Universidad de las Ciencias Informáticas

Facultad 3

“Módulo para la generación de interfaces gráficas de usuario para el marco de trabajo Sauxe_v2.2”

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

Autor: Rafael Alejandro Limia Téllez

Tutores: Ing. René R. Bauta Camejo

Ing. Dausbel Torreblanca Pavó

Ing. Inoelkis Velazquez Osorio

“06/2014”

DECLARACIÓN DE AUTORÍA

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

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

Rafael Alejandro Limia Téllez

_____

Firma del Autor

Ing. René Bauta Camejo

_____

Firma del Tutor 1

Ing. Inoelkis Velázquez Osorio

_____

Firma del Tutor 2

Ing. Dausbel Torreblanca Pavó

_____

Firma del Tutor 3

DATOS DE CONTACTO

René R. Bauta Camejo: Ingeniero en Ciencias Informáticas, graduado en 2009 en la Universidad de las Ciencias Informáticas. Instructor. Cuatro años de experiencia en el desarrollo de software. Cuatro años de graduado.

Inoelkis Velazquez Osorio: Ingeniero en Ciencias Informáticas, graduado en 2013 en la Universidad de las Ciencias Informáticas. Tres años de experiencia en el desarrollo de software.

Dausbel Torreblanca Pavó: Graduado de Ingeniero en Ciencias Informáticas en el año 2012.

Dos años de experiencia en el desarrollo de software.

Agradecimientos

Quiero agradecer a mis familiares, sobre todo a mis padres, a mi abuela. A mis amigos que siempre han estado ahí, en las buenas y en las malas, especialmente a Carlos, Yandri y Alejandro, quienes me ayudaron con todos los problemas que surgieron a la hora de darle formato al documento. A los tutores, que hicieron posible la conclusión de la tesis.

A todos los que hicieron posible esto,

GRACIAS

RESUMEN

El presente trabajo tiene como objetivo desarrollar una solución que permita a los programadores del marco de trabajo Sauxe la generación de las interfaces de usuario mediante una herramienta informática que no solo mejorará el tiempo de desarrollo de estas, sino que evitará que se escriba gran parte del código de las mismas. Para ayudar al desarrollo de la solución se realizó un estudio de las herramientas y tecnologías a utilizar a lo largo del desarrollo de la misma. Además se identificaron y validaron los requisitos funcionales con que contaría el módulo, mediante técnicas de captura y validación de requisitos. También se describen los patrones arquitectónicos y de diseño a utilizar para implementar la propuesta de solución con el fin de proveer reutilización al módulo, ahorrar tiempo en la implementación y obtener un producto con mejor calidad. Mediante las pruebas de software se valida que el sistema pueda cumplir con el objetivo que persigue la investigación.

PALABRAS CLAVE

Interfaces de usuario, tiempo de desarrollo

INDICE

Índice de Figuras

Índice de Tablas

Introducción

Durante las dos últimas décadas, el desarrollo tecnológico muestra una convergencia entre la Informática, las Telecomunicaciones, la Electrónica y la Automatización, proceso que ha devenido una nueva rama del saber denominada Tecnologías de la Información y las Comunicaciones (TIC), de alta incidencia en la modernización y eficiencia de todos los sectores de la sociedad(4).

Cuba ha identificado la conveniencia y necesidad de dominar e introducir en la práctica social las TIC lo que facilitaría al país alcanzar un desarrollo sostenible. La Industria cubana del software inmersa en el perfeccionamiento continuo de la informática y las comunicaciones tiene como objetivo informatizar la sociedad, insertar a la isla en el mercado mundial del software y llegar a convertirse en puntera del desarrollo tecnológico, producto de dicho desarrollo surgió la Universidad de las Ciencias Informáticas (UCI). La UCI cuenta con una serie de centros productivos para el desarrollo de software, entre estos se encuentra el Centro de Informatización para la Gestión Entidades (CEIGE) de la facultad 3.

El CEIGE cuenta con un marco de trabajo para desarrollar aplicaciones web de gestión, denominado Sauxe, el cual provee la estructura genérica para el desarrollo de aplicaciones web de gestión, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo.

En el desarrollo de las aplicaciones web, un elemento fundamental son las vistas o interfaces visuales, las mismas permiten la interacción del usuario con el sistema, por lo que su desarrollo es de vital importancia. Para la construcción de estas vistas los programadores se enfrentan muchas veces a un gran problema, puesto que llevar el diseño realizado en papel al sistema web es una tarea tediosa y complicada, debido a que no siempre los componentes toman la alineación requerida, ni el sistema se ve de la misma forma en todos los navegadores web. Para evitar estos problemas los desarrolladores dedican horas e incluso días de trabajo tratando de lograr una alineación perfecta de los componentes, así como la compatibilidad con la mayoría de los navegadores web.

Actualmente los desarrolladores del proyecto Sauxe_v2.2 realizan las interfaces visuales manualmente, lo que genera pérdida de tiempo en el desarrollo del sistema, puesto que estos deben hacer las vistas desde el inicio aunque lo que varíe sea solo un componente.

Por otro lado, los desarrolladores deben estudiar exhaustivamente el lenguaje a usar en la realización de las interfaces, lo que implica que deben invertir tiempo de desarrollo en dicho estudio.

Teniendo en cuenta la situación planteada, se define como **problema a resolver**: ¿Cómo disminuir el tiempo de desarrollo de las interfaces gráficas de usuario en el marco de trabajo Sauxe_v2.2?

Definiéndose como **Objeto de estudio**: Diseño de interfaces gráficas de usuario.

Enmarcado en el **Campo de acción**: Diseño de interfaces gráficas de usuario usando ExtJs.

Como **objetivo general** para resolver el problema planteado anteriormente se propone: Desarrollar un módulo para disminuir el tiempo de desarrollo de las interfaces gráficas de usuario en el marco de trabajo Sauxe_v2.2.

Para dar cumplimiento al objetivo general se trazaron los siguientes **objetivos específicos**:

- Construir el Marco Teórico de la investigación relacionada con la generación de interfaces gráficas de usuario.
- Realizar el análisis y diseño del módulo para la generación de interfaces gráficas de usuario para el marco de trabajo Sauxe_v2.2.
- Implementar el módulo para la generación de interfaces gráficas de usuario para el marco de trabajo Sauxe_v2.2.
- Validar la solución propuesta.

Se tiene como **idea a defender** en la presente investigación:

Con el desarrollo de un módulo que permita la generación de interfaces gráficas para el marco de trabajo Sauxe_v2.2, se disminuirá el tiempo de creación de las mismas.

Para el desarrollo de las tareas científicas se utilizan **Métodos de Investigación** en la búsqueda y procesamiento de la información. Los mismos se dividen en teóricos y empíricos.

Los **Métodos Teóricos** son factibles en el estudio de las características poco observables del objeto de investigación. Dentro de este grupo se utilizan:

- El método **Análisis Histórico-Lógico**, posibilita estudiar de forma analítica la trayectoria histórica real de los fenómenos, su evolución y desarrollo. El método permite realizar la primera parte de la investigación, al hacer un análisis bibliográfico de otros módulos o componentes de generación de interfaces gráficas, el razonamiento basado en caso y técnicas de programación más utilizadas en el desarrollo de este tipo de sistemas. Se utiliza para determinar a través de la evaluación de la bibliografía conceptos de esta temática, que permiten conocer el estado de la evolución actual del fenómeno e identificar posibles mejoras y alternativas de solución.

- El **Analítico-Sintético**, se utiliza para el estudio, a partir de fuentes bibliográficas, de conceptos y técnicas en soluciones previamente desarrolladas. Permite además descomponer el problema de investigación en elementos por separado y profundizar en el estudio de cada uno de ellos, para luego sintetizarlos en la solución propuesta.

Los **métodos Empíricos** tienen gran importancia ya que permiten efectuar el análisis preliminar de la información, así como verificar y comprobar las concepciones teóricas. Dentro de este grupo se utiliza:

- El **método de Observación**: Está planificado y dirigido con el fin de realizar la fundamentación teórica del problema.

- El **método de Entrevista**: Apoyará a la incorporación de conocimientos mediante las entrevistas planificadas efectuadas a los especialistas de áreas de CEIGE.

El documento cuenta con la siguiente estructura:
Capítulo 1: Fundamentos Teóricos. Se describen los aspectos y conceptos asociados al dominio del problema a resolver, siendo estos esenciales para entender el entorno del mismo.

Se presenta el estado del arte de las técnicas implicadas en el objeto de estudio y el conjunto de tecnologías involucradas en el desarrollo de la propuesta.

Capítulo 2: Análisis y diseño. Se exponen los procesos de negocios y requisitos funcionales y no funcionales que debe cumplir el módulo, además de los artefactos generados durante el diseño del mismo.

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

Capítulo 1: Fundamentos Teóricos

En este capítulo se hace referencia a una serie de conceptos que son de vital importancia para la comprensión y desarrollo de la investigación. Se realiza un análisis de las principales características de las herramientas, tecnologías y metodologías utilizadas durante el desarrollo del sistema. Además se hace un estudio de los sistemas similares.

1.1 Conceptos fundamentales

Concepto: Se ha definido de diversas formas por diferentes autores. Por ejemplo:
El Diccionario de la Lengua Española revela que este término tiene varios significados. Las definiciones más cercanas de lo que interesa en este estudio son descritas al decir que la palabra "concepto" es una idea que concibe o forma el entendimiento; un pensamiento expresado con palabras; la opinión o el juicio sobre algo. Luego se refiere a la idea de "formar concepto", lo que es entendido como el determinar algo en la mente después de examinadas las circunstancias.

También existen otras definiciones como la que plantea Houaiss, "concepto" es una idea, o sea, una representación mental de algo que puede ser concreto o abstracto (3).

En nuestra investigación se utilizará como definición de **concepto**: Es una representación mental de un objeto, que se muestra como un instrumento fundamental del pensamiento en su tarea de identificar, describir y clasificar los diferentes elementos y aspectos de la realidad (4).

Interfaz: Este concepto se ha definido ampliamente desde diferentes puntos de vista, según el ámbito de conocimientos en que se aplique. Por ejemplo, en la biología es la capa de un organismo que separa su interior del exterior; en la electrónica y las telecomunicaciones, se ha definido como puerto a través del que se envían o reciben señales desde un sistema de subsistemas hacia otros; y en química es la superficie existente entre dos fases distintas en una mezcla heterogénea. La traducción literal atendiendo a su etimología sería: “superficie vista, o lado mediador” (1).

Interfaz de gráfica de usuario: En términos de informática ha sido definida como un tipo de entorno que representa en la pantalla programas como archivos y opciones por medio de íconos como menús y cuadros de diálogos. El usuario puede seleccionar y activar estas opciones apuntando y haciendo clic con el ratón o, frecuentemente, con el teclado. Un elemento

particular (tal como una barra de desplazamiento) trabaja de igual forma para el usuario en todas las aplicaciones, pues la interfaz gráfica de usuario proporciona rutinas de software estándar; las aplicaciones invocan a estas rutinas con los parámetros específicos en lugar de intentar reproducirlas a partir de la nada (2).

1.2 Creación y edición de interfaces

“Actualmente, en el mundo prima la cultura de la interfaz gráfica fácil de aprender y vistosa, en la que con un simple clic sobre algún componente gráfico que esté presente en pantalla, se sustituye la tediosa tarea de escribir código fuente para que el ordenador interprete que debe realizar alguna acción.”(5) Esta mejora, se ha extrapolado además al escenario de los creadores de software, que si bien han de tener conocimientos suficientes para realizar una interfaz gráfica utilizando líneas de código, también han decidido crear herramientas que agilicen el proceso.

Un editor de interfaces gráficas de usuario, también conocido como constructor de GUI (GUI builder) o diseñador de GUI (GUI designer), es una herramienta de software que simplifica la creación de GUI permitiéndole al diseñador usar un editor WYSIWYG para arrastrar y soltar componentes hasta obtener lo deseado. Sin un Editor GUI, estas deben ser construidas manualmente especificando los parámetros de cada elemento mediante código, sin ninguna retroalimentación hasta que la aplicación no se esté ejecutando (6).

En la investigación correspondiente a la elaboración de un módulo para la generación de interfaces gráficas de usuario para el marco de trabajo Sauxe, se realizó una búsqueda bibliográfica, mediante la cual se encontraron varias herramientas informatizadas para este proceso, aunque todas con sus particularidades. Se analizaron cuatro características a tener en cuenta de cada herramienta: modificaciones en tiempo de ejecución; desarrollo sobre tecnologías libres; integración al marco de trabajo y usabilidad del sistema. Estos aspectos se basan fundamentalmente en las funcionalidades que debe cumplir la aplicación a implementar, aunque el indicador con más valor es la integración con la plataforma existente, pues es primordial lograr con el menor costo el mayor acoplamiento posible al marco de trabajo.

El análisis de estas herramientas tiene como objetivo llegar a conclusiones que sustenten la creación de una herramienta con las características necesarias para satisfacer a los usuarios de productos desarrollados utilizando Sauxe.

1.2.1 Artisteer 4.0

Es un programa para automatizar el diseño de plantillas destinadas a ser visualizadas en páginas Web, que crea al instante una red de gran apariencia, plantillas únicas y entradas de blog. Permite diseñar temas para insertarlos en gestores de contenidos como Drupal, Joomla, Wordpress o Blogger, consiguiendo tener una web completamente personalizada, en la que es posible configurar hasta el más mínimo aspecto. Sus principales características son: (7)

- **Edición de contenido**: se puede diseñar la plantilla y crear páginas con contenido para cualquier tipo de CMS.

- **HTML5 y soporte CSS3**: compatibilidad con la última evolución de los estándares de mejores páginas web estructuradas.

- **Soporte para dispositivos móviles**: genera plantillas escalando el diseño de sus páginas web para dispositivos móviles y portátiles.

- **Diseñador visual del encabezado**: un nuevo diseñador de encabezado que se puede utilizar para agregar funciones que antes no eran posibles como: añadir una presentación de diapositivas a su cabecera o crear collages de fotos, entre otras.

- **Soporte de varios formatos**: ARTX, HTML, JPG, PNG, GIF.

- **Interfaz simple e intuitiva**: es una aplicación fácil de usar.

- Soporte en diferentes idiomas.

La herramienta Artisteer, a pesar de ser una opción satisfactoria para la personalización de interfaces gráficas de usuario, tiene el inconveniente de centrarse en la creación de plantillas para páginas y sitios web que posteriormente pueden ser usadas en un CMS. Por otro lado esta aplicación supone un problema en la integración al tratarse de una aplicación de escritorio.

1.2.2 Ext Designer

Ext Designer es un constructor de interfaz gráfica de usuario para las aplicaciones web con Ext JS. Esta aplicación de escritorio facilita la creación de las GUI de manera muy rápida y sencilla, permitiendo dedicar la mayor parte del tiempo a la programación y no tanto al diseño. Ext Designer, proporciona un fácil uso de su entorno Drag and Drop, con el que en pocos minutos y conocimientos mínimos, se pueden realizar el prototipado rápido de componentes de interfaz de la aplicación, la conexión de componentes de interfaz de datos y exportar de forma correcta el código orientado a objetos para cada componente. El propio entorno Drag and drop da la posibilidad de acomodar los componentes a gusto con el mouse. Además proporciona gran cantidad de información técnica en el sitio oficial del producto para evacuar las dudas e incluso videos y tutoriales aclarativos con buenos ejemplos. Sus características más sobresalientes son: (8)

- **Cambio entre la vista diseño/código**: no será necesario hacer todo el diseño mediante código, con la existencia de la vista previa.

- **La lista de todos los componentes a la mano**: una paleta de componentes definidos que permite seleccionarlos o arrastrarlos hacia el área de trabajo que se esté creando.

- Exportar el proyecto a los archivos fuentes correspondientes.

- **Duplicación de Componentes**: le proporciona la capacidad de duplicar conjuntos de componentes y luego modificar sus valores sin arrastrar y soltar los componentes para volver a crear configuraciones comunes una y otra vez.

- **Transformación de Componentes**: el diseñador realizará automáticamente las transformaciones necesarias para convertir un componente en otro según se necesite.

- **Deshacer / Rehacer**: Permite deshacer un error o rehacer el último cambio que se hizo a través de botones en la barra de herramientas.

Su principal desventaja radica en que esta aplicación no es gratuita, y para utilizarlo es necesario pagar una licencia de software. Además se especifica que es una aplicación de escritorio y por lo tanto no es integrable al marco de trabajo.

