Universidad de las Ciencias Informáticas

Facultad 15

"Desarrollo del subsistema de Estructura y Composición versión 2.0 perteneciente al Sistema de Gestión de Entidades Cedrux"

Trabajo de Diploma para optar por el título de Ingeniero Informático.

Autores:

Miguel González Ortega. Lázaro Enrique Domínguez Suárez.

Tutores: Ing. René R. Bauta Camejo.

Ing. Osmar Leyet Fernández.

Junio, 2010

"Año del 52 Aniversario del Triunfo de la Revolución"

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Miguel González Ortega.

Lázaro Enrique Domínguez Suárez.

_____ _____

Firma del Autor (es) Firma del Tutor (es)

Datos de contacto

Ing. René Rodrigo Bauta Camejo: Ingeniero en Ciencias Informáticas, graduado en el año 2009. Desarrollador de la Subdirección de Tecnología del Centro de Informatización de la Gestión de Entidades (CEIGE).

Ing. Osmar Leyet Fernández: Ingeniero en Ciencias Informáticas, graduado en el año 2008. Fue miembro del equipo de implementación del sistema Saren, del proyecto Registros y Notarías. Luego se desempeñó en el rol de analista principal del sistema de Comercio electrónico B2B para la empresa Cubalse de la república de Cuba. Posteriormente ocupó el rol de Arquitecto de Tecnología del sistema Cedrux, del proyecto ERP Cubano y se convirtió en jefe de la línea de Contabilidad y Finanzas. En los momentos actuales forma parte de la Subdirección de Tecnología del (CEIGE) donde se desempeña como jefe del equipo de Arquitectura de Sistema y Datos.

Resumen

El presente trabajo se realizó con el objetivo de desarrollar una nueva versión la cual cuente con un conjunto de funcionalidades, relacionadas con la gestión de subordinaciones las cuales permitirán a las entidades organizar más específica y detalladamente su estructura organizativa, acercando más los procesos del sistema, a los procesos de la vida real y de esta forma eliminar las deficiencias que posee la versión anterior, garantizando a partir del correcto rediseño e implementación de algunas funcionalidades necesarias, la satisfacción, comodidad y conformidad de los usuarios que lo utilizarán posteriormente. Para ello se realizó el modelado, la especificación y descripción de las nuevas funcionalidades, así como el nuevo diseño propuesto para el mismo, y finalmente se llevó a cabo la implementación y pruebas para medir la calidad del diseño, la usabilidad del subsistema y la calidad del código implementado.

Esta versión tendrá un gran impacto pues no solo permitirá a los usuarios definir y gestionar la estructura física de las entidades, sino que además, se podrá manejar, visualizar de manera sencilla y confortable, la forma organizativa de las mismas mediante los distintos tipos de relaciones existentes dentro de ellas, manipulando el contenido lo más real y lógico posible.

Palabras Claves: compartir**,** estructurar**,** integrar, organizar, usabilidad.

Tabla de contenidos

Introducción *1*

Capítulo 1: Base conceptual de la investigación *4*

Introducción **4**

1.1 Estudio de la solución existente **4**

1.1.1 Gestión de subordinación de estructuras 6

1.2 Estructuras dinámicas de datos **10**

1.2.1 Estructuras de datos lineales 11

1.2.2 Estructuras de datos no lineales 13

1.3 Metodologías de desarrollo **17**

1.4 Metodología propuesta para el sistema **18**

1.5 Lenguajes de modelado **18**

1.6 Herramientas de desarrollo y tecnologías **19**

1.6.1 Marcos de trabajo 19

1.6.2 Herramientas de Modelado 20

1.6.3 Herramientas y Tecnologías de Desarrollo 20

1.6.4 Lenguajes de Programación 20

1.7 Especificaciones de la arquitectura **21**

1.7.1 Arquitectura Cliente/Servidor 21

1.8 Modelo conceptual ..... ¡Error! Marcador no definido.

1.9 Requisitos **21**

1.10 Diseño **22**

1.11 Patrones **22**

1.11.1 Singleton 22

1.11.2 Patrón Modelo-Vista-Controlador (MVC) 22

1.12 Prueba **23**

1.12.1 Pruebas de caja blanca 23

1.12.2 Métricas de Software 24

Conclusiones del capítulo 1 ..... **24**

Capítulo 2: Descripción de la solución del sistema *26*

Introducción **26**

2.1 Propuesta de la versión **26**

2.1.1 Múltiples Subordinaciones 26

2.2 Modelo conceptual ..... ¡Error! Marcador no definido.

2.3 Requisitos funcionales 29

2.4 Descripción de los nuevos requisitos 29

2.4.1 **Adicionar subordinación** 29
2.4.2 **Activar subordinación** 29
2.4.3 **Listar subordinación** 29
2.4.4 **Ordenar subordinación** 29
2.4.5 **Consultar subordinación** 30
2.4.6 **Buscar subordinación** 30

2.5 Diseño de la solución 30

2.5.1 Diseño de interfaces de usuario 30
2.5.2 Prototipo interfaz de usuario gestionar subordinación (interfaz principal) 30
2.5.3 Prototipo Interfaz de Usuario R1 Adicionar Subordinación 31
2.5.4 Prototipo Interfaz de Usuario R2 Activar Subordinación 32
2.5.5 Modelo de diseño 33
2.5.6 Diagrama de Clases del Diseño 34
2.5.7 Aplicación de patrones del diseño 34

2.6 Modelado de la base de datos 35

2.6.1 Modelo de Datos 36
2.6.2 Descripción del modelo de datos 37
2.6.3 Diagrama de clases 38

2.7 Implementación de la aplicación 38

2.7.1 Patrón Singleton 39
2.7.2 Patrón MVC 39
2.7.3 Nomenclatura 40
2.7.4 Normas de los comentarios 45
2.7.5 Tratamiento de errores 46

Conclusiones del capítulo 2 47

Capítulo 3: Validación de la solución propuesta 48

Introducción 48

3.1 Métricas de software 48

3.1.1 Resultados del instrumento de evaluación de la métrica Tamaño operacional de la clase (**TOC**) 49
3.1.2 Resultados del instrumento de evaluación de la métrica Relaciones entre Clases (RC) 51
3.1.3 Métricas de Usabilidad 54
3.1.4 Herramienta Plantilla empleada para medir las variables de la métrica 55
3.1.5 Resultados de la métrica 55
3.1.6 Comparando los datos con el diseño anterior 59
3.1.7 Prueba de caja blanca 60

Conclusiones del Capítulo 3 68

Conclusiones generales 70

Tabla de contenidos

Recomendaciones ***71***
Bibliografía ***72***

Introducción

Los Sistemas de Planificación de Recursos Empresariales (ERP), son los encargados de gestionar todos los recursos de una empresa, contribuyendo a la mejora de los procesos fundamentales que en ella se llevan a cabo. Un software ERP planea y automatiza muchos procesos con la meta de integrar información a lo largo de la empresa y elimina los complejos enlaces entre los sistemas de las diferentes áreas del negocio. La utilización de un ERP conlleva a la eliminación de barreras interdepartamentales, de esta forma, la información fluye por toda la empresa eliminando la improvisación por falta de información, lo que hace que estos sistemas adquieran mayor relevancia en la actualidad, donde las tecnologías de la información y las comunicaciones son protagonistas en todas las esferas de la vida.

La introducción de un sistema ERP es una necesidad, y más aún en un país como Cuba donde la planificación de los recursos es esencial para la toma de decisiones y además, la organización estructural de las entidades se ve afectada al no existir uniformidad y compatibilidad, incluso dentro de ella misma. Igualmente su forma organizativa no refleja una estructura concreta, poniéndose de manifiesto el desconocimiento por parte del personal de conceptos fundamentales que garantizan la organización y el control trayendo consigo que las barreras interdepartamentales sean más agudas y el flujo de información se haga ineficiente con el paso del tiempo. Por la necesidad de contar con un ERP para estandarizar estos aspectos y para evitar el pago excesivo de licencias de software, se le propone a la Universidad de las Ciencias Informáticas (UCI) desarrollar el primer ERP cubano denominado **Cedrux**.

Cedrux es un paquete de soluciones integrales de gestión de entidades capaz de adaptarse a las características económicas del país y sus empresas. Este producto pone al servicio de las entidades facilidades para la integración de las diferentes áreas productivas y departamentos administrativos. El mismo cuenta con varios subsistemas como Planificación, Seguridad, Configuración, Capital Humano, Estructura y Composición, entre otros. Este último es la columna de todo el ERP cubano pues es quien modela la estructura organizativa de una empresa brindando una gran flexibilidad a la hora de realizar esta operación, ya que posee la capacidad de soportar una estructura que contenga muchas entidades adaptándose a las especificidades de cada una. Estructura y Composición en su versión 1.0 es capaz de gestionar la estructura organizativa de las entidades y agruparlas. Mediante éste se crean las entidades y

su nivel de subordinación física, donde luego el resto de los subsistemas se encargan de desarrollar los procesos especializados para algunas de estas entidades que fueron previamente creadas.

Además, esta versión permite al usuario gestionar las subordinaciones lógicas a partir de la estructura organizativa antes creada, el mismo mediante una interfaz puede asignar una subordinación a una determinada estructura. Esta versión presenta una serie de deficiencias en el diseño y la implementación, las cuales no satisfacen las expectativas de los usuarios que la utilizan, conllevando a que el subsistema presente problemas entre los cuales se pueden citar la manera engorrosa de asignar subordinaciones, ya que cuando se tienen un número considerable de estructuras se muestra una interfaz la misma cantidad de veces que se realice esa operación.

Otro aspecto que afecta la interacción del cliente con el subsistema es la representación de las subordinaciones con colores según su tipo de relación, por lo que si se cuenta con un número considerable de tipos la escala de colores aumenta y el cliente tendrá que memorizar la relación color-tipo o de otra manera consultar constantemente una guía.

La información que se muestra al usuario es escasa lo cual puede traer como consecuencia que se haga difícil la búsqueda de subordinaciones.

Otra de las deficiencias detectadas es que las estructuras pueden repetirse tantas veces sean estas movidas en el mismo árbol de subordinaciones, lo cual resulta ilógico a la hora de realizar este proceso, ya que en la vida real, no pueden estar subordinadas a ellas mismas, como tampoco puede existir más de una subordinación activa de un mismo tipo.

Además no se restringe que se pueda subordinar una estructura a otra con un tipo de subordinación solamente.

La versión 1.0 de este subsistema carece de las funcionalidades necesarias, donde se puedan guardar y recuperar los historiales de las subordinaciones, existiendo así un diseño para la persistencia de datos bastante ineficiente.

Por lo antes descrito se hace necesario desarrollar una nueva versión del subsistema Estructura y Composición que elimine las deficiencias mencionadas anteriormente y que además permita al usuario una mejor interacción con el subsistema facilitándole de manera más sencilla, y detallada la gestión de

subordinaciones y la obtención de información de una forma más fluida y amigable, a través de un nuevo conjunto de funcionalidades que se incluirán en la misma.

De acuerdo con la situación problemática antes descrita, se define el **problema de la investigación** como sigue:

La inexistencia de un conjunto de funcionalidades en el subsistema de Estructura y Composición, está afectando la usabilidad del mismo, dentro del Sistema de Gestión de Entidades Cedrux.

Objeto de Estudio

Estructuras Dinámicas de Datos.

Campo de Acción

Estructuras dinámicas de datos para el subsistema Estructura y Composición.

