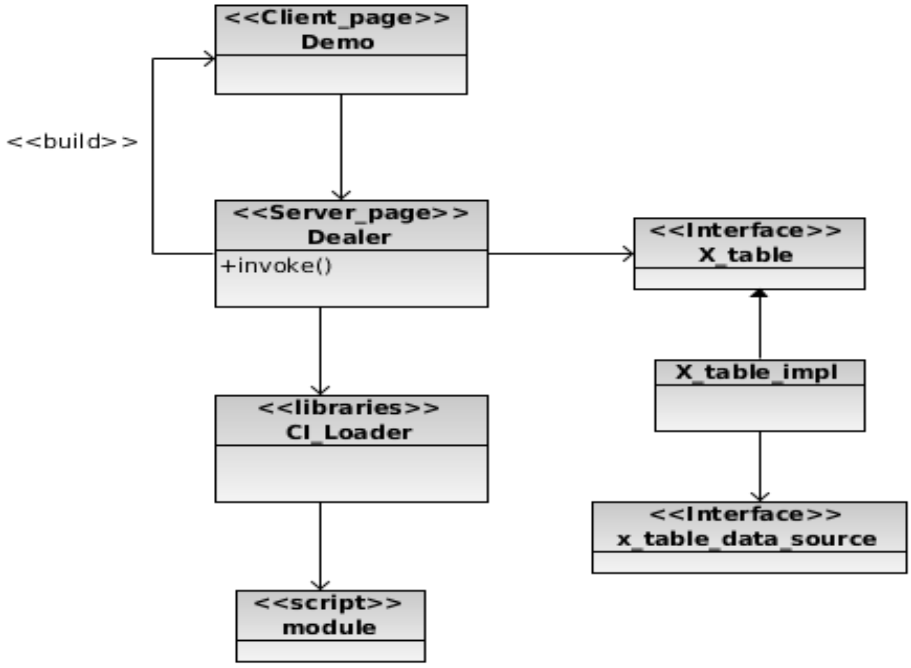


Figura C.1: Diagrama de clases para el componente tabla.

Nombre: Dealer
Tipo de clase: Controladora

Atributo	**Tipo**
$message	
Para cada responsabilidad:	
Nombre:	invoke()
Descripción:	Conecta cualquier instancia de un componente con su implementación.

Anexo D

Cuarto apéndice

D.1. Estándar de codificación

Nomenclatura de clases y métodos

Los nombres de clase deben tener siempre en mayúscula la primera letra y el método constructor debe coincidir de forma idéntica. Varias palabras deben estar separadas por un guión y no usando la función camelCased. Todos los métodos de la clase debe estar completamente en minúsculas y el nombre para indicar claramente su función, preferiblemente incluyendo un verbo. Trate de evitar los nombres excesivamente largos y verbos.

INCORRECTO: class superclass, class SuperClass

CORRECT: class Super_class

CORRECTO: class Super_class function Super_class()

Ejemplos de nombres de métodos incorrectos y correctos:

INCORRECTOS:

fileproperties function () / / No descriptivos y las necesidades de guión de separación.

fileProperties function () / / No descriptivo y utiliza CamelCase la función.

getfileproperties () / / Mejor Pero todavía falta destacar la función de separación.

getFileProperties () / / utiliza la función CamelCase

get_the_file_properties_from_the_file () / / muchas palabras.

CORRECTOS: get_file_properties function () / / descriptivos, ponen de relieve separador y todas las letras minúsculas.

Nombres de variables

Las directrices para la nomenclatura de las variables es muy similar a la utilizada para los métodos de clase. Es decir, las variables deben contener sólo letras minúsculas, use separadores y de manera razonable el nombre para indicar su finalidad y el contenido. Variables con un número muy corto de palabras sólo se debe utilizar como iteradores de los bucles.

INCORRECTO:

$j = "foo/ / variables de una sola letra, solo debe ser utilizado en ciclos.

$ Str / /contiene letras mayúsculas.

$ bufferedText / / usa camelCasing y podría ser reducido sin perder significado semántico,

$ groupid / / varias palabras, necesita guión bajo para separarlas,

$ name_of_last_city_used / / demasiado larga.

CORRECTA: for ($ j = 0; $ j <10, $ j + +) $ strgroup_id $ buffer $ last_city