1.2.3 Qt Designer

Es una herramienta para el diseño y la creación de interfaces gráficas de usuario para los componentes de Qt. Esta herramienta pertenece al entorno de desarrollo integrado Qt Creator, que proporciona apoyo a la creación y funcionamiento de aplicaciones de Qt para entornos de escritorio (Windows, Linux, FreeBSD y Mac OS) y dispositivos móviles (9).

Con Qt Designer es posible componer y personalizar los widgets o cuadros de diálogo en un entorno WYSIWYG, además se pueden probar estos componentes con diferentes estilos y resoluciones directamente en el editor (9).

Los reproductores, formas y componentes creados con Qt Designer se integran perfectamente con el código programado, utilizando las señales y el mecanismo de las franjas horarias de Qt. Cuenta además con el Diseñador Qt Quick, que es una herramienta para el desarrollo de animaciones utilizando un lenguaje de programación declarativa QML. Todas las propiedades establecidas en Qt Designer se pueden cambiar de forma dinámica dentro del código. Además, características como la promoción widget y plugins personalizados permiten utilizar componentes propios en Qt Designer (9).

El inconveniente de esta solución es su tecnología, enfocado a componentes del lenguaje Qt y apoyo en la creación de aplicaciones de escritorio. En cambio la investigación está encaminada a la búsqueda de una solución para los componentes de Ext JS en aplicaciones sobre la web.

1.2.4 Lycan-Génesis

Lycan-Genesis es una herramienta para el diseño de componentes de Ext JS desarrollada por el Centro de Tecnologías de Gestión de Datos (DATEC) de la Facultad 6 de la UCI. Concebido como un IDE para el desarrollo de aplicaciones del propio centro, principalmente para la Plataforma de Apoyo a la Toma de Decisiones y Soluciones Integrales (PATDSI). Constituye un constructor de componentes reutilizables de Ext JS que permite diseñarlos de manera intuitiva, rápida y cómoda, de modo que promueva la usabilidad y eleve la productividad de los desarrolladores. Implementa buenas prácticas de arquitectura y diseño contribuyendo a la calidad del desarrollo (10).

La herramienta provee un entorno de diseño con las siguientes características: (10)

- Diseñador integrado por los componentes que asisten el trabajo de diseño (Espacio de Diseño, Paleta de Herramientas, Editor de Propiedades, Navegador del Diseño, Editor de Código).

- Diseño vía Sujetar-Arrastrar-Soltar componentes.

- Diseño vía Seleccionar-Mover / Dimensionar.

- Asistentes de alineación de componentes (asistente de acercamiento superior, inferior, derecho izquierdo; asistentes de alineación vertical derecho, centro, izquierdo; asistentes de alineación horizontal superior, centro, inferior).

- Facilidad Deshacer / Rehacer.

- Exportación / Importación de componentes.

- Facilidad de especificación e integración de nuevos componentes para el diseñador.

Entre sus ventajas se puede destacar que: disminuye la cantidad de esfuerzo invertido en capacitación a los desarrolladores, así como eleva la productividad al disminuir los tiempos de desarrollo. Además desarrolla activos reutilizables en PATDSI para cubrir las funcionalidades de diseño de reportes y encuestas en los respectivos módulos (10).

Sin embargo el inconveniente de esta solución como opción factible para el problema planteado, radica en que Lycan ha sido concebido para el entorno de la plataforma PATDSI y para el desarrollo de aplicaciones específicamente del centro DATEC.

Resultados del análisis

Las soluciones informáticas para la creación y edición de interfaces gráficas de usuario descritas en la investigación, centran sus objetivos en determinadas características de los entornos para los que se crearon. Con respecto a los indicadores a tener en cuenta, todas estas herramientas proveen un entorno de diseño con facilidades de uso, mediante el cual se pueden

obtener los resultados de forma sencilla e intuitiva. Esto supone un factor importante referido a la usabilidad, por lo que se tiene un buen punto de partida. Mediante este estudio se pudo constatar que excepto Qt Designer, estas soluciones son orientadas de una forma u otra al desarrollo sobre la web, aunque solo Lycan-Génesis es de naturaleza web, mientras que Qt Designer, Ext Designer y Artisteer son aplicaciones de escritorio, imposibilitando así su integración al marco de trabajo. Sin embargo, Lycan es un IDE pensado para el desarrollo en PATDSI del centro DATEC, por lo que su integración tampoco es posible en el entorno de Sauxe.

Igualmente, todos tienen basamentos en tecnologías libres, excepto Ext Designer y Artisteer, cuya principal dificultad para su comunidad de desarrolladores es que para utilizarlo es necesario pagar una licencia de software. Otros puntos importantes son: la posibilidad de realizar modificaciones en tiempo de ejecución -funcionalidad que solo Lycan brinda-; y que la concepción de estas soluciones no ha sido definida para realizar edición de interfaces visuales sin hacer un rediseño de la estructura.

A pesar de no ser posible la utilización de las herramientas analizadas por las razones antes expuestas, sí se considera que aportan el contenido teórico, así como los procedimientos a seguir en la generación de interfaces gráficas. Se reafirma entonces la necesidad de crear una solución informática que se adecue a la problemática existente, defendiendo la idea de la soberanía tecnológica, integración con el menor costo posible, y que logre la personalización de las interfaces gráficas de usuario en el marco de trabajo Sauxe sin rediseñar la estructura.

1.3 Modelo de desarrollo

El departamento de Tecnología utiliza el Modelo de desarrollo de software definido por el CEIGE versión 1.2, en el que se detalla el ciclo de vida de sus proyectos, con la incorporación de los distintos subprocesos dictados por el Nivel II de CMMI obtenido por la UCI y reconocido por el SEI como aval de la calidad de su proceso de desarrollo de software. Este modelo estandarizado establece las distintas fases por las que se debe transitar durante el desarrollo, y el conjunto de artefactos que se generan en cada una de ellas. Las fases definidas son: Inicio o Estudio preliminar y Desarrollo (11).

La solución que se desea implementar se enmarca en la fase de Desarrollo, en la cual se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los

planes del proyecto considerando los requisitos y la arquitectura. Durante el Desarrollo se refinan los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto. El objetivo de esta fase es: Obtener un sistema que satisfaga las necesidades de los clientes y usuarios finales (11).

En general propone una solución sencilla y novedosa, que se centra en el desarrollo de componentes como base tecnológica, con una mayor calidad y en menor tiempo, para su posterior uso en la construcción de productos concretos; además propone dividir el trabajo y el equipo de desarrollo para lograr mayor especialización. Su buena aplicación proporcionará en gran medida la independencia tecnológica de los sistemas finales (11).

Características

- **Centrado en la arquitectura:** La arquitectura determina la línea base, los elementos de software estructurales a partir de los elementos de la arquitectura de negocio. Interviene en la gestión de cambios y diseña la evolución e integración del producto. La arquitectura orienta las prioridades del desarrollo y resuelve las necesidades tecnológicas y de soporte para el desarrollo (11).

- **Orientado a componentes:** Las iteraciones son orientadas por el nivel de importancia arquitectónica de los componentes, los mismos son abstracciones arquitectónicas de los procesos de negocio y requisitos asociados que modelan. El componente es la unidad de medición y ordenamiento de las iteraciones (11).

- **Iterativo e incremental:** Las iteraciones son planificadas y coordinadas con el equipo de arquitectura, los clientes y la alta gerencia. Cada iteración constituye el desarrollo de componentes, los cuales son integrados al término de la iteración, permitiendo de esta manera la evolución incremental del producto (11).

- **Ágil y adaptable al cambio:** El desarrollo de las partes formaliza solamente las características principales de la solución, priorizando los talleres y las comunicaciones entre las personas. Los clientes y expertos funcionales están involucrados en el proyecto y poseen parte de la responsabilidad del éxito del mismo. Los cambios son conciliados semanalmente, discutidos y aprobados (11).

1.4 Ingeniería de Requisitos

La Ingeniería de Requisitos se define como “el uso sistemático de procedimientos, técnicas, lenguajes y herramientas para obtener el análisis, documentación, evolución continua de las necesidades del usuario y la especificación del comportamiento externo que satisfaga las necesidades del usuario.”(30)

Según Pressman (30), este proceso ayuda a los ingenieros de software a entender mejor el problema para el cual se necesita la solución y su objetivo es darle a todas las partes una explicación escrita del problema. Además incluye el conjunto de tareas que conducen a comprender cuál será el impacto del software sobre el negocio, qué es lo que el cliente quiere y cómo interactuarán los usuarios finales con el software.

1.4.1 Técnicas de captura de requisitos

Para realizar el proceso de captura de requisitos existen varias técnicas. La combinación de varias de estas técnicas es una opción factible para lograr un buen proceso de captura de requisitos, y su uso está estrechamente relacionado con las características del proyecto en el que vayan a ser aplicadas.

Entrevistas: Es un medio tradicional de obtención de requisitos. La entrevista es un método muy efectivo que permite conocer los problemas de los clientes y encontrar requisitos generales. Para aplicar este método es necesario conocer la forma en que se debe realizar una entrevista para lograr una buena comunicación entre el entrevistador y el entrevistado (33).

Prototipos: Es la representación o visualización de parte del sistema. Es una herramienta valiosa para clarificar requisitos confusos. Proveen a los usuarios un contexto para entender mejor qué información necesitan proporcionar (33).

Tormenta de ideas: Esta técnica es usada para generar nuevas ideas y encontrar la solución a cuestiones específicas. Es muy común en los comienzos del proceso de ingeniería de requisitos (33).

Observación: Este método consiste en la identificación de requisitos cuando se observan a las personas haciendo su trabajo diario. Es muy usado para encontrar requisitos adicionales cuando el usuario es incapaz de explicar los requisitos que necesita para el nuevo sistema (33).

1.4.2 Técnicas de validación de requisitos

El proceso de validación de requisitos comprende actividades que generalmente se realizan una vez que se ha obtenido una primera versión de la documentación de requisitos. Según Sommerville, la validación de requisitos “trata de mostrar que éstos realmente definen el sistema que el cliente desea. Su importancia radica en que los errores en el documento de requisitos pueden conducir a importantes costes al repetir el trabajo cuando son descubiertos durante el desarrollo o después que el sistema esté en uso.”A continuación se muestran varias de estas técnicas:

Revisión de requisitos: Los requisitos son analizados sistemáticamente por un grupo de revisores. Es un proceso manual en el que se verifica el documento de requisitos en cuanto a anomalías y omisiones y se puede gestionar de igual forma que las inspecciones de programas (12).

Generación de casos de pruebas: Los requisitos deben poder probarse. Si las pruebas para éstos se conciben como parte del proceso de validación a menudo revela los problemas en requerimientos. Si una prueba es difícil o imposible de diseñar, normalmente significa que los requerimientos serán difíciles de implementar deberían ser considerados nuevamente (12).

Construcción de prototipos: Consiste en mostrar un modelo ejecutable o semi-ejecutable a los usuarios finales y clientes para que experimenten con este modelo y valoren si satisface sus necesidades (12).

1.5 Modelo de Diseño

El Modelo de diseño ha sido definido como “una abstracción del Modelo de Implementación y su código fuente, el cual fundamentalmente se emplea para representar y documentar su diseño”. (40) El modelo de diseño puede contener: diagramas, clases, paquetes, interfaces, entre otros, que son utilizados como entrada esencial en las actividades relacionadas a la implementación.

1.5.1 Arquitectura de software

Según varios autores la arquitectura del software es el diseño de más alto nivel de la estructura de un sistema, programa o aplicación. Es importante concebir una arquitectura sólida, pues la

estructura del sistema depende en gran medida de la arquitectura de software que se utilice para desarrollarlo. Esta estructura se constituye de componentes, que nacen de la noción de abstracción, cumpliendo funciones específicas e interactuando entre sí (41).

La definición oficial ofrecida por la IEEE dicta que: “La Arquitectura del Software es la organización fundamental de un sistema formado por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución” (42).

En la arquitectura de software se determina el estilo arquitectónico y el patrón arquitectónico, cuyas definiciones se muestran a continuación:

Estilo arquitectónico: expresa la arquitectura en el sentido más formal y teórico, describiendo una clase de arquitectura, o piezas identificables de las arquitecturas empíricamente dadas. Esas piezas se encuentran repetidamente en la práctica, trasuntando la existencia de decisiones estructurales coherentes. Una vez que se han identificado los estilos, es lógico y natural pensar en reutilizarlos en situaciones semejantes que se presenten en el futuro (43).

Según Buschmann (44), un patrón como concepto general es: “una solución probada que se puede aplicar con éxito a un determinado tipo de problema que aparece con frecuencia”. De ahí que la definición propuesta por el propio Buschmann para el patrón arquitectónico sea la siguiente:

Patrón arquitectónico: expresa el esquema de organización estructural fundamental para sistemas de software. Este esquema provee un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos. Pudiera representarse como una plantilla para arquitecturas de software concretas, que especifica las propiedades estructurales de una aplicación.

1.5.2 Patrones de diseño

Un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular (44). A continuación se describen las dos vertientes principales de estos patrones:

Los **Patrones GRASP**: describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones (45).

A continuación se hace una breve descripción de los principales patrones GRASP:

- **Experto**: Asigna una responsabilidad al experto en información: la clase que posee la información necesaria para cumplir con la responsabilidad.(46)

- **Creador**: Asigna a la clase B la responsabilidad de crear una instancia de clase A (46).

- **Controlador**: Asigna la responsabilidad de administrar un mensaje de eventos del sistema a una clase (46).

- **Bajo acoplamiento**: Asigna responsabilidades de modo que se mantenga bajo acoplamiento (46).

- **Alta cohesión**: Asigna responsabilidad de modo que se mantenga alta cohesión al objetivo de las clases que lo posean (46).

Aparte de los descritos, también son patrones GRASP: **Polimorfismo, Fabricación Pura, Indirección** y **Variaciones Protegidas.**

Los **Patrones GoF**: se dieron a conocer a principios de los años 90 con el libro “Design Patterns. Elements of Reusable Object-Oriented Software”. Según su naturaleza se caracteriza en dicho libro a 23 patrones en los tres siguientes grupos:

- **Creacionales**: inicialización y configuración de objetos (47).

- **Estructurales**: separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan, para formar estructuras más grandes (47).

- **Comportamiento**: más que describir objetos o clases, describen la comunicación entre ellos (47).

1.6 Modelo de implementación

El modelo de implementación se inicia a partir de los resultados obtenidos del diseño y constituye la modelación de los aspectos físicos del sistema a desarrollar. En él se describe cómo se implementan en términos de componentes los elementos del modelo de diseño que constituyen los planos lógicos del sistema. Describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje de programación utilizado. Las dependencias entre los componentes, así como los recursos necesarios para poder ejecutar la herramienta desarrollada son aspectos que también se puntualizan en este modelo (48).

1.7 Métricas de software

Las métricas de software constituyen una medida cuantitativa que permite a los desarrolladores tener una visión profunda de la eficacia del proceso de software. En esta evaluación se reúnen los datos básicos referentes a los factores de calidad que posteriormente serán analizados, comparados con promedios anteriores y evaluados, para determinar las mejorías en términos de éstos factores (30).

Además señala que las métricas serán utilizadas para la evaluación de determinados atributos de calidad. A continuación se muestran varios de estos atributos:

- **Responsabilidad**: consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta (40).

- **Complejidad de implementación**: consiste en el grado de dificultad que tiene implementar un diseño de clases determinado (40).

- **Reutilización**: consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software (40).

- **Acoplamiento**: consiste en el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de Reutilización (40).

- **Complejidad del mantenimiento**: consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirectamente, pero fuertemente en los costes y la planificación del proyecto (40).

- **Cantidad de pruebas**: consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad (unidad) del producto (componente, módulo, clase, conjunto de clases, entre otras) diseñado (40).

1.7.1 Tamaño operacional de clases (TOC)

Está dado por el número de métodos asignados a una clase (40). La relación entre esta métrica y los atributos de calidad mencionados, se observa en la Tabla I.

Tabla I Atributos que se evalúan por (TOC).

Atributo de calidad	Modo en que lo afecta
Responsabilidad	Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
Reutilización	Un aumento del TOC implica una disminución del grado de reutilización de la clase.

Para estos atributos de calidad están definidos los siguientes criterios y categorías de evaluación:

Tabla II Rango de valores para los criterios de evaluación de la métrica (TOC).