La investigación cuenta con la siguiente **idea a defender:**

Si se implementan las nuevas funcionalidades y se incorporan al subsistema Estructura y Composición, se garantizará una mayor usabilidad dentro del Sistema de Gestión de Entidades Cedrux.

Para dar solución al problema de la investigación se trazó el siguiente **Objetivo General**:

Incorporar nuevas funcionalidades al subsistema Estructura y Composición, para garantizar una mayor usabilidad del mismo, dentro del Sistema de Gestión de Entidades Cedrux.

Del objetivo general se desglosan los siguientes **Objetivos Específicos:**

1. Determinar el marco teórico de la investigación.
2. Identificar y especificar las funcionalidades a incorporar en el sistema.
3. Diseñar las nuevas funcionalidades necesarias a incorporar al sistema.
4. Implementar el diseño propuesto para el sistema.
5. Validar los resultados obtenidos para medir la variable usabilidad.

Capítulo 1: Base conceptual de la investigación

Introducción

En este capítulo se realizará un breve recuento de la situación actual del subsistema Estructura y Composición. Para eso fue necesario estudiar y conocer el funcionamiento del mismo, qué estructuras de datos y patrones se podrían utilizar, y determinar las herramientas, la metodología y arquitectura a emplear en el mismo, así como las pruebas que más adelante se realizarán, para verificar que se están cumpliendo con los requerimientos identificados para la implementación de las nuevas funcionalidades que se incluirán al subsistema, logrando con esto sentar las bases teóricas para desarrollar el trabajo.

1.1 Estudio de la solución existente

El subsistema Estructura y Composición presenta cuatro responsabilidades fundamentales:

- Brinda la posibilidad al usuario de definir la estructura organizativa de una entidad y la estructura dentro de la misma, así como permitir que se puedan especificar los cargos por los cuales van a estar compuestas las diferentes áreas dentro de las unidades y los medios asociados a dichas áreas.
- Presta servicios al resto de los subsistemas presentes en el Sistema Cedrux donde la mayoría de estos necesitan algún servicio de Estructura y Composición.
- Es configurable para que los usuarios no estén obligados a usar los mismos conceptos estructurales.
- Brinda la posibilidad de asignar subordinaciones a las estructuras conformadas por entidades.

Además el subsistema cuenta con numerosas funcionalidades, algunas de ellas, las más importantes agrupadas dentro de los procesos siguientes:

- **Crear Estructura:** En este proceso se crean las estructuras necesarias, para crear una estructura es necesario un documento que contiene las especificaciones de la estructura, todos los datos y

requisitos que debe cumplir. Y después de creada se conforma un documento con las características de la estructura y sus funcionalidades.

Objetivo del Proceso: Crear una nueva estructura.

- **Modificar Estructura:** Permite modificar las estructuras ya existentes, para ello se toma el documento existente de la situación de la estructura y después de modificada, se conforma el documento con todos los cambios realizados.

  Objetivo del Proceso: Modificar una estructura.

- **Eliminar Estructura:** Permite eliminar una estructura ya existente, para hacer este proceso es necesario un documento legal que justifique esta eliminación, y después se conforma el documento que contiene toda la información de por qué fue eliminada la estructura.

  Objetivo del Proceso: Eliminar una estructura determinada.

- **Crear Estructura Interna:** Permite crear una estructura interna, para crear la estructura interna es necesario un documento que contiene las especificaciones de la misma, todos los datos y requisitos que debe cumplir. Después de creada se conforma un documento con las características de la estructura y sus funcionalidades.

  Objetivo del Proceso: Prever la creación de: Organigrama de la Unidad, Plantilla de Cargos y Ocupaciones y Plantilla de Medios.

- **Modificar Estructura Interna:** Permite modificar una estructura interna ya existente, se toma el documento existente de la situación de la estructura y después de modificada la misma, se conforma el documento con todos los cambios realizados.

  Objetivos del Proceso: Prever la creación de: Organigrama de la Unidad, Plantilla de cargos y Ocupaciones, Plantilla de Medios.

- **Eliminar Estructura Interna**: Permite eliminar una estructura interna, para hacer este proceso es necesario un documento legal que justifique esta eliminación, y después se conforma el documento que contiene toda la información del por qué fue eliminada la estructura interna.

Objetivos del Proceso: Prever la eliminación de: Organigrama de la Unidad, Plantilla de cargos y Ocupaciones, Plantilla de medios.

También posee otras importantes funcionalidades necesarias para realizar las anteriormente mencionadas como Gestionar Nomencladores el cual permite al usuario gestionar los elementos o estructuras que luego se utilizarán para realizar la estructura general organizativa de la entidad en donde el software esté implantado, por ejemplo los Órganos, Especialidades, Cargos y Categorías que pueda tener o no una entidad. También permite al usuario, visualizar varios tipos de reportes de entidades determinadas mediante un Gestor de Reportes.

Luego de haber abarcado un estudio de cómo se realiza la gestión de estructuras dinámicas en la versión anterior, para un mejor entendimiento de este proceso y así, partiendo de los conocimientos adquiridos del mismo adentrar el desarrollo del trabajo a la gestión de subordinaciones con el objetivo de aplicarlos para mejorar la calidad del subsistema y cumplir con los objetivos trazados, y de esta forma garantizar el aumento y mejora de la usabilidad y funcionalidad de la nueva versión. A continuación se dará a conocer la importancia de la gestión de subordinación de estructuras, así como se abordará más detalladamente acerca de los problemas que afectan el trabajo con las mismas.

1.1.1 Gestión de subordinación de estructuras

Antes de que el subsistema Estructura y Composición permitiera gestionar las subordinaciones, todos los procesos que se realizaban en éste se hacían a partir del árbol de estructuras físicas, es decir, cuando se creaban los niveles, agrupaciones, entidades, cargos y áreas dentro de estas, a esa estructura física, se le creaban los procesos especializados para cada uno de estas estructuras dentro de la misma. Pero, ¿por qué la decisión de agrupar dichas estructuras en subordinaciones? ¿Por qué no realizar estos procesos a partir del árbol físico de estructuras? No debería ser pues, al crear una estructura organizativa, no hay forma de ver la relación que existe entre cada una de las estructuras que se crearon, o sea, la forma de organizar de esta estructura se pudo haber realizado pensando en una jerarquía económica, pero también se pudo haber hecho pensando en una relación jerárquica de tipo política o administrativa, o incluso sin ningún orden especifico o sin un tipo de relación de dependencia entre ellas, de manera que a la hora de realizar un proceso para una de estas entidades, no se tiene el conocimiento suficiente de qué relación existe entre la misma y la entidad a la cual esta se subordina, por tanto, este proceso se llevaría a cabo

sin saber las consecuencias que pudiera traer, el desconocimiento de este elemento tan importante y que en la versión 1.0 ya se comienza a tener en cuenta. El mismo en la actualidad comenzó a utilizarse en diferentes entidades del país, incluyendo la Universidad de Ciencias Informáticas, pero la forma de gestionar las subordinaciones se torna muy poco usable, ya que presenta un sinnúmero de problemas que frenan la interacción deseada con los usuarios, causando desagrado e insatisfacción a la hora de trabajar con el mismo.

Los problemas que se evidencian a simple vista, o sea, en la interfaz, (sin entrar aún en detalles del diseño o de la implementación de esta versión) son que, cada vez que se seleccione y arrastre una estructura hacia otra con el objetivo de asignarle una subordinación y su respectivo tipo, se mostrará la interfaz “Datos” al usuario, por ejemplo, si se tienen cien estructuras y se quiere asignar una subordinación a cada una de ellas, esta interfaz se mostraría la misma cantidad de veces. Esto resulta sumamente tedioso y engorroso, por lo que se ha considerado uno de los puntos a rediseñar en la nueva versión. Además, estas subordinaciones una vez creadas, son representadas según su tipo, con un color, identificando de esta manera la relación existente (Política, Económica, etc.), lo que sería muy difícil para el usuario recordar a partir del color, la subordinación que ha creado. Tampoco existen campos que muestren información acerca de la misma (exceptuando el tipo que como se dijo anteriormente se representa a través de un determinado color), ya que no permite la inserción de información adicional, como puede ser una descripción, o la hora de creación y de desactivación en caso de que haya sido sustituida por otra. Al igual vale destacar que si la estructura no es eliminada de la subordinación a la que pertenece, esta se clona tantas veces sea movida hacia la nueva, lo cual no resulta muy funcional, ya que podría estar subordinada a ella misma con o sin el mismo tipo de relación, además que en la misma estructura organizativa, la susodicha se podría repetir con diferente composición y diferente relación, y no solo ella sino también las estructuras que conforman dicha subordinación, lo que resulta algo ilógico a la hora de realizar este proceso. En conclusión, resulta casi imposible para el usuario interactuar con la aplicación a la hora de gestionar las subordinaciones entre estructuras, haciéndosele muy difícil, identificar, manejar y visualizar el contenido que se maneja en la interfaz del subsistema.

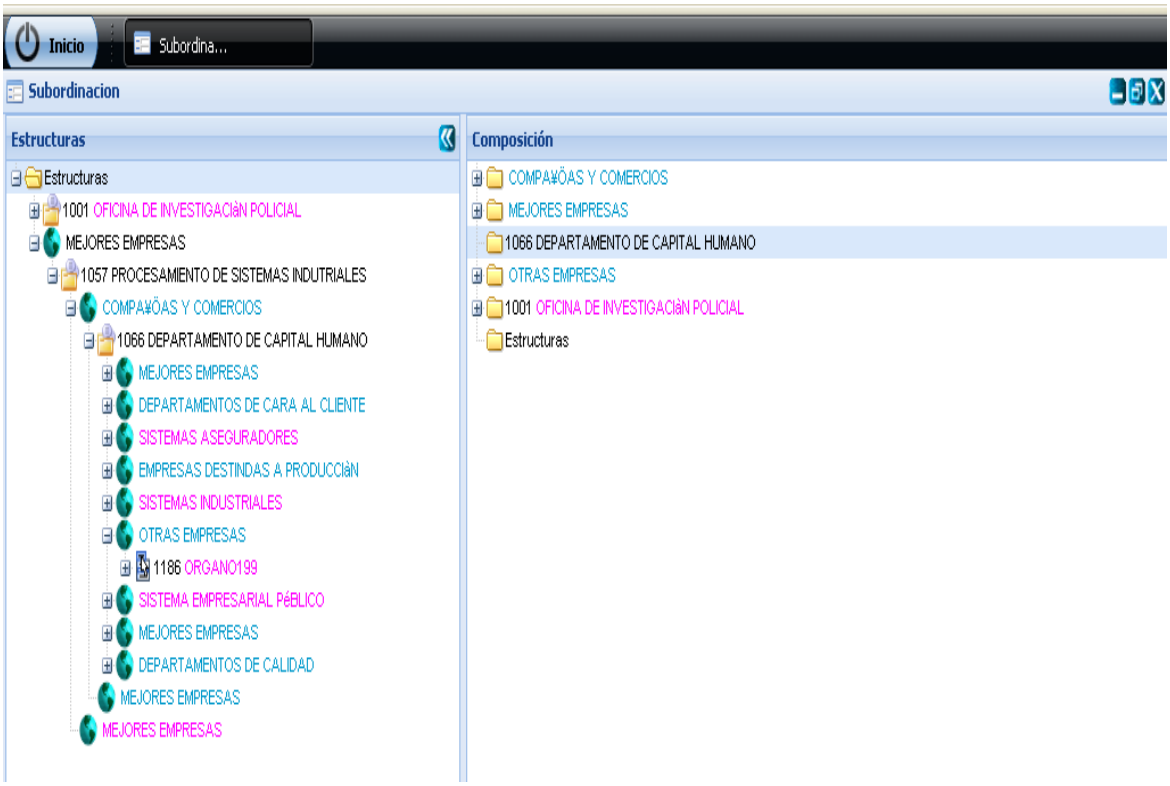


Fig. 1 Interfaz donde se gestionan las subordinaciones en Estructura y Composición v1.0

En la figura 1 se muestra la interfaz de usuario Subordinaciones perteneciente a la versión actual de Estructura y Composición. En el panel izquierdo de muestra el árbol de subordinaciones donde los colores representan las relaciones o jerarquías entre dichas estructuras. En el panel derecho se muestra la composición de dichas estructuras. Como se puede apreciar luego de observar esta imagen se puede ver que no muestra mucha información al usuario además que este árbol de subordinaciones presenta problemas de conceptos ya que permite subordinaciones recursivas entre las estructuras y las estructuras internas de estas y que anteriormente se explicó con más detalle.

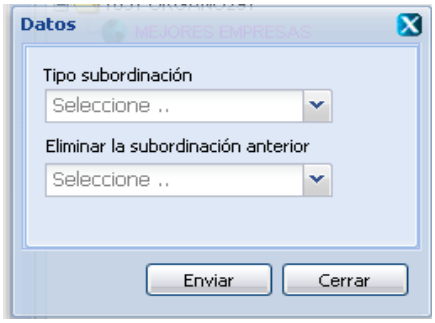


Fig. 2 Interfaz de usuario Datos

Esta es la interfaz que se obtiene como resultado de arrastrar una estructura dentro de otra para asignarle un tipo de subordinación a la misma. Como se explicó anteriormente esta interfaz muestra poca información al usuario y cada vez que se desee asignar a una estructura una subordinación se mostrará la misma, resultando tedioso el trabajo con las subordinaciones, lo que concluye a realizar mejoras en el diseño e implementación de esta versión.

Por otra parte, el modelo de datos no restringe, debido a la relación entre sus llaves primarias, que en la misma organización estructural exista una entidad con un solo tipo de subordinación, contribuyendo a que una determinada estructura se pueda subordinar a ella misma, con o sin el mismo tipo de subordinación, lo que resulta irreal e ilógico. Además la inexistencia y mal desarrollo de otros elementos, entre los que se encuentran atributos y funcionalidades mal implementadas y otras inexistentes, tanto en la vista como en la lógica, traen consigo todos los problemas que se manifiestan en la versión actual, antes mencionados. Para un mejor entendimiento de esto, se muestra a continuación, el modelo de datos de la versión 1.0 de la solución anterior para las múltiples subordinaciones.

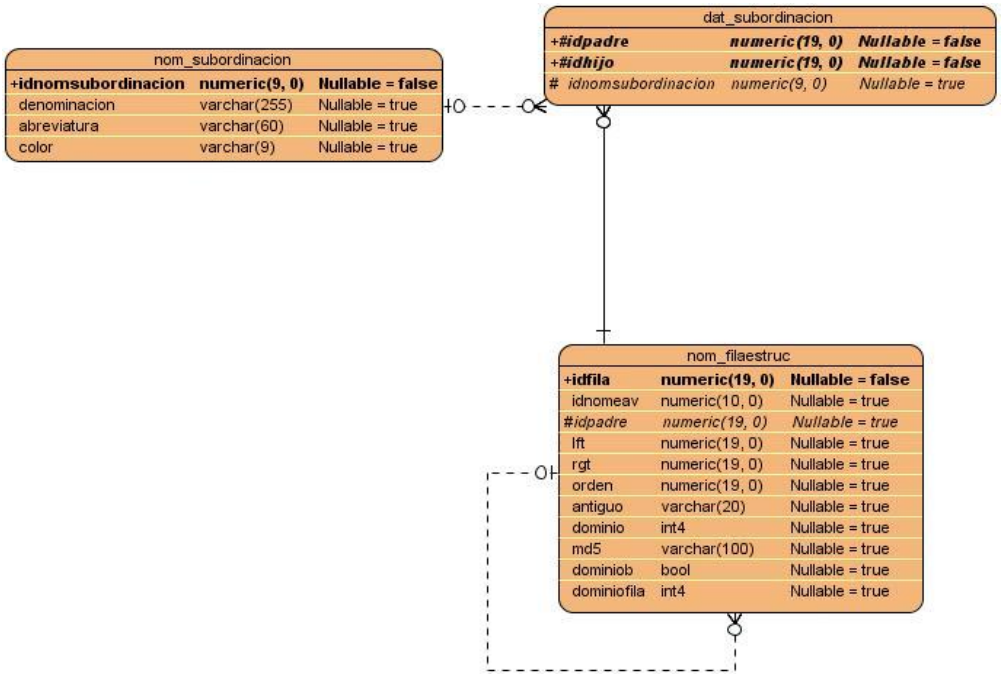


Fig. 3 Diagrama Entidad Relación de Estructura y Composición v1.0

Luego de haber visto algunos aspectos importantes de la versión en uso de Estructura y Composición, se comienza el estudio pertinente a los elementos que se usarán para el desarrollo de la nueva versión. Este análisis permitirá darle paso a la investigación, con la meta de crear todos los artefactos necesarios durante todo el ciclo de desarrollo de la misma.

1.2 Estructuras dinámicas de datos

Las estructuras dinámicas de datos son de gran utilidad para almacenar datos del mundo real, que están cambiando constantemente. Son estructuras cuya dimensión puede crecer o disminuir durante la ejecución del programa. Las mismas presentan una colección de elementos denominados nodos. Al contrario de un array (arreglo), que contiene espacio para almacenar un número fijo de elementos, una estructura dinámica de datos se amplía y contrae durante la ejecución del programa. Estas permiten gran facilidad de adaptación a las necesidades reales a las que suelen enfrentarse los programas. Pero no sólo eso, también presentan una flexibilidad la cual representa una ventaja para la creación de aplicaciones

que manipulan datos dinámicos y cambiantes , ya sea en cuanto al orden, la estructura interna o las relaciones entre los elementos que las componen.

Las **estructuras de datos dinámicos** consisten en una agrupación lógica de elementos, cada uno de los cuales, es a su vez, o bien un dato, u otra estructura de datos que a su vez contiene datos, además el número de elementos que contienen puede variar durante la ejecución del programa. Su principal inconveniente es la lentitud en el acceso, ya que normalmente se realiza de forma secuencial. La ventaja es sin embargo importante, la posibilidad de aumentar o disminuir en tiempo de ejecución el número de elementos que componen la estructura. (Korsh, y otros, 2004).

Las estructuras dinámicas de datos se pueden dividir en dos grandes grupos:

- **Lineales**: Listas enlazadas, pilas, colas.
- **No lineales**: árboles, grafos.

1.2.1 Estructuras de datos lineales

Las estructuras de datos lineales se caracterizan porque sus elementos están secuencialmente, relacionados en forma lineal, uno luego del otro. Cada elemento de la estructura puede estar conformado por uno o varios subelementos que pueden pertenecer a cualquier tipo de dato, pero que normalmente son tipos básicos. (Dpto. Central de Técnicas de Programación UCI, 2008).

1.2.1.1 Listas

Las listas aparecen frecuentemente en la solución computacional de muchos problemas, ya sea para almacenar objetos de determinada clase y calcular el máximo, mínimo, contar las ocurrencias de un elemento, etc. Una lista es un contenedor de objetos, la cual almacena cada elemento en una posición y mantiene esas posiciones dispuestas en orden lineal. Una lista es una colección lineal de elementos, también llamados nodos donde el orden de los mismos se establece de forma consecutiva, es decir un elemento precede al siguiente y así sucesivamente hasta la cantidad de elementos de la misma. También pueden ser usadas para implementar otras estructuras de datos como grafos, pilas y colas. Su definición formal plantea que:

Una lista se define como una n-tupla de elementos (donde li es el i-esímo elemento de la lista) ordenados de forma consecutiva, o sea, el elemento li precede al elemento li+1: L= (l1, l2, … , ln). (Dpto. Central de Técnicas de Programación UCI, 2008).

Si la lista contiene 0 elementos se denomina lista vacía.

```
Ejemplos:
    L = ( 5, 4, 3, 2, 1)
    M = ( "María", "Juan", "Pedro" )
    N = ( (1,2,3), (4,5,6), (7,8,9,10) )
    O = ( 5.10, 4.99, 3, 2, "1", (3,6,9) )
```

Fig. 1 Ejemplos de listas

1.2.1.2 Pilas

Una pila (stack en inglés) es una lista o estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (del inglés Last In First Out, último en entrar, primero en salir) que permite almacenar y recuperar datos. (Dpto. Central de Técnicas de Programación UCI, 2008). Algunos ejemplos de pila pueden ser:

Una pila de platos, una pila de monedas, una pila de bandejas.

Fig. 2 Ejemplo de pila

Se aplica en disímiles de ocasiones en informática debido a su simplicidad y ordenación implícita en la propia estructura.

1.2.1.3 Colas

Es una estructura de datos o lista lineal de elementos en la que las operaciones de insertar y eliminar se realizan en diferentes extremos de la cola. (Dpto. Central de Técnicas de Programación UCI, 2008).También se le llama estructura FIFO (del inglés First In First Out primero en entrar, primero en salir), pues trabajan con esa filosofía y se emplean mucho en informática principalmente en las relaciones de prioridad entre elementos. Algunos ejemplos de **Colas** son:

- Cola de automóviles esperando servicio en una gasolinera.
- Cola de clientes en una ventanilla del banco para pagar un servicio.
- Cola de programas en espera de ser ejecutados por una computadora.

Fig. 3 Representación de una cola

Algunos tipos de colas son:

- Cola simple: Estructura lineal donde los elementos salen en el mismo orden en que llegan.
- Cola circular: Representación lógica de una cola simple en un arreglo.
- Cola de Prioridades: Estructura lineal en la cual los elementos se insertan en cualquier posición de la cola y se remueven solamente por el frente.

1.2.2 Estructuras de datos no lineales

Las estructuras de datos no lineales se caracterizan por no existir una relación de adyacencia entre sus elementos, es decir, un elemento puede estar relacionado con cero, uno o más elementos. En Estructuras de Datos Lineales, como Listas, Pilas, Colas los elementos que forman la estructura de datos, son almacenados unos detrás de otro, secuencialmente. En estas estructuras cada elemento tiene un único

elemento anterior y un único elemento posterior, aunque mediante la combinación de algunas de estas estructuras se puede representar de otra forma y eliminar esta adyacencia creando más relaciones entre los elementos que la componen. Existen diversos problemas cuya solución no puede representarse mediante una disposición secuencial de elementos, por ejemplo, análisis de circuitos eléctricos, representación de fórmulas matemáticas, la estructura sintáctica de un programa fuente en los compiladores, el índice de un libro, el explorador de carpetas de un sistema determinado, etc. Estos problemas se resuelven utilizando Estructuras de Datos no Lineales, las cuales representan los datos en una manera mucho más cercana a la realidad. (Dpto. Central de Técnicas de Programación UCI, 2008).

1.2.2.1 Estructuras Arbóreas

Una de las Estructuras de Datos no Lineales más utilizadas es el Árbol. Las estructuras de árbol o estructuras arbóreas expresan una relación jerárquica entre los elementos de un conjunto dado. (Dpto. Central de Técnicas de Programación UCI, 2008). Un ejemplo de ello se muestra a continuación:

Por ejemplo, considérese el índice general de un libro, el cual se representa en la siguiente figura.