Atributo	Categoría	Criterio
Responsabilidad.	Baja	<=Promedio
	Media	Entre promedio y 2 * Promedio
	Alta	>2*Promedio
Complejidad implementación.	Baja	<=Promedio
	Media	Entre Promedio y 2 * Promedio
	Alta	>2*Promedio
Reutilización.	Baja	>2*Promedio
	Media	Entre Promedio y 2 * Promedio
	Alta	<=Promedio

1.7.2 Relación entre clases (RC)

Está dado por el número de relaciones de uso de una clase con otra (40). La relación entre esta métrica y los atributos de calidad mencionados, se observa en la Tabla III.

Tabla III Atributos de calidad que mide (RC).

Atributo de calidad	Modo en que lo afecta
Acoplamiento	Un aumento del RC implica un aumento del Acoplamiento de la clase.
Complejidad de mantenimiento	Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
Reutilización	Un aumento del RC implica una disminución en el grado de reutilización de la clase.
Cantidad de pruebas	Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

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

Tabla IV Criterios y categorías para evaluarla métrica (RC).

Atributo	Categoría	Criterio
Acoplamiento.	Ninguna	0
	Baja	1
	Media	2
	Alta	>2
Complejidad de mantenimiento.	Baja	<=Promedio
	Media	Entre Promedio y 2 * Promedio
	Alta	>2*Promedio
Reutilización.	Baja	>2*Promedio
	Media	Entre Promedio y 2 * Promedio
	Alta	<=Promedio
Cantidad de pruebas	Baja	<=Promedio
	Media	Entre Promedio y 2 * Promedio
	Alta	>2*Promedio

1.8 Pruebas de Software

Las pruebas del software son un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones, del diseño y de la codificación. Dichas pruebas son realizadas con el objetivo de detectar errores en el sistema, por lo que se llevan a cabo durante todo el ciclo de vida del producto. Los casos de prueba especifican una forma de probar el sistema, incluyendo las entradas con las que se ha de probar, las condiciones bajo las que ha de probarse, así como los resultados esperados (40). Por otra parte Pressman (30), asegura que además de detectar errores, las pruebas tienen como objetivo medir el grado en que el software cumple con los requerimientos, además, expone que hay dos enfoques principales, las pruebas de “caja blanca” y las pruebas de “caja negra”.

1.8.1 Pruebas estructurales o de caja blanca

También conocidas como pruebas de “caja de cristal”, este es un método de diseño de casos de pruebas que utiliza la estructura de control del diseño procedimental para obtener los casos de pruebas que garanticen que (30):

- Se ejerciten al menos una vez todas las rutas independientes del módulo.
- Se ejecuten todas las decisiones lógicas en sus opciones verdadera y falsa.
- Se ejerciten todos los bucles en sus límites.
- Se usen las estructuras internas de datos para asegurar su validez.

Existen varias técnicas de pruebas de caja blanca, como se describen a continuación:

Camino básico: permite obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Para obtener el conjunto de caminos independientes se construye el grafo de flujo asociado y se calcula su complejidad ciclomática. Los casos de pruebas obtenidos garantizan que se ejecute al menos una vez cada sentencia del programa (30).

Prueba de condición: ejercita las condiciones lógicas contenidas en el módulo de un programa y su propósito es detectar no solo errores en las condiciones, sino también otro tipo de errores.

Pruebas de flujos de datos: selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variantes del programa (30).

Prueba de bucles: se centra exclusivamente en la validez de las construcciones de bucles. Se pueden definir cuatro clases diferentes de bucles: bucles simples, bucles concatenados, bucles anidados y bucles no estructurados (30).

1.8.2 Pruebas funcionales o de caja negra

Este tipo de pruebas, también conocidas como “de comportamiento”, se concentran en los requisitos funcionales del software y descubren una clase diferente de errores de los que se

descubrirán con los métodos de caja blanca (30). Las pruebas de caja negra tratan de encontrar errores en las siguientes categorías:

- Funciones correctas o faltantes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de comportamiento o desempeño.
- Errores de inicialización y término.

Para realizar las pruebas de caja negra existen varias técnicas, algunas de ellas son:

Partición equivalente: es una técnica que divide el dominio de entrada de un programa en clases a partir de las cuales pueden derivarse casos de pruebas, permitiendo examinar los valores válidos e inválidos de estas entradas existentes en el software. Además descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. Esto permite reducir el número de casos de prueba a elaborar (30).

Análisis de valores límites: los errores tienden a darse más en los límites del campo de entrada que en el centro. Esta es una técnica que complementa la partición equivalente y la mayoría de las veces se aplica de forma inconsciente. En lugar de seleccionar cualquier elemento de una clase de equivalencia, lleva a elección de casos de prueba en los extremos de la clase (30).

Prueba de comparación: este tipo de pruebas se emplea cuando la fiabilidad del software es algo crítico, (por ejemplo cuando se desarrolla para aeronaves o plantas nucleares) varios equipos de ingeniería del software desarrollan versiones independientes de una misma aplicación, usando los mismos requisitos. Todas las versiones son probadas con datos iguales, para asegurar que todas proporcionan una salida idéntica (30).

Prueba de la tabla ortogonal: esta prueba puede aplicarse a problemas en que el dominio de entrada es relativamente pequeño pero demasiado grande para solicitar pruebas exhaustivas. El método de la tabla ortogonal es útil al encontrar errores asociados con fallos localizados (30).

1.9 Tecnologías propuestas

Se estudiaron un conjunto de tecnologías a utilizar durante todo el proceso de desarrollo del módulo, debido a que la realización del mismo forma parte de un proceso iniciado por CEIGE, las tecnologías propuestas son definidas en el documento “Vista de entorno de desarrollo tecnológico de Sauxe_v2.2”, las cuales están referenciadas y brevemente explicadas a continuación:

1.9.1 Lenguajes de programación

Un lenguaje de programación es aquel elemento dentro de la informática que permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que se ponen a disposición del programador para que éste pueda comunicarse con los dispositivos de hardware y software existentes (14).

A continuación se describen los lenguajes de programación a utilizados en el desarrollo del componente.

PHP v5.2.6

PHP (acrónimo de "PHP: Hypertext Preprocessor") es un lenguaje de código abierto interpretado, de alto nivel, incrustado en páginas HTML y está orientado al desarrollo de aplicaciones web, que son interpretadas del lado del servidor. Sus sintaxis son muy similares a lenguajes como C y PERL. Puede ser utilizado en casi todos los sistemas operativos existentes, permitiendo migrar las aplicaciones de un sistema a otro sin necesidad de realizar cambios en el código. Su rapidez en la ejecución y los bajos requerimientos de consumo en los sistemas donde es desplegado lo hacen uno de los preferidos por los desarrolladores.

Se integra perfectamente a la mayoría de los Sistemas Gestores de Bases de Datos. Su mayor ventaja radica en ser un lenguaje libre, por lo que se convierte en una alternativa de muy fácil acceso. Quizás una de sus mayores desventajas radica en que promueve la creación de código desordenado, por lo que lo hace muy complejo de mantener (15).

JavaScript v1.8

JavaScript es un lenguaje basado en objetos, que se utiliza principalmente para crear páginas web dinámicas y permite el desarrollo de interfaces de usuario mejoradas. Una página web dinámica es aquella que permite la interacción entre el contenido de la misma y el usuario. JavaScript permite incorporar a dichas páginas efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario. Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios (16).

1.9.2 Lenguaje de modelado UML

La herramienta utilizada para el modelado es Visual Paradigm. Es una herramienta de diseño UML libre y profesional, diseñada para contribuir al desarrollo de software. Soporta los principales estándares como UML, SysML, BPMN y XML. Ofrece un conjunto de herramientas útiles para los equipos de desarrollo de software, necesarias para la captura de requisitos, la planificación de software, la planificación de pruebas, el modelado de clases y el modelado de datos (17).

1.9.3 Librerías y Marcos de Trabajo

El proceso de implementación de la solución se efectuará utilizando el marco de trabajo Sauxe, desarrollado por CEIGE, el cual contiene un conjunto de componentes reutilizables que provee la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo (18).

ExtJS v4

Es una librería Java Script de código abierto de alto rendimiento para la creación y desarrollo de aplicaciones web dinámicas. Provee interfaces gráficas de usuario que brindan experiencias parecidas o iguales a las que se tienen con aplicaciones de escritorio. Permite la creación de aplicaciones complejas utilizando componentes predefinidos. Es extensible para la gran mayoría de los navegadores, evitando el tedioso problema de validar el código para cada uno de estos. Entre sus principales ventajas se encuentra el balance entre Cliente-Servidor, disminuyendo la carga de procesamiento en el último, y este al tener menor carga, maneja los clientes de manera más eficiente. La comunicación asíncrona permite el intercambio de

información con el servidor sin necesidad de pedirle una acción al usuario, dando la libertad de cargar la información sin que este lo note (19).

Zend Framework v1.11

Es un marco de trabajo de código abierto para el desarrollo de aplicaciones Web con PHP. Brinda soluciones para construir sitios web modernos, robustos y seguros. Posee un bajo acoplamiento entre sus componentes, lo que posibilita la utilización de los mismos a conveniencia. Brinda una alta abstracción de bases de datos, haciendo extremadamente simple la interacción con estas, sin necesidad de escribir ninguna consulta SQL. Cuenta con módulos para el manejo de ficheros PDF, canales RSS, entre otros. Cuenta con clientes para el acceso a Web Services y robustas clases para la autenticación y el filtrado de entrada; completa documentación y pruebas de alta calidad (20).

1.9.4 Herramientas utilizadas

Servidor Web Apache v2.2.9

Es un servidor web HTTP de código abierto para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual. Es una tecnología gratuita, de código abierto y altamente configurable, de diseño modular por lo que resulta muy sencillo ampliar sus capacidades. Actualmente existen muchos módulos para Apache que son adaptables a este, y están disponibles para su instalación cuando sean necesarios. Permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor y es posible configurarlo para que ejecute un determinado script cuando esto suceda (23).

Visual Paradigm 6.4

Visual Paradigm para UML es una herramienta CASE profesional que soporta el ciclo de vida completo del desarrollo de software. El software de modelado UML ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Esta herramienta también proporciona abundantes tutoriales de UML, demostraciones interactivas y proyectos UML. Soporta UML v2.1, permite modelado colaborativo con CVS y Subversion, generación de código, ingeniería inversa, generación de bases de datos (transformación de diagramas entidad-relación en tablas de la base de datos),

importación y exportación a ficheros XML, distribución automática de diagramas, entre otras características (24).

NetBeans 7.0

El IDE NetBeans es un entorno de desarrollo integrado de código abierto para desarrolladores de software. Cuenta con todas las herramientas necesarias para crear aplicaciones profesionales de escritorio, empresariales, web y aplicaciones móviles con la plataforma Java, así como con C / C++, PHP, JavaScript y Groovy. NetBeans recibe el soporte integrado para lenguajes dinámicos, todo en una herramienta de gran alcance. El editor de PHP de NetBeans ofrece plantillas de código y generación (getters y setters), la refactorización, información sobre herramientas de parámetros, consejos y soluciones rápidas, y la finalización de código inteligente. El IDE también ofrece un completo editor de HTML, JavaScript y CSS, con la ventaja de proveer un resaltado de sintaxis completo, completamiento de código inteligente y la comprobación de errores para HTML, CSS y JavaScript, incluyendo HTML 5, JavaScript 1.7 (25).

Control de Versiones RapidSVN 0.12.0

RepidSVN es un cliente gráfico para Subversion. Es fácil de usar, tanto para los que conocen Subversion como para los que empiezan, pudiendo acceder a direcciones SVN, subir y descargar contenido y sincronizarlo con el servidor original, comprobar su estado, crear y fusionar direcciones (21).

Subversion

Subversion es una herramienta de código abierto, multiplataforma para el control de versiones de ficheros electrónicos, como son el software o la documentación. Se basa en un repositorio central que actúa como un servidor de ficheros, con la capacidad de recordar todos los cambios que se hacen tanto en sus directorios como en sus ficheros. El repositorio incrementa un número global de revisión con cada conjunto de cambios enviados (commit). Es posible copiar y renombrar ficheros; crear una rama del proyecto es tan fácil como copiar un directorio. También se puede pedir una salida con las diferencias entre dos revisiones arbitrarias, o que recupere algún sub-árbol de la revisión N (22).

Conclusiones parciales

La revisión bibliográfica permitió conocer y caracterizar los conceptos fundamentales relacionados con la situación problemática, como la interfaz gráfica de usuario, la interfaz, entre otros que contribuyen a un mejor entendimiento del contexto.

El análisis de varias soluciones informáticas para la creación y edición de interfaces, arrojó como resultado que debido a sus especificidades, ninguna de ellas puede ser integrada al marco de trabajo Sauxe, requisito fundamental a tener en cuenta para definir su viabilidad en el contexto de la solución. Sin embargo, teniendo en cuenta sus características específicas y analizando por separado los indicadores definidos: su estudio sí aporta una buena base para la elaboración de la solución en cuanto a las formas de realizar modificaciones en tiempo de ejecución, así como el tratamiento de la usabilidad en dichos sistemas.

El análisis del Modelo de desarrollo de software del CEIGE promovió la comprensión de sus principales características y sus fases con el objetivo de guiar el posterior diseño de la solución. De igual forma, el estudio de las herramientas y tecnologías definidas por el departamento para el desarrollo de sus productos, garantiza un buen punto de partida en la construcción del módulo para la generación de interfaces gráficas de usuario para el marco trabajo Sauxe.

Capítulo 2: Análisis y diseño

En este capítulo se desarrolla la propuesta de solución del módulo para la generación de interfaces. Se realiza una descripción de los procedimientos que definen el funcionamiento del módulo. Se muestran además los diferentes artefactos generados durante las fases tanto de Requisitos como de Análisis y diseño.

2.1 Modelo conceptual

El modelo conceptual podría considerarse como un diccionario visual de las abstracciones relevantes, vocabulario e información del dominio. Mediante un lenguaje visual se pueden representar las cosas del mundo real que son de interés para el dominio de forma más sencilla y entendible (46).

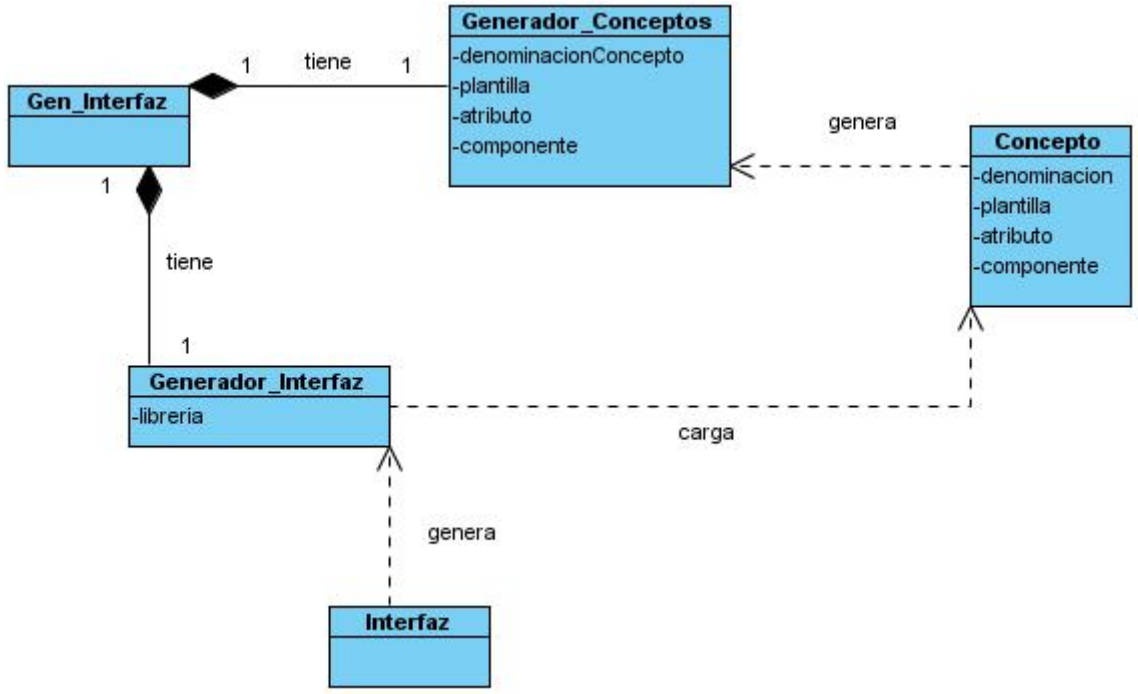


Figura 1 Modelo conceptual.