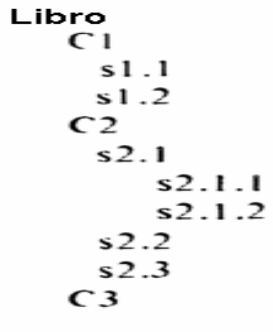


Fig. 4 Representación del índice de un libro

Tal índice es la representación de un árbol. Se puede redibujar en la forma mostrada en la figura que se muestra a continuación.

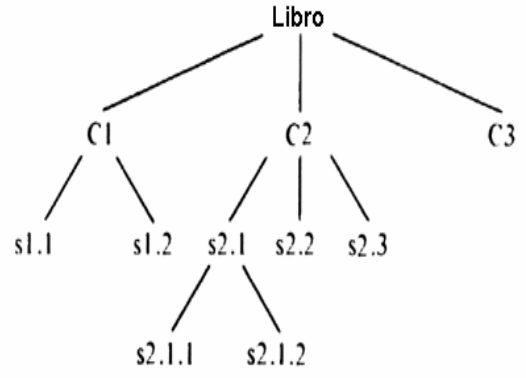


Fig. 5 Representación de un árbol

Otro ejemplo de la vida real, en la que los datos cumplen estas relaciones jerárquicas de que se ha hablado y por lo tanto, su forma más natural de representación es mediante estructuras arbóreas.

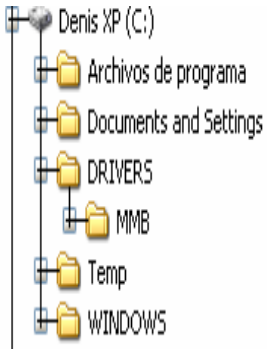


Fig. 6 Estructura de directorios en Windows

En algunos casos las estructuras arbóreas son la solución más natural para la organización de la información, siendo además muy eficiente en cuanto a tiempo computacional en algunas operaciones, algunos ejemplos de ello son:

- Búsqueda y ordenamiento. -- Codificación de la información. -- Análisis de la sintaxis de los lenguajes de programación. -- Representación del conocimiento y búsqueda de soluciones.

Las estructuras basadas en árboles solo pueden tener una sola relación: la de padre e hijo, es decir las nodos que sean padres de sus hijos siempre se comportarán de esa forma y no al revés en ningún otro

caso pues como bien dice su definición: ***"Un árbol es un grafo conexo, no dirigido y a-cíclico",*** no son factibles para el trabajo con varias subordinaciones. (Dpto. Central de Técnicas de Programación UCI, 2008). Los árboles sólo permiten almacenar elementos cuya única relación existente entre ellos es una relación jerárquica, por ejemplo "X es padre de Y". Esta relación tiene como limitante que no es simétrica, es decir, si "X es padre de Y" no se puede decir que "Y es padre de X".

En muchos problemas prácticos incluyendo el sistema **Cedrux**, donde se tiene un conjunto de elementos de un mismo tipo, puede suceder que esta no sea la única relación que exista entre ellos, esto quiere decir que incluso puede haber otras relaciones las cuales no necesariamente debe existir exactamente la misma dependencia jerárquica. Por ejemplo, si se tiene un problema donde se debe tratar un conjunto de personas, y se quiere estudiar cómo se comportan las relaciones de amistad entre dichas personas, entonces la relación en cuestión sería "**X** es amigo de **Y**". Para este problema, un árbol es una estructura insuficiente, pues no permite la ocurrencia de ciclos en su estructura, y por tanto, no permitiría tratar el hecho de que para tres personas cualesquiera, **Antonio** (A), **Pedro** (B) y **Carlos** (C), se pueda cumplir que **A** sea amigo de **B**, **B** sea amigo de **C** y **C** sea amigo de **A**, cuando son hechos que en la práctica pueden suceder (Fig. 7). Tampoco permite tratar el hecho de que **A** sea amigo de **B** y además **B** sea amigo de **A**, (Dpto. Central de Técnicas de Programación (UCI), 2010) por lo que para la solución general y el cumplimiento de los objetivos propuestos usar esta estructura de datos solamente, no sería lo más factible, aunque sí se emplearán en el desarrollo de la nueva versión.

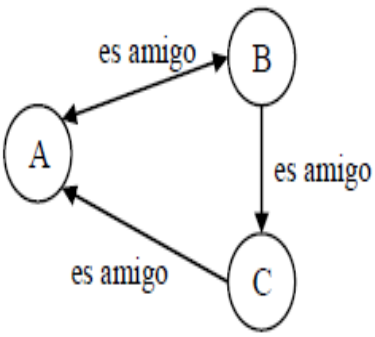


Fig. 7 Representación gráfica de la relación "X es amigo de Y" para tres personas A, B y C

1.2.2.2 Grafos

La estructura no lineal de datos de uso más general es el grafo donde sus nodos pueden relacionarse de cualquier manera sin una relación de orden predefinida. Al contrario de las estructuras basadas en árboles, las ED grafos son el mecanismo más general para la interrelación entre objetos o nodos. (Dpto. Central de Técnicas de Programación (UCI), 2010). Intuitivamente se puede decir que un grafo es un conjunto de puntos que pueden estar conectados entre sí dos a dos mediante aristas. Los puntos representarían los elementos del problema (estructuras), y las aristas representarían la existencia o no de la relación. (En la Fig. 7, si A es amigo de B, entonces existirá una arista que conecte al punto que representa a **A** con el punto que representa a **B**). A continuación se presenta una definición de grafo más formal.

"Un grafo G es un par G = (V, A) donde V es un conjunto finito de elementos que se denominan Vértices y A es un conjunto de pares no ordenados <x, y>, donde (x ε V) y (y ε V), denominados Aristas o Arcos." (Dpto. Central de Técnicas de Programación UCI, 2008).

Cuando se dice que A es un conjunto de pares no ordenados ***<x,y>,*** significa que los pares ***<x,y>*** y ***<y,x>*** se referirán a una misma arista, la que conecta los vértices ***x*** e ***y***. Esto significa que la relación entre los pares ***<x, y>*** es una relación simétrica; que tanto ***x*** puede estar relacionado con ***y*** al igual que ***y*** puede tener relación con ***x.*** Ver Fig. 7.

El uso de grafos en esta versión del subsistema, permitirá mediante la incorporación de las funcionalidades que se describirán más adelante, eliminar los problemas de usabilidad que presenta, ya que en la utilización y aplicación de las múltiples subordinaciones, propone una solución prometedora para lograr que el sistema permita al usuario gestionar, estructurar y organizar mejor la información de las entidades, siendo esta estructura el núcleo de la investigación y el principal elemento para alcanzar el resultado esperado en el desarrollo de la nueva versión de Estructura y Composición.

1.3 Metodologías de desarrollo

En las dos últimas décadas las notaciones de modelado y posteriormente las herramientas pretendieron ser las "balas de plata" para el éxito en el desarrollo de software, sin embargo, las expectativas no fueron satisfechas. Esto se debe en gran parte a que otro importante elemento, la metodología de desarrollo, había sido postergado. De nada sirven buenas notaciones y herramientas si no se proveen directivas para

su aplicación. Así, esta década ha comenzado con un creciente interés en metodologías de desarrollo. (Torres, y otros, 2010).

1.4 Metodología propuesta para el sistema

Para el desarrollo del presente trabajo de diploma se seguirá el nuevo modelo de desarrollo definido por el proyecto ERP Cuba. Este modelo está basado en compontes, centrado en la arquitectura, ágil y adaptable al cambio y es iterativo e incremental. De esta forma se obtiene como resultado el documento Modelo de desarrollo orientado a componentes del Proyecto ERP-Cuba llamado también modelo de desarrollo del CEIGE (Centro de Informatización y Gestión Empresarial).El trabajo se realizará aplicando este modelo por las características que posee anteriormente mencionadas:

- Este modelo de desarrollo está basado en las metodologías RUP y XP, dos de las metodologías más usadas en la construcción de software, tomando de estas las características más notables. (Ing. Jaime Robaina, y otros, 2009).
- Este modelo fue diseñado por especialistas profesionales a partir de las metodologías mencionadas en el punto anterior, se implantó hace poco más de un año obteniendo resultados satisfactorios para el proyecto, y para no violentar las decisiones de la dirección del Centro, el trabajo va a valerse del mismo.

- Tomaría mucho tiempo diseñar, estimar e implantar una metodología que se acople a las características del proyecto y que puede a largo o corto plazo cumplir o no con las necesidades del mismo.

1.5 Lenguajes de modelado

El lenguaje de modelado es un conjunto estandarizado de símbolos y de modos de disponerlos para modelar un software. En la mayoría de los casos son utilizados en combinación con una metodología de desarrollo de software para realizar la especificación del desarrollo de un software y de este modo hacerlo extensivo a todo el equipo de desarrollo. El uso de un lenguaje de modelado es más sencillo que la auténtica programación. Por su robustez y fiabilidad el equipo de desarrollo decidió que se utilizaría el

lenguaje UML (Unified Modeling Lenguage, lenguaje unificado de modelado) para especificar los artefactos que se deben generar.

UML: Es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. (James Runbaugh, 1998).

1.6 Herramientas de desarrollo y tecnologías

A continuación se detallan las herramientas que ayudarán al cumplimiento del objetivo general de este trabajo. Estas, en su mayoría, fueron las que se utilizaron para el desarrollo de la primera versión. Las mismas ponen en práctica el aprovechamiento y reutilización de la tecnología, un punto que siempre debe tenerse en cuenta cuando se va a desarrollar una versión superior de un software, pues buscar otras herramientas conllevaría a un gasto innecesario de tiempo y de recursos. Esto atado al hecho de que las definiciones arquitectónicas del proyecto están sobre estos pilares de desarrollo y sería impropio violentarlas si con las mismas se puede resolver la problemática en cuestión.

1.6.1 Marcos de trabajo

Zend Framework: Es un framework de código abierto para desarrollar aplicaciones y servicios Web con PHP 5. Se ejecuta utilizando un 100% de código orientado a objetos. Su estructura por componentes es algo único, cada componente ha sido diseñado con unas dependencias de otros componentes. Esta flexibilidad permite a los desarrolladores de la arquitectura utilizar los componentes individualmente. Ofrece una alta capacidad y robustez para la implementación del patrón Modelo Vista Controlador. (Zend Company, 2008).

Zend_Ext Framework 1.0: Es una extensión del framework de Zend desarrollada por el Centro de Informatización de Gestión Empresarial y el centro de Desarrollo y Asimilación de Tecnologías de la UCID (UCI-Defensa) con el objetivo de crear un marco de trabajo extensible y configurable, centrando el desarrollo de las aplicaciones en la lógica del negocio y en las interfaces de usuario y alejando a los

programadores de los detalles arquitectónicos, con soporte para entornos multientidad y para una arquitectura de sistema orientada a componentes.

Doctrine Framework: El framework Doctrine es una potente, multiplataforma y completo sistema para el mapeo de objeto-relacional (ORM: Object Relational Mapper, mapeador relacional de objetos) para PHP 5.2 ó superior.

ExtJS Framework: Es una biblioteca de Java Script para el desarrollo de aplicaciones web interactivas usando tecnologías como AJAX, DHTML y DOM. Incluye un alto rendimiento, interfaces de usuario personalizables, bien diseñado y extensible modelo de componentes, una interfaz intuitiva y fácil de utilizar con licencias de código abierto y comercial. (ExtJS, 2010).

1.6.2 Herramientas de Modelado

Visual Paradigm Suite for UML 6.4: Es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. (Sun Microsystems, 2008).

1.6.3 Herramientas y Tecnologías de Desarrollo

Zend Development Environment 5.5: Es un IDE (siglas en Ingles Integrated Development Environment en español entorno de desarrollo integrado) de código abierto para profesionales de los desarrolladores de PHP. Ha sido diseñado para maximizar la productividad de los desarrolladores por lo que le permite desarrollar, probar y depurar código más rápido, las cuestiones de aplicación de solucionar de forma rápida y mejorar la colaboración en equipo. (Zend Company, 2010).

Aptana Studio-1.2.1: Es un entorno de desarrollo de código abierto para la construcción de aplicaciones web, basado en la plataforma de herramientas de Eclipse. Incluye soporte para Java Script, HTML, DOM y CSS con código de finalización, depuración de Java Script, y aviso de error de notificación y documentación integrada. Está disponible para las plataformas Microsoft Windows, Mac OS y Linux. (Aptana, 2010).

1.6.4 Lenguajes de Programación

PHP 5.2: Es un lenguaje interpretado de alto nivel, diseñado originalmente para la creación de páginas web dinámicas de manera rápida y fácil. Es un lenguaje de programación usado principalmente en interpretación del lado del servidor. Esta versión sigue mejorando el soporte para la Programación Orientada a Objeto así como los temas de seguridad, el rendimiento y manejo de excepciones. (php.net, 2010).

Java Script: Java Script es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas. Técnicamente, Java Script es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con Java Script se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios. (Pérez, 2010).

PostgreSQL 8.3: Es un poderoso sistema de gestión de base de datos relacional orientada a objetos de software libre. Tiene una arquitectura probada por lo que ha ganado una sólida reputación para la fiabilidad, integridad y exactitud de los datos. Funciona en todos los principales sistemas operativos. Incluye los mejores tipos de datos. También soporta el almacenamiento de grandes objetos binarios, incluyendo imágenes, sonidos o vídeo. Tiene interfaces de programación nativo de C / C + +, Java, Net, Perl, Python, Ruby, Tcl, ODBC. (postgresql, 2010).

1.7 Especificaciones de la arquitectura

Teniendo en cuenta que el presente trabajo de diploma está centrado en el desarrollo de la versión 2.0 del subsistema Estructura y Composición, no es propósito de los autores crear una nueva arquitectura para el desarrollo de la herramienta y por tanto adopta la arquitectura definida por el proyecto ERP Cuba, la misma está basada en desarrollo por componentes y se encuentra actualmente en explotación.

1.7.1 Arquitectura Cliente/Servidor

La arquitectura Cliente/Servidor es una nueva tendencia en el desarrollo de redes, que tiene como objetivo optimizar el uso tanto del hardware como del software, a través de la separación de funciones: el cliente, quien inicia una determinada petición y el servidor, dedicado a responder dichas peticiones.

1.8 Requisitos

El flujo de trabajo de Levantamiento de Requisitos radica su mayor esfuerzo en el establecimiento de un acuerdo entre los clientes y los desarrolladores, sobre lo que el sistema debe hacer. Se precisa el ámbito del sistema y se pretende entender el comportamiento del software definiendo una interfaz enfocada a las necesidades y metas del usuario. Este flujo de trabajo intenta proveer a los desarrolladores de un mejor entendimiento de los requerimientos del sistema. (Jacobson, et al., 2004).

Existen dos clasificaciones para los requisitos de software: funcionales y no funcionales. En el capítulo siguiente se exponen y describen los requisitos funcionales y no funcionales de la versión 2.0 de Estructura y Composición.

1.9 Diseño

El modelo de diseño es una abstracción del modelo de implementación y su código fuente, el cual fundamentalmente se emplea para representar y documentar su diseño. Es usado como entrada esencial en las actividades relacionadas a la implementación. El modelo de diseño puede contener: los diagramas, las clases, paquetes, subsistemas, cápsulas, protocolos, interfaces, relaciones, colaboraciones, atributos, las realizaciones de los casos de uso, entre otros que se puedan considerar para el sistema en desarrollo.

1.10 Patrones

1.10.1 Singleton

El Patrón **Singleton** es un patrón de diseño, específicamente de creación, es utilizado para garantizar que una clase sólo tenga una instancia y proporcione un punto de acceso global a ella. Su funcionamiento puede resumirse a que oculta el constructor de la clase Singleton, para que no se puedan crear otras instancias.

1.10.2 Patrón Modelo-Vista-Controlador (MVC)

El **Modelo-Vista-Controlador** es un patrón de arquitectura utilizado en sistemas web para separar los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos, permitiendo flexibilidad y facilidad a la hora de hacer futuros cambios.

El **modelo** es un conjunto de clases que representan la información del mundo real que el sistema debe procesar. (Deacon, 2010).

Las **vistas** son el conjunto de clases que se encargan de mostrar al usuario la información contenida en el modelo. Una vista está asociada a un modelo, pudiendo existir varias vistas asociadas al mismo modelo. (Bergin, 2010).

El **controlador** es un objeto que se encarga de dirigir el flujo del control de la aplicación debido a mensajes externos, como datos introducidos por el usuario u opciones del menú seleccionadas por él. (Burbeck, 2009).

1.11 Prueba

La fase de pruebas es una de las más costosas del ciclo de vida software. En sentido estricto, deben realizarse pruebas de todos los artefactos generados durante la construcción de un producto, lo que incluye especificaciones de requisitos, diagramas de diversos tipos y, por supuesto, el código fuente y el resto de productos que forman parte de la aplicación. En el trabajo se llevarán a cabo pruebas a la implementación mediante pruebas de caja blanca para verificar que el código desarrollado presenta la calidad suficiente para darle uso a la aplicación.

1.11.1 Pruebas de caja blanca

Las pruebas de caja blanca están dirigidas a las funciones internas. Entre las técnicas usadas se encuentran: la cobertura de caminos, pruebas sobre las expresiones lógico-aritméticas, pruebas de camino de datos, comprobación de bucles. Mediante los métodos de prueba de caja blanca, se pueden obtener casos de prueba que:

- Garanticen que se ejercitan por lo menos una vez todos los caminos independientes de cada módulo.
- Ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa.
- Ejecuten todos los bucles en sus límites y con sus límites operacionales.
- Ejerciten las estructuras internas de datos para asegurar su validez.

1.11.1.1 Técnica del camino básico

La técnica del camino básico permite obtener una medida de la complejidad lógica del código de cada método, programa o módulo dado. La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes que existen en la codificación por los cuales puede circular el flujo de control. Es además una de las más eficientes en cuanto a cobertura de código, pues logra que se ejecuten todos los bucles en sus límites operacionales (Echavarría, 2008).

1.11.2 Métricas de Software

Según Pressman en si libro Ingeniería del software, un enfoque práctico, plantea lo siguiente:

"El proceso del software y las métricas del producto son una medida cuantitativa que permite a la gente del software tener una visión profunda de la eficacia del proceso del software y de los proyectos que dirigen utilizando el proceso como un marco de trabajo. Se reúnen los datos básicos de calidad y productividad. Estos datos son entonces analizados, comparados con promedios anteriores, y evaluados para determinar las mejoras en la calidad y productividad. Las métricas son también utilizadas para señalar áreas con problemas de manera que se puedan desarrollar los remedios y mejorar el proceso del software". (Pressman, 2005).

1.11.2.1 Métricas de usabilidad

Se pueden definir las mismas como aquellos criterios o variables que son medibles de forma objetiva. Mientras que la interpretación de una opinión es un análisis cualitativo o subjetivo por parte del experto, la interpretación de datos objetivos responde a un análisis cuantitativo. (deinterfaz., 2010).

Conclusiones del capítulo 1

- El estudio anterior de los elementos necesarios para el trabajo, ha brindado un gran apoyo para el inicio del mismo, ya que permitió sentar las bases para el comienzo del desarrollo de la nueva versión. En el mismo se llevo a cabo un análisis acerca de los problemas que afectan la usabilidad de la versión en uso, debido a las deficiencias y problemas que posee.
- También se trataron aspectos acerca de las estructuras dinámicas de datos las cuales serán empleadas por sus características principales como son la adaptabilidad y la flexibilidad que

poseen en el manejo de datos en tiempo real. Entre las estructuras de datos lineales se estudiaron las listas, pilas y colas y sus aplicaciones en la vida real. De las mismas se definió utilizar la filosofía de listas la cual es ideal para el almacenamiento y búsqueda lineal de datos en tiempo real.

- De las estructuras no lineales se realizó un estudio acerca de las estructuras arbóreas, el cual brindó resultados positivos para la investigación y serán empleadas para representar la estructura organizativa de las subordinaciones.
- Además se estudiaron los grafos, de los cuales se determinó que esta estructura de datos sería factible para implementar las funcionalidades que darán vida a la gestión de subordinaciones dentro de la nueva versión.
- Por otra parte, se aplicará el Modelo de Desarrollo del CEIGE definido por el centro debido a las características antes mencionadas.
- Las herramientas que se utilizarán para el desarrollo del sistema serán las mismas que sirvieron en la implementación de la primera versión ya que bastan para resolver el problema.
- La arquitectura empleada será la de Cliente-Servidor que es la empleada por el Centro para el desarrollo de proyectos.
- Por último, para validar la variable usabilidad y buen diseño del sistema a desarrollar, se utilizarán las métricas de diseño Tamaño Operacional de Clase y Relación entre Clases, así como las de usabilidad y caja blanca, descritas anteriormente.

Capítulo 2: Descripción de la solución del sistema

Introducción

En este capítulo se realiza la descripción de la propuesta de solución del presente trabajo, para ello se enumeran, describen, diseñan e implementan los requisitos que se deben incluir al sistema.

2.1 Propuesta de la versión

En el este trabajo, se ha propuesto desarrollar una nueva versión la cual tenga constancia de un correcto rediseño de la versión en uso, donde se eliminen los problemas de usabilidad que presenta la misma, en la cual el usuario podrá crear subordinaciones entre estructuras de una manera mucho más fácil y eficiente. Visualizar de forma más sencilla, y más detallada, las subordinaciones que hayan sido creadas así como las existentes. La misma brindará además, la posibilidad de mostrar la composición estructural correspondiente a cada una de estas subordinaciones, tanto activas, como inactivas, así como registrar subordinaciones que sean modificadas en un historial que tendrá cada una. El usuario también podrá realizar búsquedas, donde especifique los datos de la subordinación deseada, así como activar subordinaciones que no estén en uso, que hayan sido registradas con anterioridad.

De todo lo antes mencionado se encargará la nueva versión de Estructura y Composición, ya que es en este subsistema donde se manipulará toda la información de una entidad a partir de la creación de su estructura, donde luego se gestionarán las subordinaciones pertinentes, con el objetivo de organizar y visualizar mejor su contenido, obteniendo una vista más lógica y real de las entidades dentro de la empresa. Para ello se hará uso de estructuras arbóreas, y se incorporarán también las estructuras basadas en grafos, el centro de este trabajo. Además de grafo y árboles se utilizará la filosofía de las estructuras de datos listas para las operaciones con objetos que se manipulan desde el servidor, obtenidos tanto del modelo como de la controladora.