A continuación se describen los principales conceptos representados en el modelo conceptual perteneciente a la solución:

Generador Conceptos: Define los atributos que tendrá un concepto.

Concepto: Almacena los atributos del concepto generado.

Generador Interfaz: Define la librería que se utilizará para crear la interfaz.

Interfaz: Interfaz visual generada por el sistema.

2.2 Requisitos de software

2.2.1 Captura de requisitos

El equipo de desarrollo debe determinar las técnicas que se deberán usar según el producto a desarrollar, una opción factible para realizar este proceso es combinar varias de estas técnicas. Las utilizadas para el desarrollo de la solución son: Entrevista, Tormenta de ideas y Observación.

2.2.2 Requisitos funcionales

Los requisitos funcionales identificados en el proceso suman un total de 7 requisitos, incluidos en las 2 agrupaciones de requisitos como se muestra en la siguiente tabla.

Tabla V Listado de requisitos funcionales

Agrupación de requisitos	Nombre del requisito
Gestionar concepto	RF 1.1 Crear concepto
	RF 1.2 Modificar concepto
	RF 1.2 Listar concepto
Crear interfaz	RF 2.1 Crear interfaz en ExtJS 4
	RF 2.2 Crear interfaz en ExtJS 2
	RF 2.3 Crear interfaz en Dojo
	RF 2.4 Crear interfaz en JQuery

2.2.3 Requisitos no Funcionales

El desarrollo del módulo para la generación de interfaces forma parte del marco de trabajo Sauxe, desarrollado en el CEIGE. Por tanto, y de acuerdo con los requisitos que fueron establecidos por el centro al iniciar el proceso de desarrollo, los requisitos no funcionales a los que se debe acoger la aplicación a desarrollar son los que se describen a continuación:

Usabilidad

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

Rendimiento

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

Seguridad

Autenticación (Contraseña de acceso.)

Protección contra acciones no autorizadas o que puedan afectar la integridad de los datos.

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

Verificación sobre las acciones irreversibles (eliminaciones).

Portabilidad

El sistema debe ser multiplataforma.

Soporte

La aplicación contará antes de su puesta en marcha con un período de pruebas para detectar posibles errores, se brindará el servicio de instalación, se garantizará la configuración inicial y se le dará mantenimiento una vez instalada.

2.2.4 Validación de requisitos

Para lograr una correcta visualización de lo que se desea implementar en cada requisito y permitir que los clientes valoren si a través de éstos se satisfacen sus necesidades se utiliza la técnica de validación de los requisitos: **Construcción de prototipos**. Dentro de cada descripción de requisitos se podrá observar el prototipo de interfaz correspondiente al requisito que se esté especificando.

2.3 Modelo de diseño

El Modelo de diseño ha sido definido como *"una abstracción del Modelo de Implementación y su código fuente, el cual fundamentalmente se emplea para representar y documentar su diseño*" (40). El modelo de diseño puede contener: diagramas, clases, paquetes, interfaces,

entre otros, que son utilizados como entrada esencial en las actividades relacionadas a la implementación. En el presente epígrafe se muestran los elementos que se tuvieron en cuenta para el diseño de la solución, así como los resultados y artefactos generados durante esta parte del proceso de desarrollo.

2.3.1 Arquitectura de Software

Para el desarrollo del marco de trabajo Sauxe, en el Departamento de Tecnologías del CEIGE se definió el **Estilo N capas**, con cuatro capas fundamentales que se describen a continuación: (40)

- **Capa de presentación**: esta es la capa encargada de elaborar y mostrar las interfaces de los usuarios. La capa de presentación está compuesta principalmente por el marco de trabajo ExtJS y centra su desarrollo en tres componentes fundamentales archivos JS, archivos CSS, páginas clientes y páginas servidoras.

- **Capa de control o negocio:** en esta capa se encuentra ubicado el marco de trabajo Zend, el cual implementa el patrón Modelo-Vista-Controlador.

- **Capa de acceso a datos:** está ubicado el marco de trabajo Doctrine, como ORM para la comunicación con el servidor de datos mediante el protocolo PDO.

- **Capa datos:** en esta capa se encuentran ubicado los servidores de base de datos y los archivos XML donde se almacena la información de la aplicación.

En la capa de control o negocio de la arquitectura descrita anteriormente para Sauxe, se implementa el patrón arquitectónico MVC, el cual se encarga de separar los datos de la aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos: (40)

- **Modelo:** está compuesto por datos, reglas de negocio y las funcionalidades correspondientes para la comunicación con el encargado de persistir los datos.

- **Controlador:** gestiona las entradas y las respuestas del sistema al usuario.

- **Vista:** muestra la información del modelo al usuario.

2.3.2 Patrones de diseño

En la implementación del sistema se utilizan patrones de diseño que pertenecen a la vertiente GRASP como se describe a continuación:

Patrones GRASP:

- **Experto**: Se evidencia el uso de este patrón en todas las clases a utilizar en el módulo pues cada clase conoce su información y es la encargada de implementar las funcionalidades que les corresponde como por ejemplo la clase GenInterfazController.

- **Creador**: Su uso se evidencia en la clase controladora pues es la que se encarga de la creación de objetos de varias clases, como son ExtJS4Model, entre otras.

- **Controlador**: En el sistema las clases GenInterfazController y GenConceptoController se encargan de llevar el control de todos los eventos relacionados con el negocio. Implementan las funcionalidades que dan respuesta a las peticiones del usuario.

- **Bajo acoplamiento**: El uso de este patrón se evidencia en la poca relación existente entre las clases que conforman el módulo.

- **Alta cohesión**: El uso de éste patrón indica que la información almacenada en las clases debe ser coherente y relacionada a lo que se maneja en esas clases. Se evidencia su uso en todas las clases como por ejemplo en GenInterfazController.

2.3.3 Diagrama de clases del diseño con estereotipos web

Los diagramas de clases especifican las diferentes clases que serán utilizadas en el sistema, así como las relaciones que existen entre ellas. A continuación se muestran los diagramas de clases del diseño para las agrupaciones de requisitos 1 Gestionar conceptos y 2 Crear interfaz.

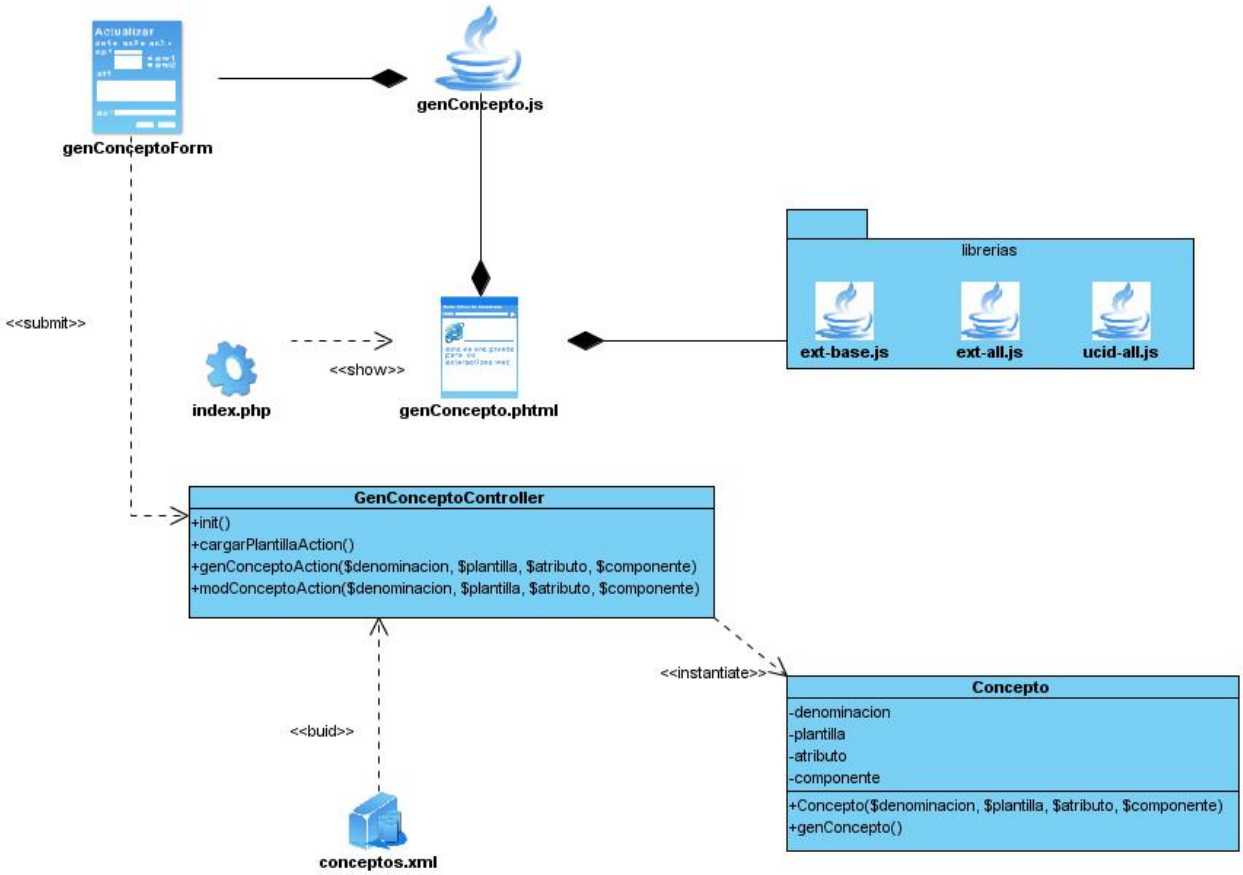


Figura 2 Diagrama de clases del diseño con estereotipos web para la agrupación de requisitos 1

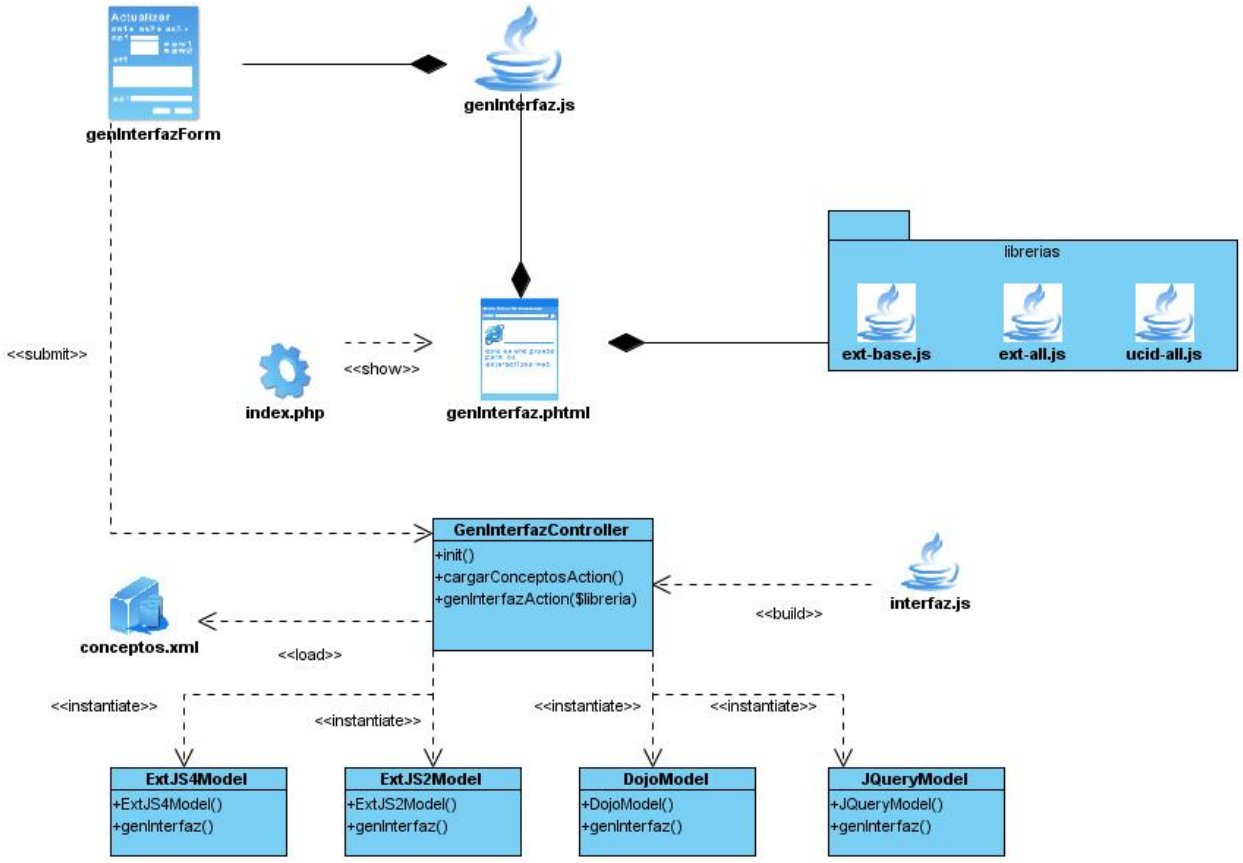


Figura 3 Diagrama de clases del diseño con estereotipos web para la agrupación de requisitos 2

2.3.4 Modelo de datos

El modelo de datos del módulo a desarrollar se basa en un XML, el cual contiene un concepto y que a su vez tiene atributos. La relación del concepto con los atributos es de uno a muchos, pues un concepto puede tener muchos atributos.

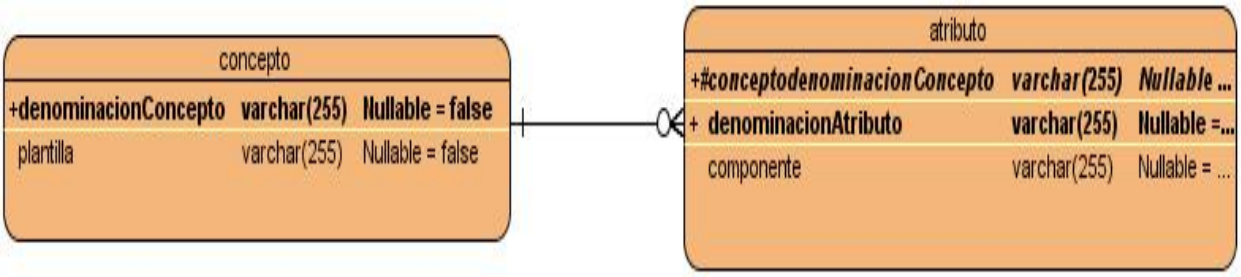


Figura 4 Modelo de datos

Conclusiones parciales

El modelo conceptual mostrado en el capítulo puntualiza los conceptos relacionados con el dominio del problema y su representación conforma una guía visual para la implementación de la solución tributando directamente a la fase de requisitos.

La captura, especificación y validación de los siete requisitos funcionales mostrados en dos agrupaciones de requisitos generales, aportó una explicación escrita del problema y una mayor comprensión del comportamiento que se necesita implementar en la solución. Los requisitos no funcionales que debe cumplir la herramienta, garantizan que su desempeño es el adecuado para su integración en Sauxe.

La utilización de estilos y patrones arquitectónicos brinda solidez a la arquitectura, garantizando que la estructura propuesta rija de forma correcta el posterior diseño, a la vez que los patrones de diseño especificados, permiten ganar en reutilización y brindan robustez a la solución.

La realización de los diagramas de clases del diseño con estereotipos web y el modelo de datos, permitió representar: las clases y relaciones que componen el sistema y la estructura que tendrán los datos de la herramienta respectivamente, de forma tal que a través de estos diagramas se facilitó el entendimiento previo al proceso de implementación.

Capítulo 3: Implementación y Prueba

El siguiente capítulo aborda los elementos relacionados con la implementación del Módulo para la generación de interfaces gráficas de usuario para el marco de trabajo Sauxe, así como lo referente a las pruebas realizadas al sistema para validar su correcto funcionamiento. En el desarrollo se describen los estándares de codificación utilizados en función de garantizar el entendimiento del código fuente. Se muestran los casos de pruebas diseñados para realizar las pruebas funcionales, así como los resultados obtenidos, validando así que los requisitos identificados fueron implementados correctamente.

3.1 Modelo de implementación

3.1.1 Diagrama de despliegue

Para modelar la vista de despliegue estática del marco de trabajo Sauxe (sistema al cual se integra la solución) se utiliza el diagrama de despliegue siguiente:

Figura 5 Diagrama de despliegue del marco de trabajo Sauxe

3.2 Estándares de codificación.

Un factor importante para la implementación lo constituye la forma en que se construye el código fuente, esencialmente su organización y el estilo de codificación utilizado en el desarrollo. Una buena estructuración del código, mejora la lectura del software, permitiendo entender código nuevo rápidamente y más a fondo (10).

El uso de estándares de codificación permite lograr un código más legible y reutilizable, de tal forma que se pueda aumentar su mantenibilidad a lo largo del tiempo (49). Para el desarrollo de la solución, se utilizaron algunos de los estándares y normas de codificación propuestos como parte de la Línea de Arquitectura determinada para el proyecto ERP Cuba, los cuales se muestran a continuación:

- **Notación Húngara:** definir prefijos para cada tipo de datos y según el ámbito de las variables. La idea de esta notación es la de dar mayor información al nombre de la

variable, método o función definiendo en ella un prefijo que indique su tipo de dato o ámbito (50).

- **Notación PascalCasing:** es como la notación húngara pero sin prefijos. En este caso los identificadores y nombres de las variables, métodos y funciones están compuestos por múltiples palabras juntas, iniciando cada palabra con letra mayúscula (51).

- **Notación CamelCasing:** es parecido al PascalCasing con la excepción que la letra inicial del identificador no debe estar en mayúscula (51).

Nomenclatura de las clases

Los nombres de las clases comienzan con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación PascalCasing. Con sólo leerlo se reconoce su propósito.

- **Nomenclatura según el tipo de clases:**
  - **Clase controladora**: después del nombre llevan la palabra “Controller”. Ejemplo: GenInterfazController.

- **Clases del modelo:**
  - **Business (Negocio):** las clases que se encuentran dentro de business después del nombre llevan la palabra “Model”. Ejemplo: ExtJS4Model.

Nomenclatura de las funciones

El nombre a emplear para las funciones se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing, y en caso de ser una acción de la clase controladora se debe especificar el nombre de dicha acción en minúscula y seguido el sufijo “Action”. Ejemplo: genConceptoAction.

Nomenclatura de las variables

El nombre a emplear para las variables se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing. Ejemplo: $atributo.

Normas de comentariado

Se debe comentar todo lo que se haga dentro del desarrollo, establecer las pautas que conlleven a lograr un código más legible y reutilizable y así se pueda aumentar su mantenimiento a lo largo del tiempo.

Estilo del código

En la implementación, al escribir las sentencias en php, se utilizarán los tabs del lenguaje como se muestra en la Figura 6:

```
<?php
//código
?>
```

Figura 6 Estilo del código

- **Sangría o indexado:** La política de sangría a utilizar en la implementación es por **tab**. Ejemplo:

```
<?php
/**
 * Indentation
 */
class Example {
    var $theInt = 1;
    function foo($a, $b) {
    switch ( $a) {
    case 0 :
    $other->doFoo ();
    break;
    default :
    $other->doBar ();
    }
    }
    function bar($v) {
    for($i = 0; $i < 10; $i ++) {
    $v->add ( $i );
    }
    }
}
?>
```

Figura 7 Estilo del código: Sangría o indexado

- **Brazas o llaves:** En la declaración de clases o interfaces, métodos, bloques y switch, la apertura de llaves se hace en la misma línea. Ejemplo:

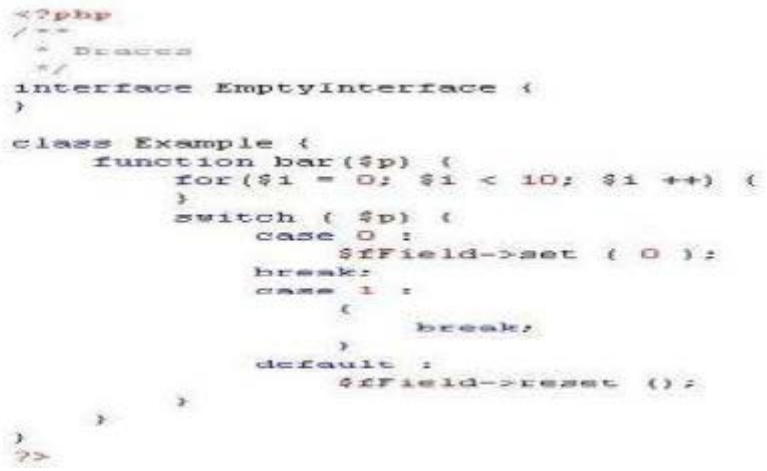


Figura 8 Estilo del código: Brazas o llaves

3.3 Métricas para validar el diseño

Según Pressman (30), la medición permite tener una visión del proceso de software, pues proporciona un mecanismo para lograr una evaluación objetiva. La calidad de un sistema, depende directamente de los requisitos que detallan el problema, del diseño que modela la solución, del código ejecutable del programa y de las pruebas que se ejecutan al software para detectar errores. Por tanto, se considera de vital importancia evaluar con medidas técnicas la calidad del diseño en la presente investigación y se utilizan las métricas TOC y RC.

3.3.1 Resultados obtenidos al aplicar (TOC)

En la siguiente tabla se recoge como datos las 6 clases a las cuales se le aplicó la métrica TOC en la columna **Clase**, la **cantidad de métodos** que tiene cada una en la columna correspondiente con un total de 15 y los atributos de calidad que utiliza esta métrica en las demás columnas y en ellas se representa la categorización en baja, media o alta según los criterios que aparecen en la Tabla II para la cual se tomó el valor promedio obtenido en la siguiente tabla con un valor de 2.5.

Tabla VI Evaluación de la métrica (TOC).

Clase	Cantidad de Procedimientos	Responsabilidad	Complejidad	Reutilización
GenConceptoController	4	Media	Media	Media

GenInterfazController	3	Media	Media	Media
ExtJS4Model	2	Baja	Baja	Alta
ExtJS2Model	2	Baja	Baja	Alta
DojoModel	2	Baja	Baja	Alta
JQueryModel	2	Baja	Baja	Alta

Las gráficas mostradas a continuación muestran el resultado de haber aplicado (TOC).

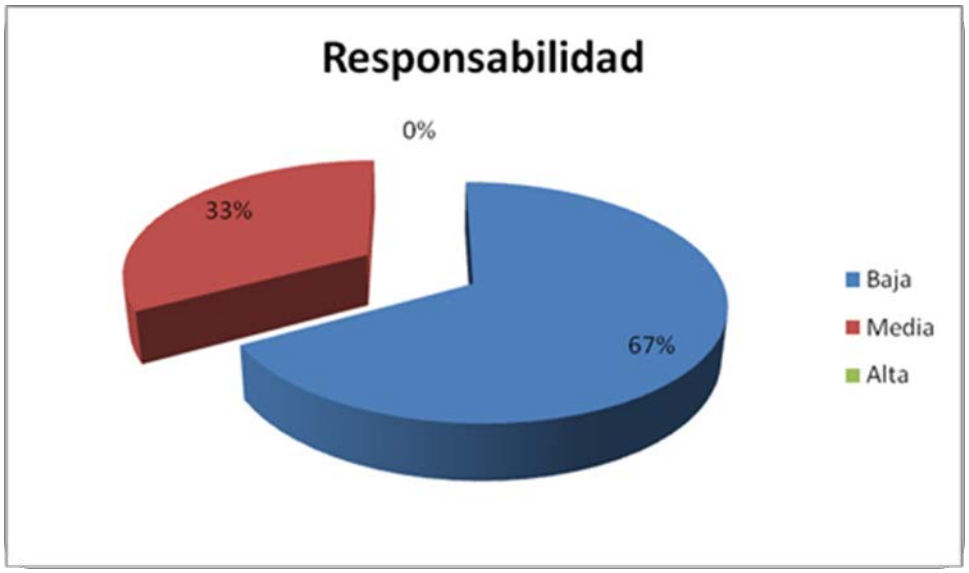


Figura 9 Resultados obtenidos de la evaluación de la métrica TOC para el atributo Responsabilidad.

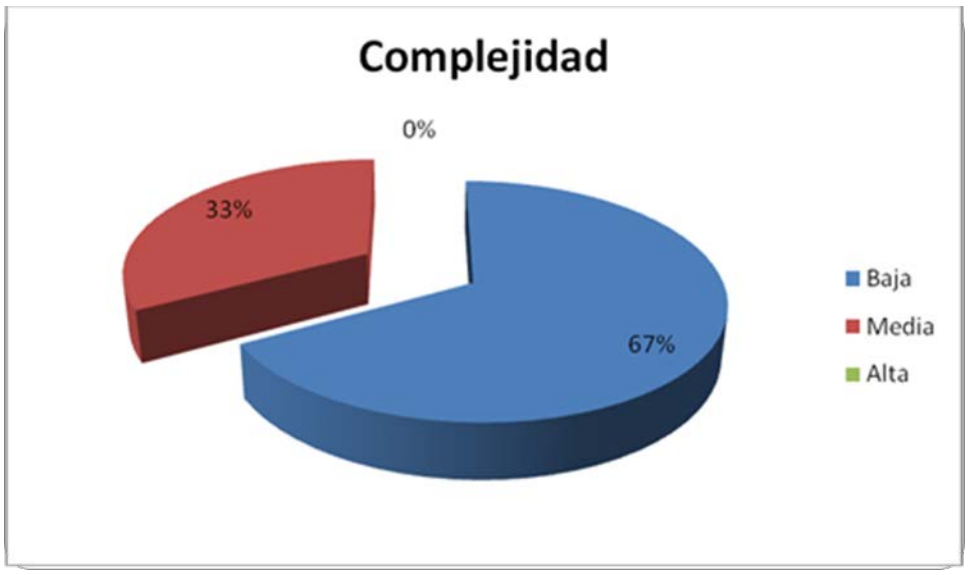


Figura 10 Resultados obtenidos de la evaluación de la métrica TOC para el atributo Complejidad.

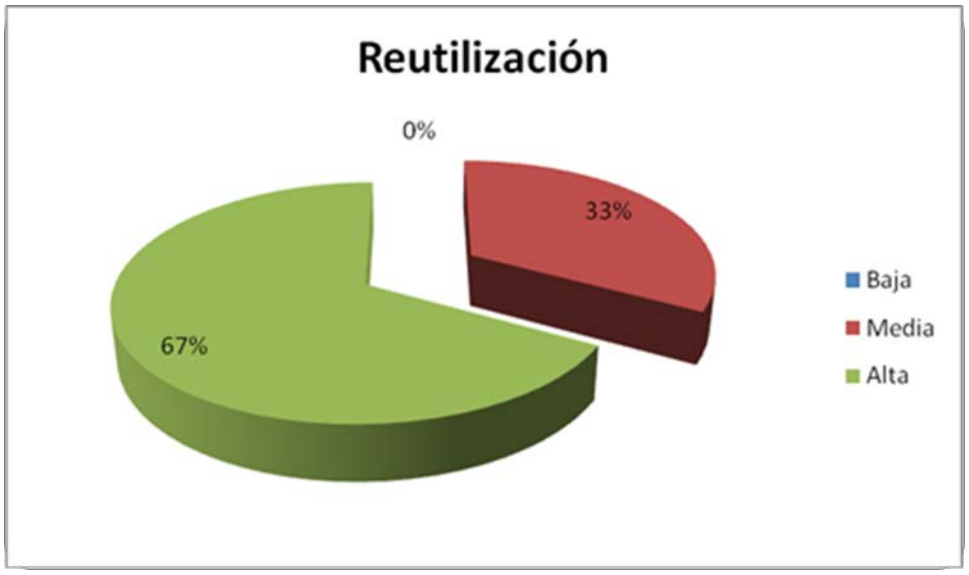


Figura 11 Resultados obtenidos de la evaluación de la métrica TOC para el atributo Reutilización.

Análisis de los resultados obtenidos al evaluar la métrica TOC

Haciendo un análisis de los resultados obtenidos para los atributos de la métrica TOC apreciables en las gráficas de pastel mostradas anteriormente, se puede observar que la mayoría de las clases que conforman el sistema para los atributos responsabilidad y complejidad están dentro de la categoría Baja para un 67% del total, lo que representa que las clases no tienen mucha responsabilidad y no son tan complejas y en caso de ocurrir algún cambio en el sistema de clases del componente, estaría menos comprometido el funcionamiento correcto del mismo. Mientras que el atributo Reutilización cuenta con igual por ciento pero en la categoría Alta mostrando así que el componente cuenta con una elevada reutilización que permite que las clases puedan ser utilizadas. Por lo que se concluye que los resultados obtenidos según esta métrica son positivos.

3.3.2 Resultados obtenidos al aplicar (RC)

En la siguiente tabla se muestra como datos las 6 clases a las cuales se le aplicó la métrica RC en la columna **Clase**, la cantidad de relaciones que tiene cada una respecto a las demás clases en la columna **Cantidad de Relaciones de Uso**, junto a los atributos de calidad de Acoplamiento, Complejidad de Mantenimiento, Reutilización y Cantidad de Pruebas en sus columnas correspondientes. Al promediar la cantidad de relaciones que tiene cada clase se

obtuvo un resultado de relaciones de dependencia por clase de 0.2 el cual se utiliza en la Tabla IV para hallar el valor de la columna criterio.

Tabla VII Evaluación de la métrica (RC).

Clase	Cantidad de Relaciones de uso	Acoplamiento	Complejidad mantenimiento	Reutilización	Cantidad de pruebas
GenConceptoController	1	Baja	Media	Media	Media
GenInterfazController	1	Baja	Media	Media	Media
ExtJS4Model	0	Ninguna	Baja	Alta	Baja
ExtJS2Model	0	Ninguna	Baja	Alta	Baja
DojoModel	0	Ninguna	Baja	Alta	Baja
JQueryModel	0	Ninguna	Baja	Alta	Baja

Las gráficas mostradas a continuación muestran el resultado de haber aplicado (RC).

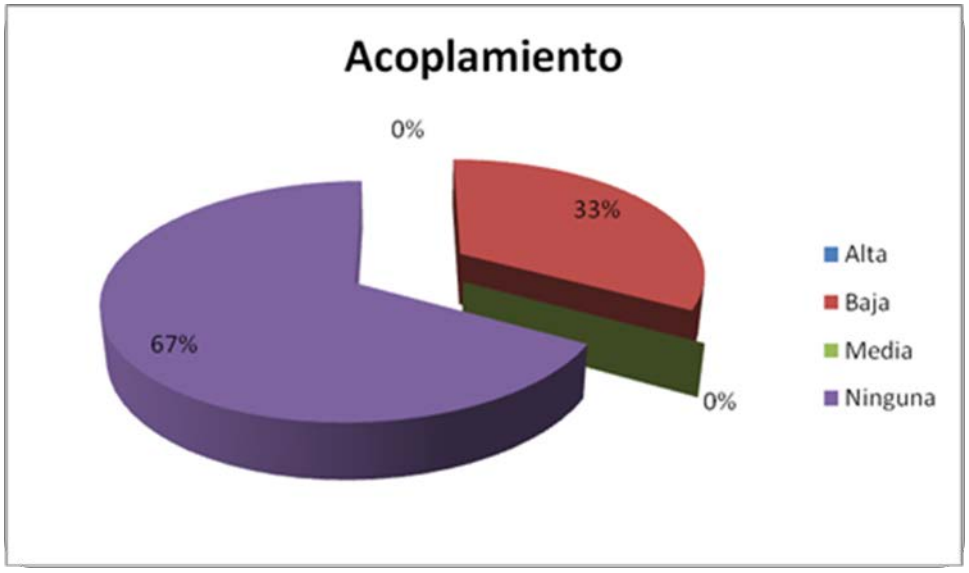


Figura 12 Resultados obtenidos de la evaluación de la métrica RC para el atributo Acoplamiento.