2.1.1 Múltiples Subordinaciones

El problema que presenta Estructura y Composición 1.0 se podrá solucionar con la implementación de las múltiples subordinaciones basadas en grafos auxiliándose de las estructuras descritas anteriormente. El

uso de estos elementos permitirá al subsistema crear varias subordinaciones a partir de una misma estructura. Esto será posible principalmente por la utilización de la estructura de dato no lineal “grafo”, la cual presenta características que darán soporte para desarrollar finalmente las funcionalidades necesarias para obtener el producto final y dar cumplimiento al objetivo del trabajo, buscando siempre la satisfacción del cliente que posteriormente le dará un uso y explotación, con el fin de lograr una organización estructurada y real de las entidades y sus procesos.

Una subordinación es una relación entre dos o más estructuras, cuyo significado es el mismo para cada una de ellas. Representa un conjunto finito de **n** (n>1) elementos que se encuentran englobados o agrupados en una misma relación Ejemplo: una subordinación de tipo económica entre un conjunto de entidades.

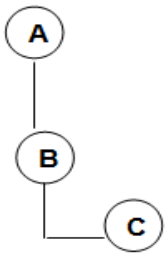
Las subordinaciones se basan en relaciones de dependencia que presentan las estructuras entre sí llamadas también jerarquías, donde esta dependencia puede variar o no según el tipo de relación que posean las estructuras. Cuando una estructura presenta más de una relación jerárquica entre los elementos que la conforman, a esto se le denomina **subordinación múltiple**.

Al aplicar las estructuras de datos grafos al concepto de múltiples subordinaciones se podrá obtener el siguiente resultado: una estructura **X** puede ser padre de una estructura **Y,** pero además la estructura **Y** podría al mismo tiempo, ser padre de **X** en dependencia del **tipo de relación** o tipo de **subordinación** que se establezca entre ellas.

Para entender mejor el concepto de múltiples subordinaciones se muestra el siguiente ejemplo práctico:

Si se tienen tres entidades **A**, **B** y **C**, las cuales pertenecen a la entidad UCI. La entidad **A** es el rector de dicha entidad, la **B** es el vice-rector de trasportación y la entidad **C** es el chofer de **B**.

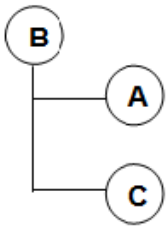
Si el tipo de relación o jerarquía de estas tres entidades fuera Administrativa la dependencia de las mismas sería la siguiente:



Jerarquía administrativa (B se subordina a A, C se subordina a B, A no se subordina a nadie en dicha subordinación).

Fig. 8 Árbol representando una subordinación de tipo administrativa

Pero si el vice-rector de transporte alias **B**, es secretario del PCC a nivel UCI, y las entidades **A** y **C** son solamente militantes de la organización, la dependencia entre estas cambiaría completamente si se cambia el tipo de relación y en lugar de establecerse una jerarquía administrativa, y se quisiera crear una relación jerárquica política, la estructura quedaría de la siguiente forma:



Jerarquía política (**B** seria el padre de **A** y **C**, y estas dos últimas estarían al mismo nivel).

Fig. 9 Árbol representando una subordinación de tipo política

Como se ha podido ver, la estructura de la entidad (UCI) ha pasado de ser **UCI(A (B (C))) a UCI(B (A, C))**, esto ha sucedido pues, la estructura organizativa puede también cambiar si el tipo de jerarquía o subordinación cambia, aunque no necesariamente tiene que cambiar siempre, se puede dar el caso que esto suceda, lo que significa que una entidad debe ser capaz de gestionar su estructura externa e interna, pero, para lograr una mayor fortaleza y organización de la información, lo más lógico, y cercano a la realidad es que esta información se agrupe en subordinaciones, partiendo de las relaciones existentes entre las estructuras, lo cual sería el punto de partida para realizar los demás procesos y tareas que se manejan en estas.

Capítulo 2: Descripción de la solución del sistema.

Para dar solución al problema de la investigación identificado en el Capítulo 1 se decidió incluir a la versión actual de Estructura y Composición 1.0 un conjunto de funcionalidades las cuales deben permitir al usuario:

- Adicionar, buscar, ordenar y listar subordinaciones así como activar una subordinación inactiva cuando sea requerido.
- Consultar las estructuras relacionadas de una subordinación.

2.2 Requisitos funcionales

Las técnicas utilizadas para el Levantamiento de Requisitos de la nueva versión de Estructura y Composición fueron las entrevistas realizadas a los clientes y los jefes de Estructura y Composición, los talleres impartidos por estos, ámbito que fue propicio para desencadenar lluvias de ideas. De las mismas surgieron los requisitos que se muestran y detallan a continuación.

2.3 Descripción de los nuevos requisitos

2.3.1 ***Adicionar subordinación****.*

El sistema debe permitir registrar las estructuras que pertenecerán a la subordinación e indicar a qué estructura se subordinará cada una.

2.3.2 ***Activar subordinación****.*

El sistema debe permitir al usuario activar la subordinación.

2.3.3 ***Listar subordinación****.*

El sistema muestra al usuario un listado de las subordinaciones. Se muestran la descripción de la subordinación, la fecha de inicio y en caso de estar inactiva la fecha en que se desactivó o fecha de fin y el listado de estructuras que la componen.

2.3.4 ***Ordenar subordinación****.*

El sistema permite ordenar las subordinaciones por varios campos como el tipo de jerarquía, la fecha de inicio y la fecha de fin.

2.3.5 ***Consultar subordinación****.*

El sistema muestra el listado, organizado jerárquicamente, de estructuras que componen la subordinación.

2.3.6 ***Buscar subordinación****.*

El sistema muestra un listado de las subordinaciones que cumplen los criterios de búsqueda especificados. Se muestran tipo de subordinación, Fecha de Inicio, Fecha de fin y descripción de las subordinaciones.

2.4 Diseño de la solución

2.4.1 Diseño de interfaces de usuario

En apoyo al mejor entendimiento y desarrollo de esta aplicación se comienzan a definir y diseñar las principales vistas que van a interactuar con el usuario final, en busca de aminorar la complejidad y abstracción que puede conllevar la implementación de un software para los desarrolladores y también con el objetivo de evaluar el nivel de satisfacción del cliente y garantizar el entendimiento con este. Estas vistas parten de las necesidades que dan lugar a la creación del software y casi nunca en un principio se ajustan a todos los requisitos que se proponen, es sólo durante su desarrollo que se alcanzan los prototipos finales de los que va a disponer el sistema. La aplicación constará de 3 vistas o interfaces, en las que se gestionarán todas las funcionalidades que tendrá, descritas en el epígrafe que describe la solución propuesta. Cada una de ellas se encuentra estrechamente relacionada y se encuentran descritas a continuación. Seguidamente se muestra el prototipo desarrollado para la interfaz principal (Figura 13).

2.4.2 Prototipo interfaz de usuario gestionar subordinación (interfaz principal)

En la interfaz principal se muestran en el panel que cubre la parte izquierda y central de la interfaz, el listado ordenado por tipo, de subordinaciones. Se muestran la descripción (debajo del tipo de subordinación), el tipo de subordinación, la fecha de inicio y en caso de que se desee ver las subordinaciones inactivas se mostrará la fecha de desactivación. En la parte derecha se muestran los detalles de la subordinación que se desee consultar donde se muestra el tipo de subordinación y el árbol de estructuras que la componen.

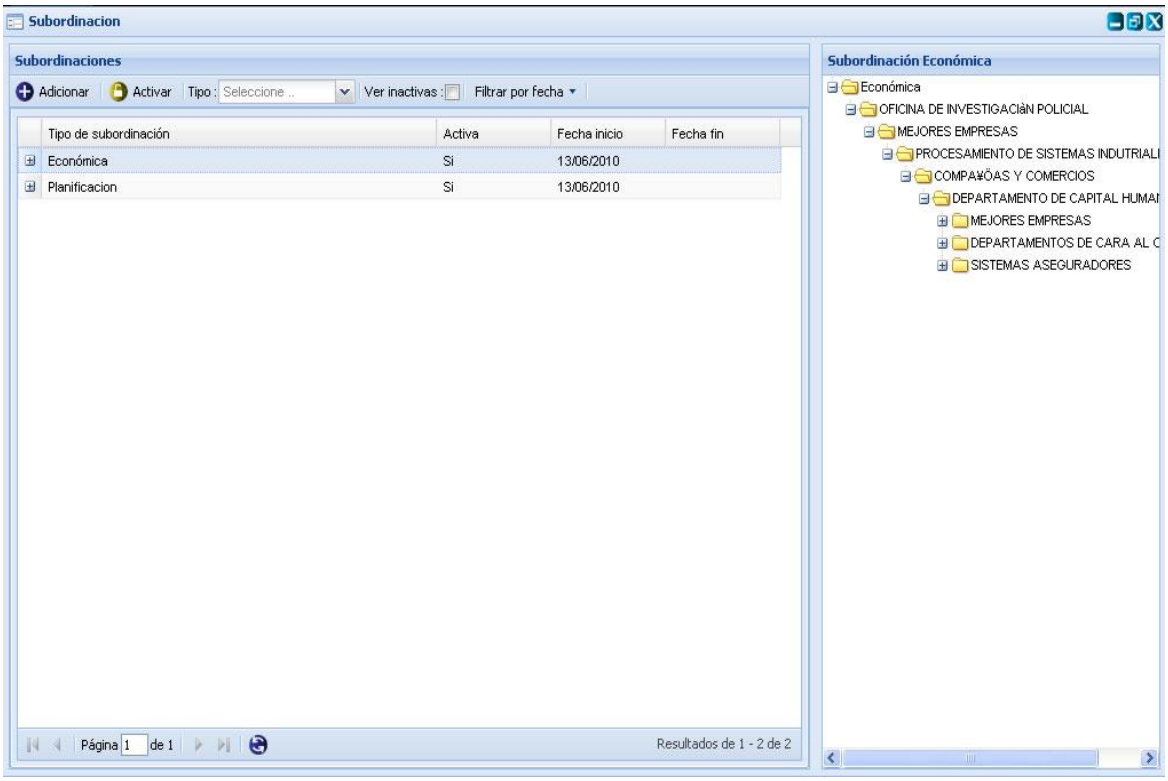


Fig. 10 Gestionar subordinación (interfaz principal)

2.4.3 Prototipo Interfaz de Usuario R1 Adicionar Subordinación

En la interfaz adicionar subordinación se muestran en la parte izquierda, los campos: tipo de subordinación donde se especifica la jerarquía a la que pertenecerá dicha subordinación, debajo de esta se encuentra el campo **descripción**, donde se inserta una breve descripción de la subordinación que se creará una vez que se organicen en la región central y derecha donde aparece el árbol de las subordinaciones, el cual permite al usuario crear si no existe una subordinación de un tipo determinado y si ya existe algún tipo de subordinación de ese tipo pues entonces el usuario puede hacer cambios en la misma.