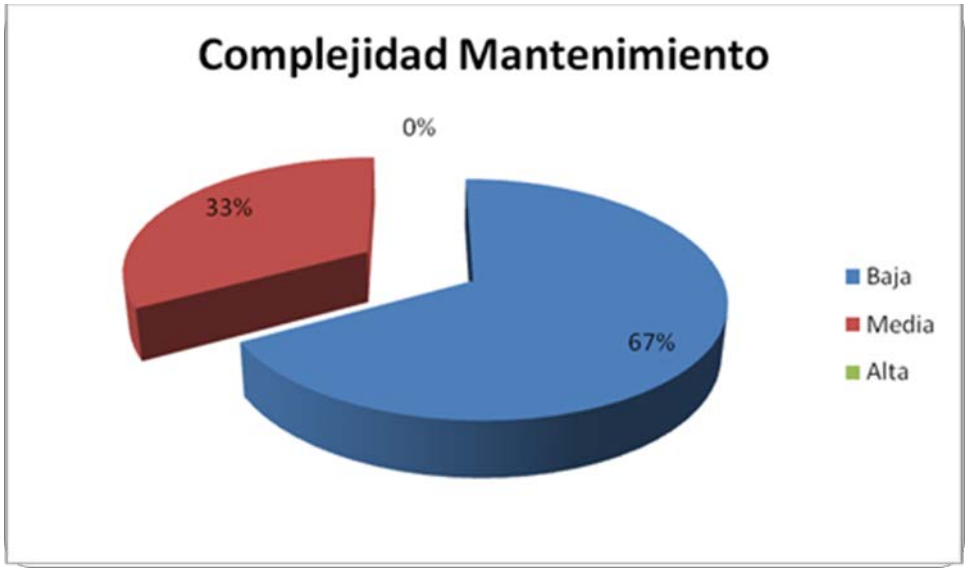


Figura 13 Resultados obtenidos de la evaluación de la métrica RC para el atributo Mantenimiento.

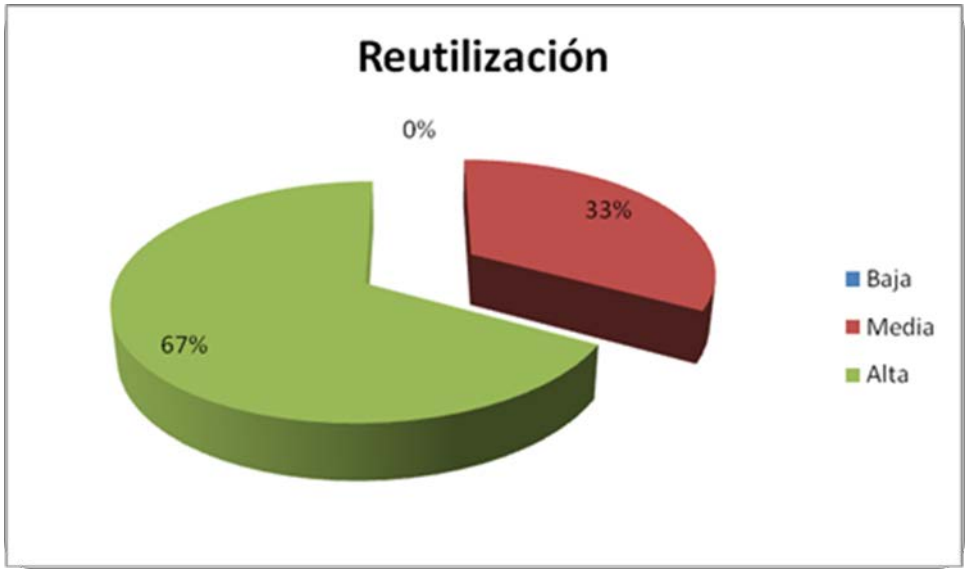


Figura 14 Resultados obtenidos de la evaluación de la métrica RC para el atributo Reutilización.

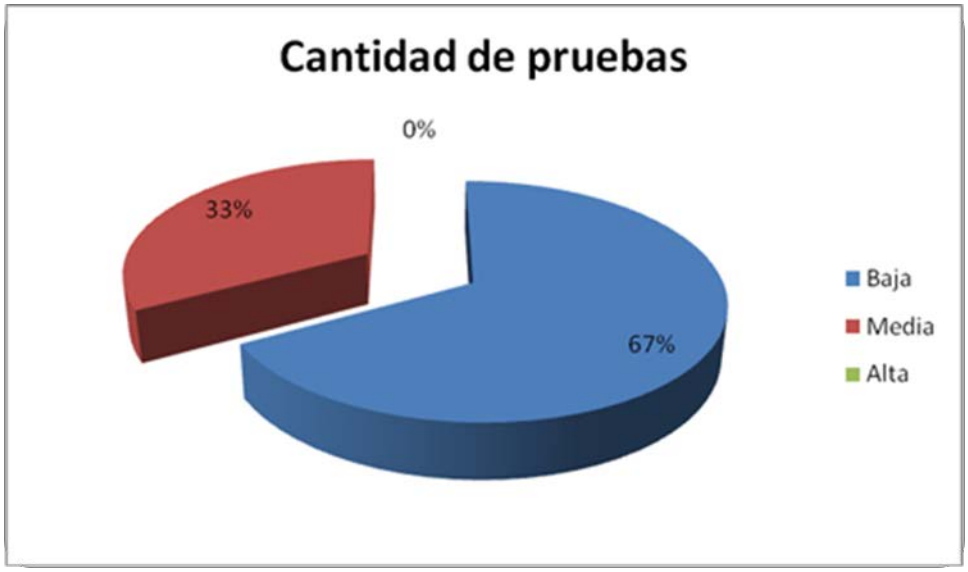


Figura 15 Resultados obtenidos de la evaluación de la métrica RC para el atributo Cantidad de pruebas.

Análisis de los resultados obtenidos al evaluar la métrica RC

Los resultados obtenidos durante la evaluación del instrumento de medición de la métrica RC demuestran que las clases del diseño poseen un bajo acoplamiento, ya que para este atributo la categoría Ninguno suma un 67% del total, mostrando igual por ciento en la categoría alta del atributo reutilización. Los atributos complejidad de mantenimiento y cantidad de pruebas, sumaron un 67 % en la categoría baja lo que demuestra que no es necesario un elevado esfuerzo en el momento de realizar cambios, rectificaciones y pruebas al software.

3.4 Pruebas de Software

3.4.1 Resultados de la aplicación de las pruebas de caja blanca

La técnica utilizada para realizar las pruebas de caja blanca al módulo para la generación de interfaces es la del **camino básico**. A continuación se muestra el fragmento del código y el grafo generado durante la aplicación de esta técnica a la funcionalidad **generarInterfaz(),** perteneciente a la clase ExtJS4Model.

```
      public function generarInterfaz($concepto){
/*1*/      $colft="";
/*1*/      $con=  new SimpleXMLElement($_SERVER['DOCUMENT_ROOT']."/../apps/herramientas/gi/comun/rec
/*2*/      foreach($con->concepto->atributos->atributo as $atributo){
/*3*/         if($col==""){
/*4*/            $col="{\n  name: '".(String)$atributo->denominacion."'\n}";
/*4*/            $cold="{\n  text: '".(String)$atributo->denominacion."',\n  flex: 1,\n  dataIndex:
/*5*/            switch((String)$atributo->componente){
/*6*/                case 'Arbol':
/*7*/                    $codigo.="var stTrForm".(String)$atributo->denominacion." = Ext.create('Ext
/*7*/                    $colft="TrForm".(String)$atributo->denominacion;
                        break;
/*8*/                case 'Checkbox':
/*9*/                    $colf=" {xtype: 'fieldcontainer',\n  defaultType: 'checkboxfield',\n    ite
                        break;
/*10*/               case 'Combobox':
/*11*/                   $codigo.="var stcmb".(String)$atributo->denominacion."=Ext.create('Ext.data
/*11*/                   $colf="cmb".(String)$atributo->denominacion;
                        break;
/*12*/               case 'Datefield':
/*13*/                   $colf="{xtype: 'datefield',\n anchor: '100%',\n fieldLabel: '".(String)$atr
                        break;
/*14*/               case 'Radio':
/*15*/                   $colf="{xtype: 'fieldcontainer',\n   defaultType: 'radiofield',\n   default
                        break;
/*16*/               case 'TextArea':
/*17*/                   $colf="{xtype: 'textareafield',\n  grow : true,\n  fieldLabel: '".(String)$
                        break;
/*18*/               case 'Textfield':
/*19*/                   $colf="{xtype:'textfield',\n  fieldLabel:'".(String)$atributo->denominacion
                        break;
               }
```

```
                }
/*20*/          else{
/*21*/             $col.=",{\n  name: '".(String)$atributo->denominacion."'\n}";
/*21*/             $cold.=",{\n  text: '".(String)$atributo->denominacion."',\n  flex: 1,\n  dataIndex
/*22*/             switch((String)$atributo->componente){
/*23*/                 case 'Arbol':
/*24*/                     $codigo.="var stTrForm".(String)$atributo->denominacion." = Ext.create('Ext
/*24*/                     $colft.=",TrForm".(String)$atributo->denominacion;
                         break;
/*25*/                 case 'Checkbox':
/*26*/                     $colf.=", {xtype: 'fieldcontainer',\n  defaultType: 'checkboxfield',\n  ite
                         break;
/*27*/                 case 'Combobox':
/*28*/                     $codigo.="var stcmb".(String)$atributo->denominacion."=Ext.create('Ext.data
/*28*/                     $colf.=",cmb".(String)$atributo->denominacion;
                         break;
/*29*/                 case 'Datefield':
/*30*/                     $colf.=",{xtype: 'datefield',\n anchor: '100%',\n fieldLabel: '".(String)$a
                         break;
/*31*/                 case 'Radio':
/*32*/                     $colf.=",{xtype: 'fieldcontainer',\n   defaultType: 'radiofield',\n   defau
                         break;
/*33*/                 case 'TextArea':
/*34*/                     $colf.=",{xtype: 'textareafield',\n  grow : true,\n  fieldLabel: '".(String
                         break;
/*35*/                 case 'Textfield':
/*36*/                     $colf.=",{\n  xtype:'textfield',\n  fieldLabel:'".(String)$atributo->denomi
                         break;
                 }
             }
/*37*/     }
/*38*/      $add="var btnAdicionar".(String)$con->concepto->denominacion." = Ext.create('Ext.Button',
/*38*/      $mod="var btnModificar".(String)$con->concepto->denominacion." = Ext.create('Ext.Button',
/*38*/      $del="var btnEliminar".(String)$con->concepto->denominacion." = Ext.create('Ext.Button',
/*38*/      $grid="var stGp".(String)$con->concepto->denominacion." = Ext.create('Ext.data.ArrayStore
/*38*/      $arbol="var stTr".(String)$con->concepto->denominacion." = Ext.create('Ext.data.TreeStore
/*38*/      $ucid="UCID.portal.cargarAcciones(window.parent.idFuncionalidad);\n";
/*39*/     if($colft==""){
/*40*/        $fun="var winAdicionar".(String)$con->concepto->denominacion.";\n var winModificar".(St
         }
/*41*/     else{
/*42*/         $fun="var winAdicionar".(String)$con->concepto->denominacion.";\n var winModificar".(S
             }

/*43*/     switch((String)$con->concepto->plantilla){
/*44*/         case 'CRUD-Simple':
/*45*/             $codigo.=$add.$mod.$del.$ucid.$fun.$grid;
/*45*/             $item="Gp".(String)$con->concepto->denominacion;
                 break;
/*46*/         case 'Arbol y Grid':
/*47*/             $codigo.=$add.$mod.$del.$ucid.$fun.$grid.$arbol;
/*47*/             $item="Gp".(String)$con->concepto->denominacion.",Tr".(String)$con->concepto->den
                 break;
             }

/*48*/     $archivo=$concepto.".js";
/*48*/     $fp = fopen($archivo, "w");
/*48*/     fwrite($fp,"var perfil = window.parent.UCID.portal.perfil;". PHP_EOL."UCID.portal.cargarE
/*48*/     fclose($fp);
/*48*/     echo "Interfaz generada";
     }
 }
```

Figura 16 Código fuente de la funcionalidad generarInterfaz().

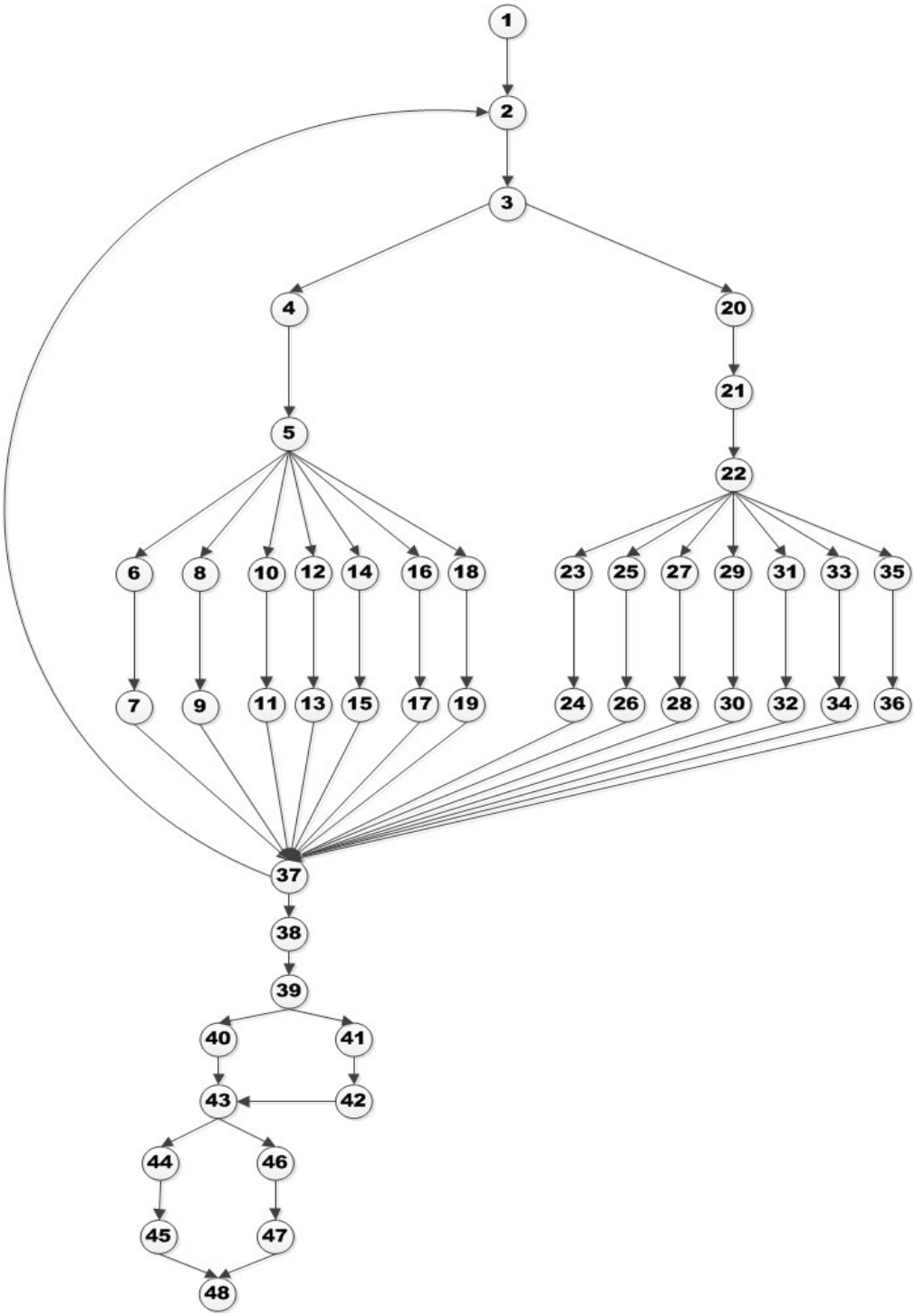


Figura 17 Grafo de flujo correspondiente a la funcionalidad generarInterfaz().

La complejidad ciclomática es la métrica de software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Esta métrica calcula la cantidad de caminos independientes de cada una de las funcionalidades del programa. También provee el límite

superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. El cálculo de la complejidad ciclomática se realiza después de construir el grafo y para ello se utilizan tres fórmulas que aseguran que esta métrica se ha calculado correctamente si coinciden sus resultados (30).

1. V(G)=R donde **R** representa la cantidad total de regiones.
V(G)=7

2. V(G)=A–N+2 donde **A** es el número de aristas del grafo de flujo y **N** es el número de nodos.
V(G)=53–48+2
V(G)=7

3. V(G)=P+1 donde **P** es el número de nodos predicado contenidos en el grafo de flujo (se denomina nodo predicado a los nodos de los cuales parten dos o más aristas).
V(G)=6+1
V(G)=7

Posterior al cálculo de las fórmulas mencionadas, se pudo constatar que las tres dan el mismo resultado. Por tanto se puede afirmar que la complejidad ciclomática del método **generarInterfaz()** tiene un valor de 7. Este valor significa que existen 7 caminos posibles para el flujo del método:

Camino básico #1: 1-2-3-4-5-6-7-37-38-39-40-43-44-45-48
Camino básico #2: 1-2-3-4-5-8-9-37-38-39-40-43-44-45-48
Camino básico #3: 1-2-3-4-5-6-7-37-38-39-40-43-46-47-48
Camino básico #4: 1-2-3-4-5-6-7-37-38-39-41-42-43-44-45-48
Camino básico #5: 1-2-3-4-5-6-7-37-38-39-41-42-43-46-47-48
Camino básico #6: 1-2-3-4-5-6-7-37-2-20-21-22-23-24-37-38-39-41-42-43-46-47-48
Camino básico #7: 1-2-3-4-5-6-7-37-2-20-21-22-23-24-37-38-39-40-43-44-45-48

Además, este valor representa la mínima cantidad de casos de prueba para el procedimiento, teniendo en cuenta que se realiza un caso de prueba por cada camino básico. A continuación se muestra el caso de prueba para el camino básico #1, los casos de prueba para los restantes caminos se encuentran entre los artefactos generados en el desarrollo del sistema:

Caso de prueba para el camino básico #1: Si $col==” ”, Si $atributo->componente== ‘Arbol’, Si $colft==” ”, Si $con->concepto->plantilla==’CRUD-Simple’

Con la realización de las pruebas de caja blanca utilizando la técnica del camino básico se garantizó que todas las sentencias del programa se han ejecutado al menos una vez.

3.4.2 Resultados de la aplicación de las pruebas funcionales o de caja negra

Para realizar este tipo de pruebas, se utilizó la técnica de Partición equivalente para la cual se confeccionaron los diseños de casos de pruebas por cada funcionalidad del sistema, los cuales pueden ser observados en el expediente del proyecto Sauxe. Para comprobar la calidad del módulo para la generación de interfaces, se realizaron dos iteraciones de revisiones por el grupo de calidad del CEIGE. Las no conformidades encontradas se clasificaron en:

- No conformidades detectadas en la aplicación.
- No conformidades detectadas en la documentación.

En la Tabla VIII se muestra la cantidad de no conformidades detectadas en la aplicación por cada iteración.

Tabla VIII Resultados de las pruebas funcionales por cada iteración

Tipo de no conformidad	Pruebas exploratorias	Primera iteración	Segunda iteración
Aplicación	10	3	0

3.5 Validación de la investigación

Con el objetivo de validar la propuesta se escogieron dos miembros del proyecto Marco de trabajo para el desarrollo de aplicaciones web de gestión, para realizar la generación de una interfaz en dos maneras diferentes: de forma manual y con la herramienta. Para lograr una estimación del tiempo de generación de la interfaz se realizó un cuestionario con dos preguntas abiertas. Las respuestas a estas preguntas indicaron los tiempos de realización de cada una de las tareas pertenecientes a la generación, que está conformada por la creación de un concepto (Ch) y la generación de la interfaz (Ih).

Para dar cumplimiento a la tarea, los implicados realizaron las dos actividades necesarias para generar una interfaz, en forma manual y con la herramienta. Los resultados obtenidos en el cuestionario se muestran a continuación, en ellos el tiempo de duración de estas tareas estará representado en horas (8h cada jornada laboral).

Tabla IX Resultados del cuestionario aplicado para el primer implicado

	Ch	Ih	Total
De forma manual	3	4	7
Con la herramienta	0.15	0.1	0.25

Tabla X Resultados del cuestionario aplicado para el segundo implicado

	Ch	Ih	Total
De forma manual	2	3.5	5.5
Con la herramienta	0.15	0.1	0.25

Antes de aplicada la propuesta, se puede observar que la duración de cada tarea varía en dependencia de la persona que la realice. Luego de aplicada la propuesta se observa una disminución del tiempo de generación de cada tarea.

Los resultados del cuestionario revelan además que de forma manual, la generación tiene una duración aproximada de 6 horas, para una sola persona. Mientras que haciendo uso de la herramienta, solo una duración de 0,25 horas, para una reducción del **95,83%** del tiempo de generación que se dedicaba antes de aplicada la propuesta.

3.6 Validación del módulo para la generación de interfaces

Para la validación de la herramienta se tuvieron en cuenta tres indicadores: 1) factores de calidad, 2) el alcance de los requisitos y 3) la aceptación del cliente.

Factores de calidad

Funcionalidad: El módulo mantiene un conjunto apropiado de funciones para las tareas y los objetivos del usuario especificados.

Fiabilidad: El módulo es capaz de detectar los tipos de error, recuperarse y notificar al usuario el tipo de error ocurrido, manteniendo así todas sus funcionalidades.

Usabilidad: El módulo le permite al usuario entender si el software es idóneo, y cómo puede usarse para las tareas, especificando en las etiquetas de cada funcionalidad y los campos de cada interfaz, títulos asociados a su función de negocio. El uso de iconografía apropiada y de nombres de cada interfaz de no más de 30 caracteres le posibilitan al usuario aprender a utilizar el software. No se utilizan más de tres interfaces para lograr una funcionalidad completa, y se diferencian los mensajes de información de los mensajes de error utilizando íconos para cada tipo, brindándole a la herramienta una interfaz atractiva.

Mantenibilidad: La utilización de los estilos y patrones arquitectónicos, garantizó la flexibilidad, permitiendo agregar nuevas funcionalidades o modificar alguna existente sin romper la estructura y consistencia de los componentes.

Portabilidad: La solución puede ser utilizada en los navegadores más utilizados y ejecutado bajo cualquier sistema operativo.

Alcance de los requisitos

Del total de 7 requisitos de software identificados, se logró implementar el 100 %, dando cumplimiento así a todas las especificaciones planteadas por los clientes.

Aceptación del cliente

Con el objetivo de validar que la herramienta cumple las características planteadas por los clientes o especialistas, se creó una comisión en el proyecto Marco de trabajo para el desarrollo de aplicaciones web de gestión, encargada de someter la solución a una revisión técnica. Esta determinó que la herramienta está estable y lista para su uso.

Conclusiones parciales

Las normas y estándares de codificación utilizados permitieron brindarle legibilidad y escalabilidad al código fuente.

La evaluación aplicada al diseño mediante las métricas TOC y RC, evidencia niveles satisfactorios como el 67 % de baja responsabilidad e igual porciento de reutilización de las clases del sistema, así como los niveles de los restantes indicadores evaluados.

Las pruebas estructurales realizadas al código a partir de la técnica del camino básico, evidencian una correcta implementación, que garantizó que todas las sentencias del programa se ejecutaron al menos una vez.

Los resultados de las pruebas funcionales, en las que con dos iteraciones totales se logró dar respuesta a las no conformidades detectadas, acreditan la correcta puesta en práctica de los requisitos funcionales.

El pre experimento realizado evidencia la disminución en un 95,83 % del tiempo de generación de interfaces gráficas de usuario para el marco de trabajo Sauxe, dándole cumplimiento a los objetivos específicos y al objetivo general propuesto para la investigación.

Conclusiones

El análisis de varias soluciones informáticas para la creación y edición de interfaces, arrojó como resultado que debido a sus especificidades, ninguna de ellas puede ser integrada al marco de trabajo Sauxe, aunque su estudio sí aporta una buena base para la elaboración de la solución necesaria.

El modelo conceptual permitió puntualizar los conceptos relacionados con el dominio del problema y su representación conforma una guía visual que tributa directamente a la fase de requisitos. La captura, especificación y validación requisitos funcionales, aportó una explicación escrita del problema y una mayor comprensión del comportamiento que se necesita implementar en la solución. Los requisitos no funcionales que debe cumplir la herramienta, garantizan que su desempeño es el adecuado para su integración en Sauxe.

La utilización de estilos y patrones arquitectónicos brinda solidez a la arquitectura, garantizando que la estructura propuesta rija de forma correcta el posterior diseño, a la vez que los patrones de diseño especificados, permiten ganar en reutilización y brindan robustez a la solución.

La evaluación aplicada al diseño mediante las métricas TOC y RC, evidencia niveles satisfactorios como el 67 % de baja responsabilidad e igual porciento de reutilización de las clases del sistema, así como los niveles de los restantes indicadores evaluados.

Las pruebas de software realizadas a la solución evidencian una correcta implementación y puesta en práctica de los requisitos funcionales.

El pre experimento realizado evidencia una disminución en un 95,83 % del tiempo de generación de interfaces gráficas de usuario para el marco de trabajo Sauxe. El tiempo de duración de la tarea se redujo en 5,75 h, dándole cumplimiento a los objetivos específicos y al objetivo general propuesto para la investigación.

Recomendaciones

- Implementar la creación de interfaces utilizando las librerías ExtJS 2, Dojo, JQuery.

BIBLIOGRAFIA

1. Expósito, Carlos Marrero. 2008. Interfaz gráfica de usuario: Aproximación semiótica y cognitiva. Mentexpansiva. [En línea] 2008. [Citado el: 12 de Enero de 2014.] http://www.chr5.com/investigacion/investiga_igu/index_igu.html.
2. Moreno, Luciano. 2005. Componentes de una interfaz web. Desarrolloweb.com. [En línea] 2005. [Citado el: 10 de Diciembre de 2013.] http:// www.desarrolloweb.com/articulos/2171.php.
3. Houaiss, Antônio e outros (2001), Dicionário Houaiss da língua portuguesa. Objetiva, Rio de Janeiro.
4. Polis, Revista de la Universidad Bolivariana, Volumen 9, N° 26, 2010, p. 19-40.
5. Gordo, Marc Serra. 2007. Usabilidad de una biblioteca online con diseño centrado en el usuario. [En línea] 2007. http://openaccess.uoc.edu/webapps/o2/bitstream/10609/31061/6/mserragoTFC0707memoria.pdf.
6. Vromans, Johan. 2010. Vromans.ORG. GUI development with wxGlade. Johan Vromans. [En línea] 2010. [Citado el: 27 de Noviembre de 2013.] http://www.vromans.org/johan/articles/wxglade.pdf.
7. Artisteer. Sitio Web oficial de Artisteer. [En línea] 2008. [Citado el: 10 de Febrero de 2014.] http://www.artisteer.com.
8. What is Sencha Ext JS? sitio web de Sencha. [En línea] 2008. [Citado el: 1 de Diciembre de 2013.] http://www.sencha.com/products/extjs/.
9. Manuales. Sitio Web oficial de Qt. [En línea] 2013. [Citado el: 20 de Febrero de 2014.] http://qtproject.org/doc/qt-4.8/designer-manual.html.
10. Lobo, Armando Robert. 2012. 2012. Lycan-Génesis,Component Builder. Manual del Desarrollador. Centro DATEC. La Habana. Cuba: Facultad 6, Universidad de las Ciencias Informáticas., 2012.
11. Obregón, William González. 2013. Modelo de desarrollo de software v1.2. Centro de la Informatización de la Gestión de Entidades, Universidad de las Ciencias Informáticas. 2013.
12. Sommerville, Ian. 2005. Ingeniería de Software.7ma. 2005.
13. M, S Perez. Propuesta de modelo de desarrollo de software tecnológico del Centro de Soluciones de Gestión. Cuidad de la Habana: s.n., 2009.
14. Escribano, Gerardo Fernández. Introducción a Extreme Programing. 2002.
15. Manual de PHP. Cuidad Habana: s.n., 2011.
16. H,Ty.Manuales. [En línea][Citado: 2014-03-10] http://max-alva.webs.com/javascript.htm.

17. Visual Paradigm. Visual Paradigm for UML - UML tool for software application development. . [En línea] [Citado: 2014-03-10] http://www.visual-paradigm.com/product/vpuml/.
18. O Gomes Baryolo, Mariela Tenrero Cabrera, Nemuris Silega Martinez. Plantilla Registro de propiedad Intelectual (Sauxe). La Habana: s.n., 2008.
19. S. FRederick, Ramsay , Colin y Blades, Steves' Cutter. Learning EXT JS.
20. Secure Programming with Zend Framework. Esser, S. Ámsterdam: s.n., 2009.
21. Softonic. Cliente para Subversion para principiantes y expertos. Softonic. [En línea] 2010. [Citado el: 2 de Febrero de 2014.] http://rapidsvn.softonic.com.
22. Control de Versiones con Subversion y TortoiseSVN. ATICA. Universidad de Murcia. [En línea] 2008. [Citado el: 3 de Diciembre de 2013.] http://www.um.es/atica/documentos/PREsubversion.pdf .
23. Apache httpd 2.0.65 Released. The Apache HTTP server Project. [En línea] 09 de 07 de 2013. [Citado el: 10 de Diciembre de 2013.] http://httpd.apache.org/
24. Visual-Paradigm. Visual Paradigm for UML 8.3 Model-Code-Deploy Platform. [En línea] [Citado: 2014-03-10] http://www.visual-paradigm.com/product/vpuml/features/umlmodeling.jsp.
25. FOSS License Exception. MySql. [En línea] 2012. [Citado el: 5 de Diciembre de 2013.] http://www.mysql.com/about/legal/licensing/foss-exception
26. Sawyer, I.S y P. Requirements Engineering: A good practice guide.
27. Ivar Jacobson, Grady Booch, James Rumbaugh. El Proceso Unificado de Desarrollo de software. 2000.
28. María José Escalona, Nora Koch. Ingeniería de Requisitos en Aplicaciones para la Web – Un estudio comparativo. Sevilla: s.n., 2002.
29. Ingeniería de Requisitos en Aplicaciones para la Web. Un estudio comparativo. Sevilla: s.n., 2002.
30. Pressman, Roger S. 2005. Ingenería del Software. Un enfoque práctico. s.l.: Quinta Edición.McGraw Hil, 2005.
31. Little Boxes - Was Sie wissen sollten, bevor Sie eine Website bauen (lassen)", Peter Müller, 2008, Édition Markt + Technik, München/Germany
32. Créez votre site web en 2 heures chrono*, Micro Hebdo*, Couverture du n°541, 28 de agosto del 2008

33. Ganesh, Gunda Sai. 2008. Requirements Engineering: Elicitation Techniques. Suecia : Universidad del Este, Departamento de Tecnología, Matemática y Ciencia de la Computación, 2008. PR003.
34. Kincaid, Jason. (June 10, 2008). "Weebly Adds AdSense Support For Drag And Drop Cash." *TechCrunch*. Retrieved May 14, 2009.
35. Barros, Oscar. Reingeniería de Procesos de negocio. Chile: Editorial Dolmen, 1994.
36. Letelier, Patricio. Rational Unified Process (RUP). Rational Unified Process (RUP). Valencia: Universidad Politécnica de Valencia.
37. Microsoft. Revisiones de código y estándares de codificación. [En línea] [Citado el: 2014-03-06.] http://msdn.microsoft.com/es-es/library/aa291591%28v=vs.71%29.aspx.
38. Gamma, Erich, y otros. Design Patterns: Elements of Reusable Object -Oriented Software. s.l.: Addison- Wesley, 1994. 0- 201- 63361- 2.
39. Corporation, Rational Software. Glosario de Rational Unified Process. 2003.
40. Baryolo, Oiner Gómez. 2010. Solución informática de autorización en entornos multientidad y multisistema. Universidad de las Ciencias Informáticas. La Habana : s.n., 2010. Tesis de Maestría.
41. Camacho, Erika, Cardeso, Fabio y Nuñez, Gabriel. 2004. Arquitecturas de software. Guía de estudio. 2004.
42. Introducción a la Ingeniería de Requisitos. Ingeniería de software. . Estados Unidos: s.n., 2007.
43. Reynoso, B. 2004. Architect Academy: Seminario de Arquitectura de Software. 2004.
44. Buschmann, F, Kevlinn Henney, Douglas C. Schmidt. 2007. Pattern-Oriented Software Architecture: A pattern language for distribuited computing. Tennessee: J ohn Wiley & Sons. 2007. 978-0-470-05902-9.
45. ¿Qué es un Patrón de Diseño? [En línea] 2012. [Citado el: 27 de Febrero de 2014.] http://msdn.microsoft.com/es-es/library/bb972240.aspx.
46. Larman, Craig. 1999. 1999. UML y patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado. México: Alhambra S. A, 1999. ISBN: 970-17-0261-1. /8420534382.
47. Prieto, Félix. 2009. Patrones de diseño. . España: Departamento de Informática. Universidad de Valladolid, 2009.
48. Selección de metodologías de desarrollo para aplicaciones web. [En línea] 2009. http://www.eumed.net/libros/2009c/584/RUP%20Diseno%20e%20implementacion%20del%20sistema.htm.