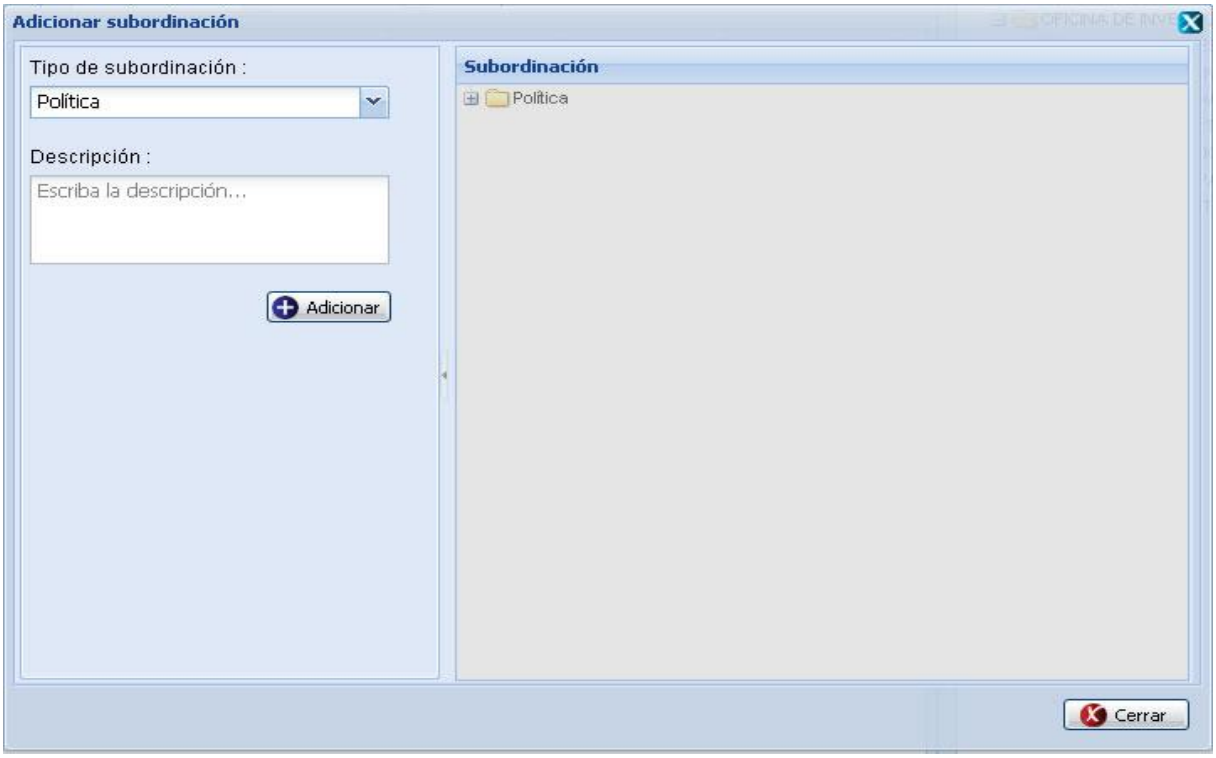


Fig. 11 Interfaz Adicionar subordinación

2.4.4 Prototipo Interfaz de Usuario R2 Activar Subordinación

En esta interfaz se muestra un listado organizado de las subordinaciones inactivas que existen. Si al seleccionar una jerarquía o tipo de subordinación cualquiera dicha sea, si existe una subordinación de ese tipo activa entonces esa selección pasaría como la subordinación activa de ese tipo y automáticamente la que está activa se desactiva y pasaría entonces para el listado de subordinaciones inactivas

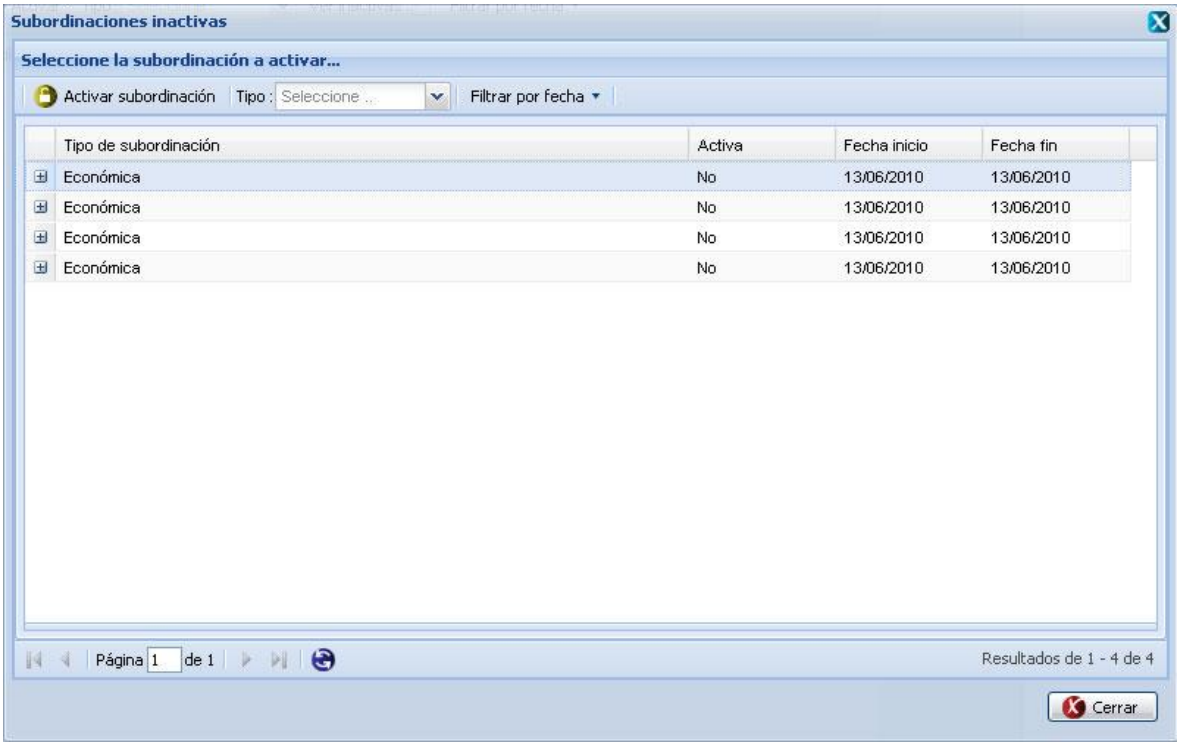


Fig. 12 Interfaz de usuario activar subordinación

2.4.5 Modelo de diseño

Este diagrama tiene como objetivo representar la estructura que sigue una aplicación web basándose en los principales componentes que la integran y su relación entre sí. En la figura siguiente se describen las interfaces de usuario con sus formularios correspondientes en relación con la página servidora que controla todas las funcionalidades que se realizan, y el conjunto de las principales clases que tributan a la existencia de dichas funcionalidades. Como se puede apreciar las páginas clientes están compuestas por archivos java script los cuales representan las interfaces de usuario de la aplicación. Por otra parte estas interfaces están compuestas por formularios, los cuales registran los datos que serán enviados al servidor para su manipulación. Finalmente las clases del modelo de datos proveen la información solicitada por la controladora y esta se encarga de que las páginas clientes reciban la misma, separando las interfaces de la controladora y el modelo de datos, siguiendo la filosofía del patrón de diseño **Modelo**- **Vista**-**Controlador** el cual se describe más detalladamente en el capítulo anterior y se aplica a continuación.

2.4.6 Diagrama de Clases del Diseño

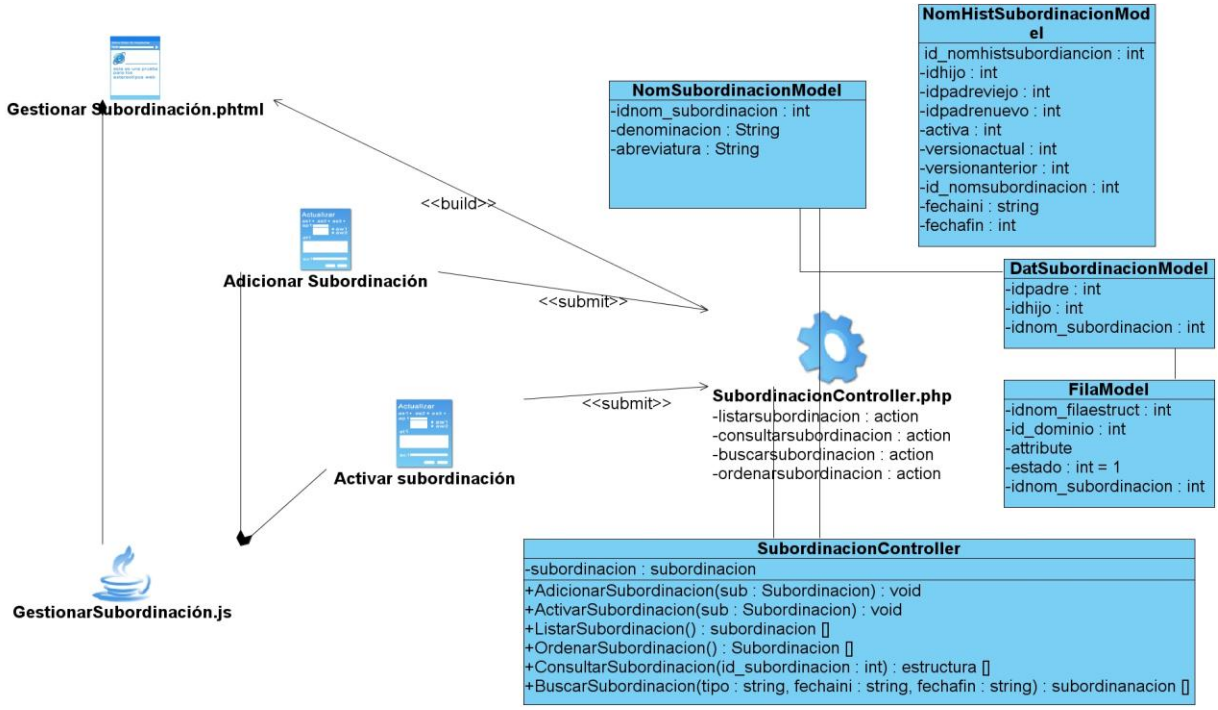


Fig. 13 Diagrama de clases del diseño Gestionar Subordinación

2.4.7 Aplicación de patrones del diseño

El patrón MVC (Modelo Vista Controlador) también se aplicó en el diseño del subsistema para representar la arquitectura del sistema. En **controllers** se guardan las clases controladoras de la aplicación, en **views**, se guardan los archivos relacionados con las interfaces de usuario y en **models** se guardan los archivos del modelo de datos donde se crean y utilizan las consultas de la aplicación:

El paquete **Views** contiene cada una de las clases necesarias para modelar la funcionalidad del componente, una interfaz fácil de utilizar y manejable mediante la cual el cliente interactúa con la aplicación para realizar cada una de las funcionalidades de la misma.

En el paquete **Controllers** se encuentran las clases controladoras encargadas de gestionar las funcionalidades del sistema

Contiene las clases necesarias para acceder a los datos que persisten en la base de datos y las clases generadas por el ORM Doctrine a partir de cada una de las tablas existentes en la base de datos. Cada una de estas clases heredan de una clase generada igualmente por el Doctrine las cuales se ubican en otro paquete dentro del paquete **Domain** llamado **Generated**.

2.5 Modelado de la base de datos

Un modelo de datos es un conjunto de conceptos, reglas y convenciones que permiten describir y manipular los datos del mundo real que se desea almacenar en la base de datos. Al producto del modelo de datos se le llama esquema (descripción de la estructura de la base de datos) y a los datos en concreto almacenados en la base de datos en ese momento, ocurrencia del esquema.