49. Pérez Alfonso, Damián. 2012. Normas y estándares de codificación de Sauxe. La Habana,Cuba : Universidad de las Ciencias Informáticas, 2012.
50. Sperberg, Camilo. 2012. Sobre convenciones y notaciones (húngara, CamelCase, etc). unreal4u´s Personal Network. [En línea] 2012. [Citado el: 16 de Marzo de 2014.] http://blog.unreal4u.com/2011/03/sobre-convenciones-y-notaciones-hungara-camelcase-etc/.
51. Svensk, Magnus. 2012. DiVA. [En línea] 2012. [Citado el: 16 de Marzo de 2014.] http://urn.kb.se/resolve?urn=urn:nbn:se:hig:diva-12010.
52. Gonzalez, DH. Métricas en el desarrollo del Software.

Glosario de términos

GUI: Por sus siglas en inglés **G**raphics **U**ser **I**nterface

WYSIWYG: por sus siglas en inglés **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et. Traducción: lo que ves es lo que obtienes.

IDE: por sus siglas en inglés **I**ntregrated **D**evelopment **E**nvironment. Traducción: Entorno de desarrollo integrado.

Drag and Drop: término en inglés que significa arrastrar y soltar.

Theming: término del inglés cuyo espectro aborda el trabajo separado en temas de GUI.

CSS: por las siglas en inglés de **C**ascade **S**tyle **S**heet.

CMMI: por las siglas en inglés de **C**apability **M**aturity **M**odel **I**ntegration.

SEI: por las siglas en inglés de **S**oftware **E**ngineering **I**nstitute.

IEEE: por las siglas en inglés **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers. Traducción: Insituto de Ingenieros Eléctricos y Electrónicos.

GRASP: acrónimo de **G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns. Traducción: Patrones Generales de Software para Asignación de Responsabilidades.

GoF: acrónimo de **G**ang **o**f **F**our. Traducción: Banda de los Cuatro.

UML: por sus siglas en inglés **U**nified **M**odeling **L**anguage.

PHP: por sus siglas en inglés Hypertext Preprocessor.

HTML: por sus siglas en inglés **H**yper **T**ext **M**arkup **L**anguage.

CASE: por sus siglas en inglés **C**omputer **A**ssistant **S**oftware **E**ngineering.

DOM: por sus siglas en inglés **D**ocument **O**bject **M**odel.

AJAX: por sus siglas en inglés **A**synchronous **J**avaScript **A**nd **X**ML.

DHTML: por sus siglas en inglés **D**ynamic **HTML**.

PDF: por sus siglas en inglés **P**rofessional **D**ocument **F**ile.

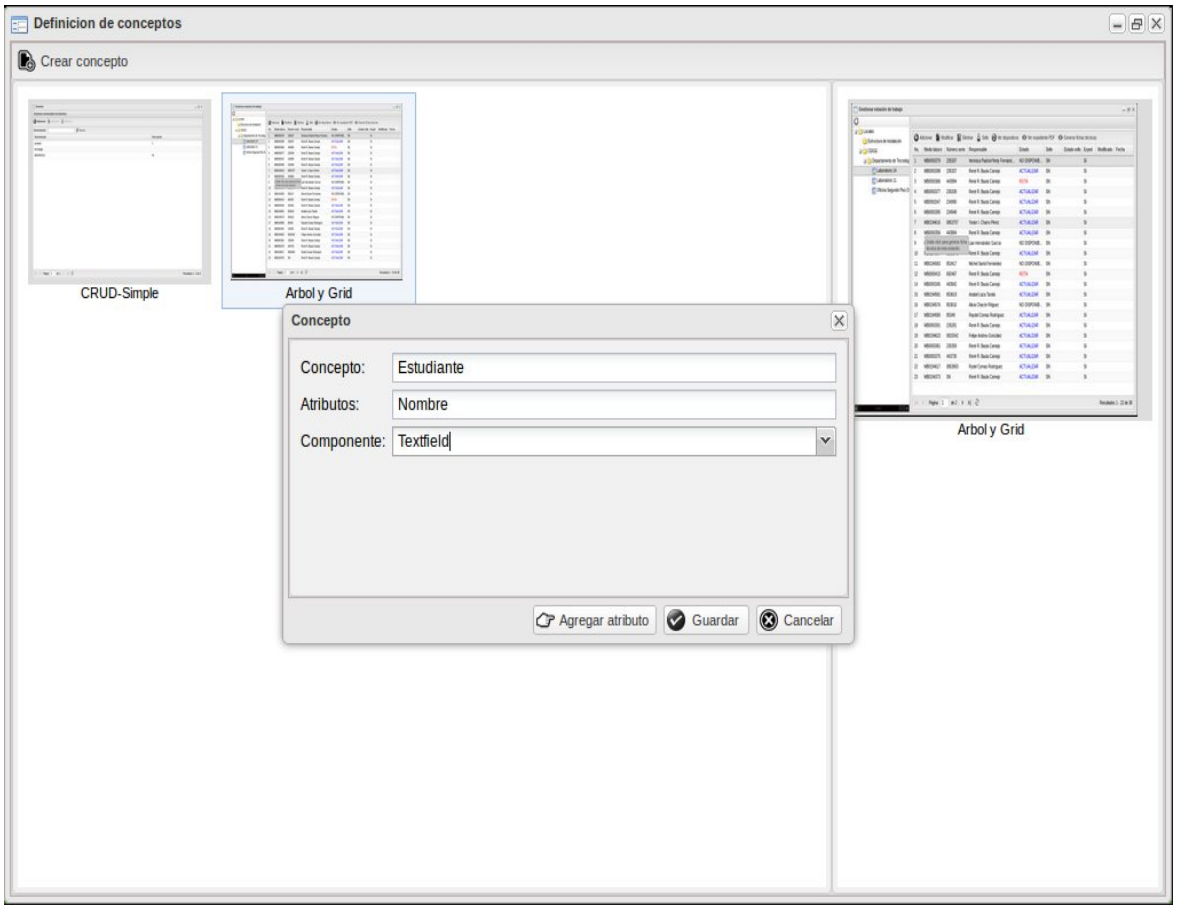
MVC: patrón arquitectónico **M**odelo **V**ista **C**ontrolador.

PDO: Protocolo de conexión a base de datos basado en objetos.

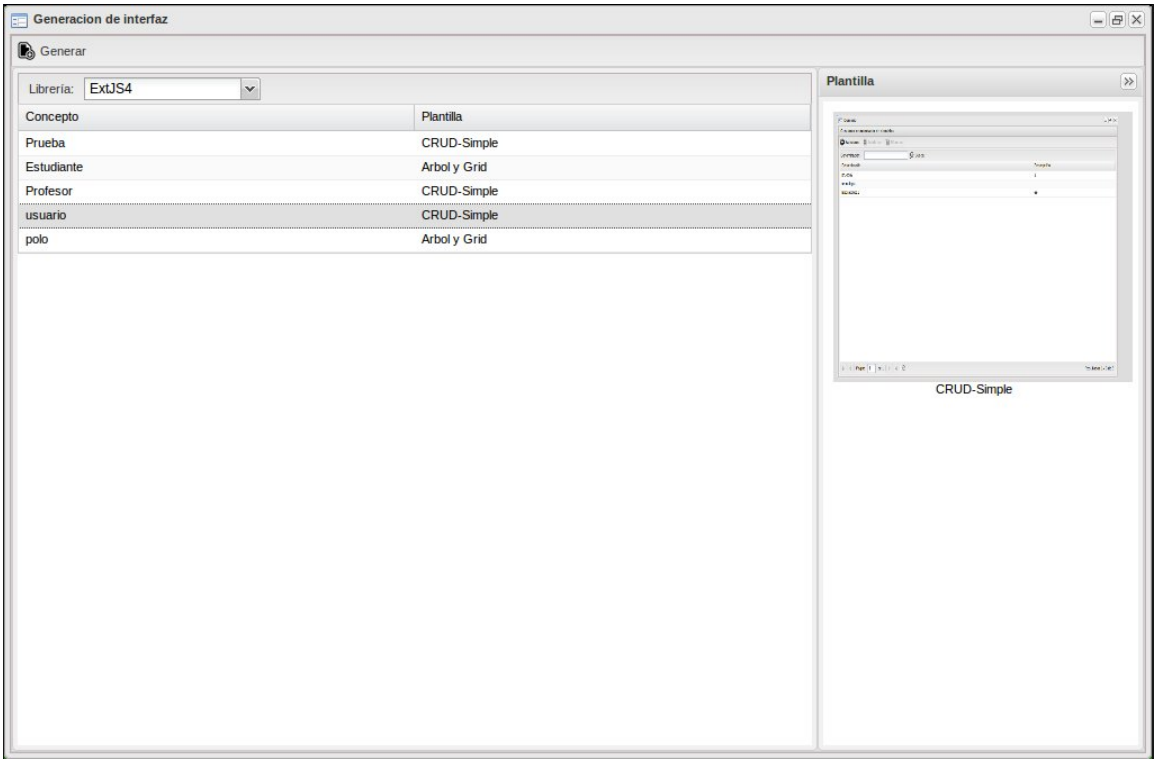
XML: Siglas en inglés de e**X**tensible **M**arkup **L**anguage ('lenguaje de marcas extensible').

ANEXOS

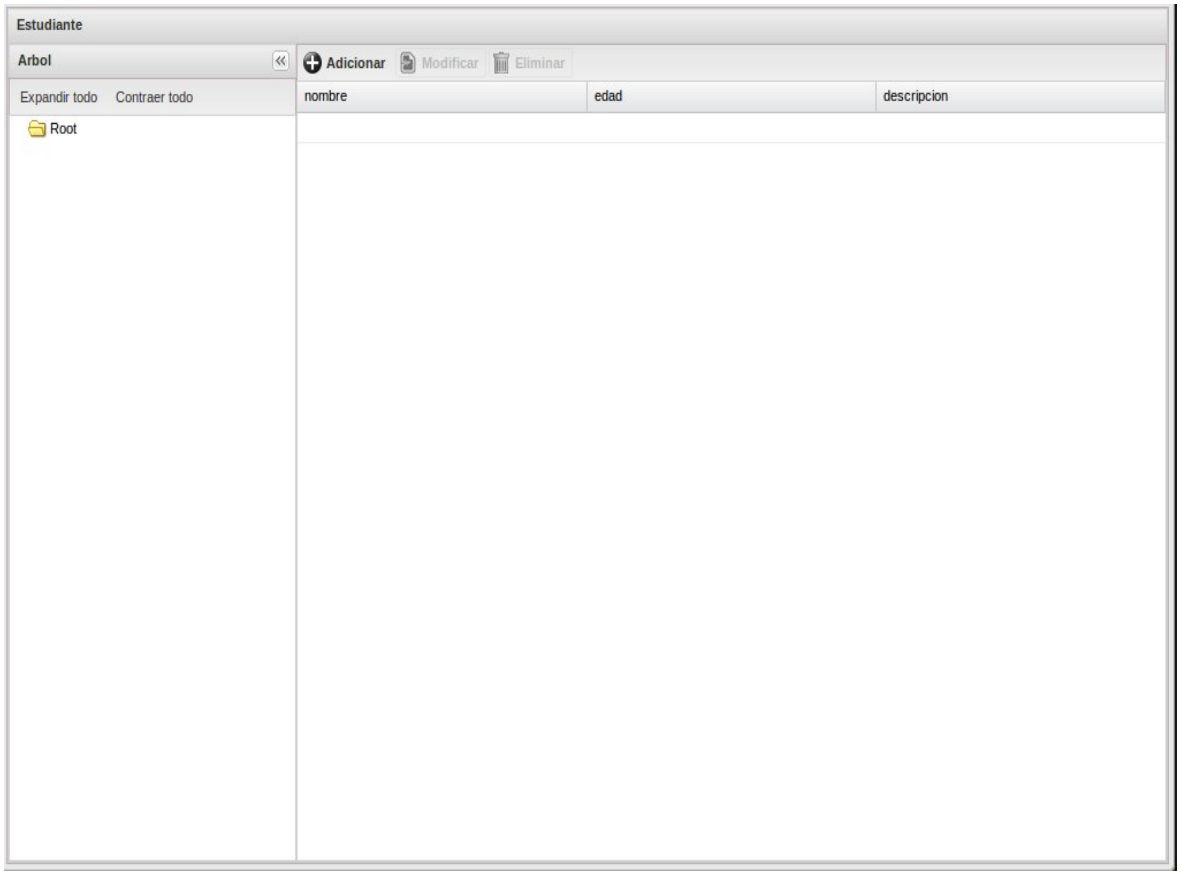
Anexo 1 Vista del componente Definición de conceptos



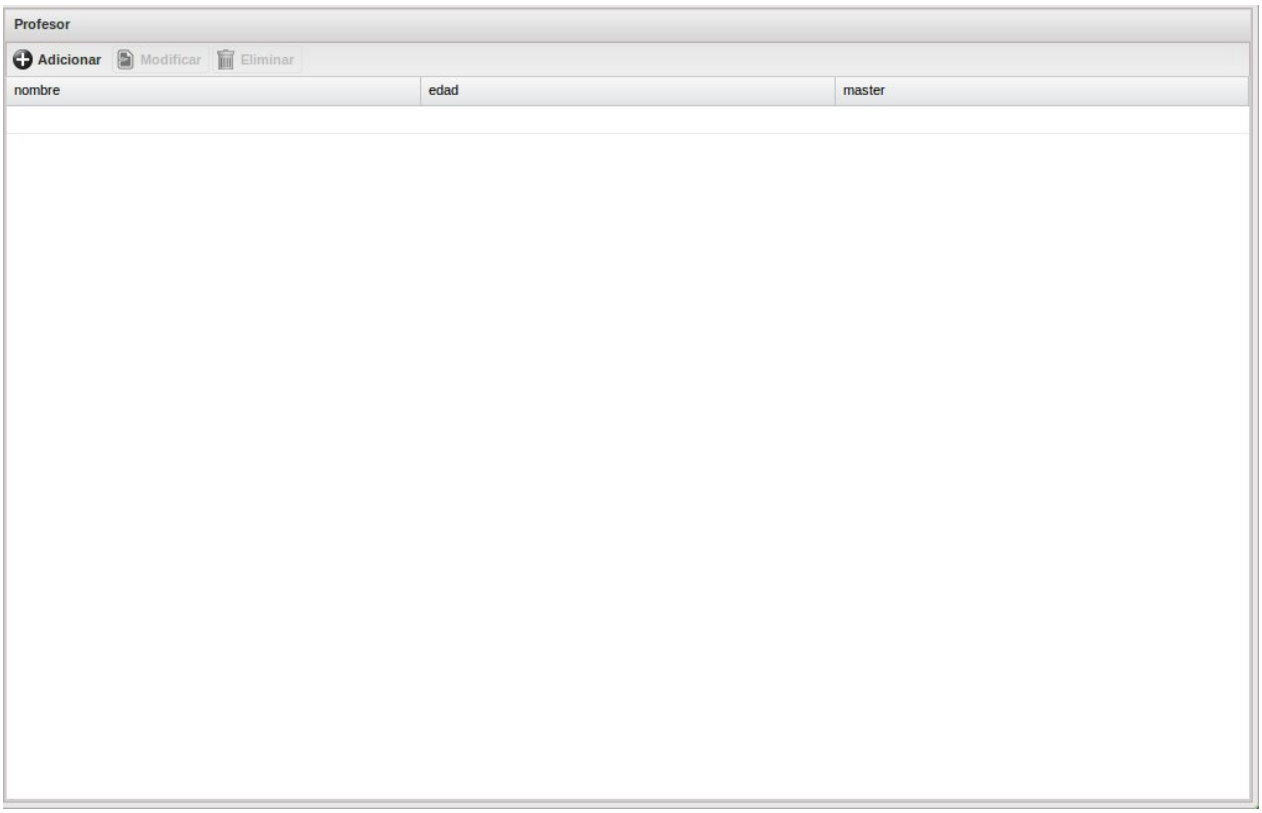
Anexo 2 Vista del componente Generación de Interfaz



Anexo 3 Interfaz generada utilizando la plantilla Arbol y Grid



Anexo 4 Interfaz generada utilizando la plantilla CRUD-Simple



Anexo 5 Datos almacenados en el XML

```
<?xml version="1.0" encoding="UTF-8"?>
<conceptos>
        <concepto>
                <denominacion>facultades</denominacion>
                <plantilla>CRUD-Simple</plantilla>
                <urlG>cargarFacultades</urlG>
                <atributos>
                        <atributo>
                                <denominacion>idFacultad</denominacion>
                                <componente>Textfield</componente>
                        </atributo>
                        <atributo>
                                <denominacion>descripcion</denominacion>
                                <componente>Textfield</componente>
                        </atributo>
                </atributos>
        </concepto>
</conceptos>
```