Para cumplir con las exigencias propuestas para el correcto desarrollo del conjunto de funcionalidades que debe presentar la versión a desarrollar se ha realizado el siguiente Diagrama Entidad-Relación el cual está compuesto por cuatro entidades, de estas se crearon tres nuevas; estas son: "**nom_subordinación**", "**dat_subordinacion**" y "**nom_histsubordinacion**", la primera tiene como atributos el **id** de la subordinación, siendo este la llave primaria, la denominación describe el tipo de subordinación (Económica, Política, etc.), la abreviatura es la representación en siglas del tipo de subordinación, en la entidad "**nom_filaestruct**" es donde se construye el árbol físico de estructuras, esta tiene una relación de muchos a muchos con "**nom_subordinacion"** por lo que se genera una tabla "**dat_subordinacion**" que es donde estarán subordinadas de diferentes tipos las estructuras tiene como llave primaria las dos llaves de las tablas por la que está formada, además tiene una tercera ("**idpadre"**) que también es foránea de "**idfila"** en "**nom_filaestruc"**.

Es aquí donde se aplica la solución basada en la estructura dinámica grafo debido a que las estructuras se pueden subordinar unas a otras, siempre y cuando el tipo de subordinación no sea el mismo, pues la llave primaria no lo permite al ser de tipo ternaria, validando así que una estructura se pueda subordinar a otra y a la vez en otro tipo de relación esto pueda invertirse, o sea se cumple la definición de grafo planteada anteriormente en el Capítulo 1.

Además tiene otros atributos como son, “activa” que toma valores verdadero o falso en dependencia del estado que tenga, la descripción; es un pequeño fragmento de información de la subordinación a la que pertenece, y “**fechaini**”; guarda le fecha en la que es creada la subordinación.

Por último la entidad “**nom_histsubordinacion”**; guarda el historial de cambios pertenecientes a la entidad “**dat_subordinación** “ para una posterior recuperación, tiene atributos tales como “**idhijo”** que es el **id** de la estructura que ha sido cambiada, “**idpadreviejo”** que representa a cual estructura estaba subordinada anteriormente, “**idpadrenuevo”**, constituye la nueva estructura a la que se subordinó, “**idtiposub** “incorpora el **id** del tipo de subordinación, “**versionactual”** y “**versionanterior”** guarda el número de las versiones, facilitando el traslado entre estas, “**idnomhistsubordinacion”**, forma la llave primaria, “**fechaini”** toma la fecha en que fue creada la subordinación, “**fechafin”** no es más que el momento en que pasa a estar inactiva, y por último el atributo activa que representa el último cambio que se hizo.

La siguiente figura muestra el modelo de datos propuesto durante el diseño de la versión 2.0 de Estructura y Composición.

2.5.1 Modelo de Datos

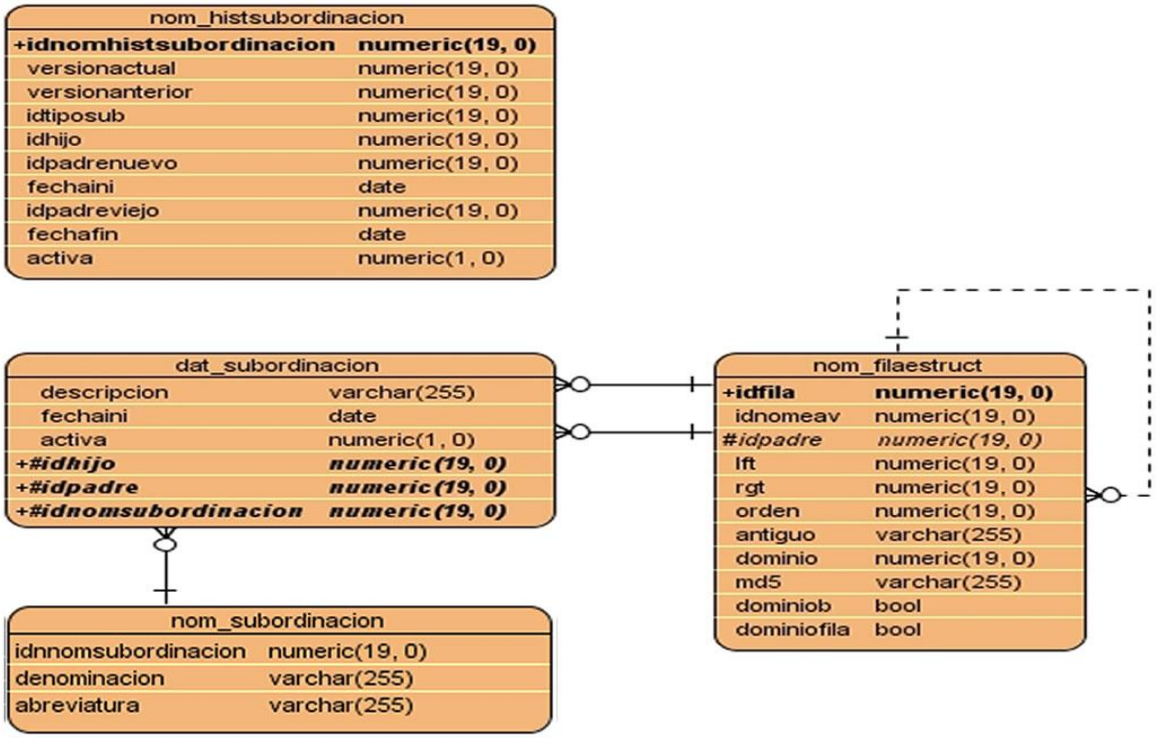


Fig. 14 Diagrama Entidad-Relación Gestionar Subordinación

2.5.2 Descripción del modelo de datos

TABLAS	DESCRIPCIÓN
nom_subordinacion	Guarda los datos correspondientes al nomenclador de subordinación (Ej. : Política, Administrativa, Económica)
dat_subordinacion	Guarda los datos correspondientes a las asignaciones de subordinación
nom_filaestruct	Guarda los datos correspondientes al nomenclador de estructuras
Nom_historialsubordinacion	Guarda los cambios correspondientes a las subordinaciones

Tabla 1 Descripción del modelo de datos Gestionar Subordinación

2.5.3 Diagrama de clases

A continuación se muestra el diagrama de clases de la solución propuesta. El mismo consta de cuatro tablas de las cuales tres son propias de la solución (**nom_subordinacion**, **dat_subordinacion**, **nom_histsubordinacion**), y **nomfilaestruct** la cual guarda los datos de las estructuras que van a estar agrupadas en las subordinaciones existentes. La clase **nom_subordinacion** tiene una relación de cero a muchos con **dat_subordinacion,** porque puede que existan tipos de subordinaciones que no estén presentes en una subordinación. La tabla **nomfilaestruct** tiene una relación de uno a muchos con **dat_subordinacion** debido a que una subordinación está compuesta por varias estructuras. La tabla **dat_subordinacion** tiene una relación de uno a muchos con **nom_histsubordinacion** ya que toda subordinación tiene un historial.

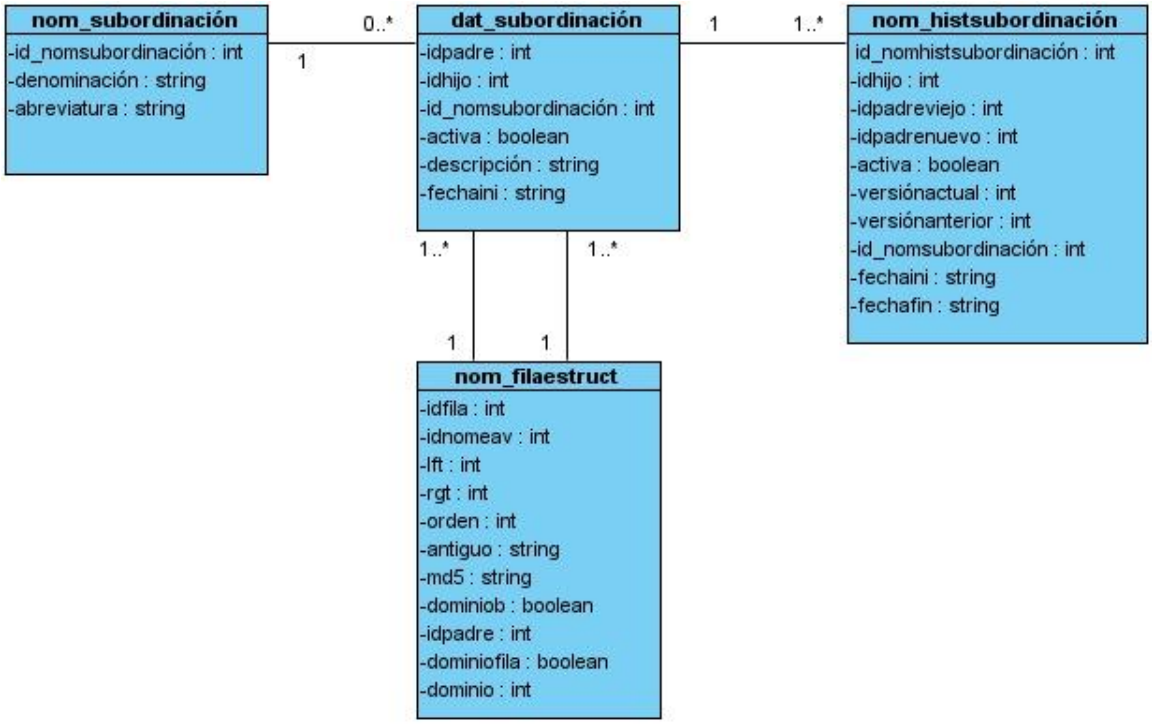


Fig. 15 Diagrama de clases Gestionar Subordinación

2.6 Implementación de la aplicación

Una vez definidas las interfaces candidatas a la solución final es muy importante la implementación continua de la aplicación por parte de los desarrolladores. El trabajo en conjunto puede brindar muchas facilidades a la hora de integrar las diferentes partes del software y lograr un mejor entendimiento entre el personal encargado de la lógica de presentación para con los de la lógica del negocio. Luego de disponer de las clases entidades y controladora del negocio representadas en los diagramas y modelos mostrados anteriormente, se procede a su implementación haciendo uso de los patrones y estándares que permitan obtener un código de alta calidad y usabilidad. En este caso se sigue un estilo **Modelo Vista Controlador** y se aplica la creación de sólo una instancia para cada clase en todo el sistema (**Singleton**), características que serán explicadas a continuación.

2.6.1 Patrón Singleton

En el caso de este software el patrón Singleton se utiliza en todas las clases dentro del paquete Model (DatSubordinacionModel.php, NomHistsubordinacionModel.php, NomSubordinacionModel.php, NomFilaestruc.php) haciéndolas actuar como instancias únicas en todos los casos para así poder utilizar cada una de ellas en la clase controladora (SubordinacionController.php). En la siguiente figura se muestra un ejemplo de código donde se aplica dicho patrón.

```
public function NomHistsubordinacionModel()
{
    parent::ZendExt_Model();
    $this->instance = new NomHistsubordinacion();
}
```

Fig. 16 Uso del Patrón Singleton

2.6.2 Patrón MVC

Este patrón es el utilizado para definir la arquitectura que tendrá el sistema, puesto que constituye una potente guía a seguir en el desarrollo de aplicaciones web y además, por estar implementado dentro de la arquitectura del framework seleccionado para la implementación del negocio, en este caso el Zend Framework de PHP. Esta aplicación se ajustará estrictamente a la descripción de cada una de sus partes, para garantizar el mejor aprovechamiento del patrón y a la vez obtener una herramienta de calidad y usabilidad en el mundo del software actual. Se definen en la parte de las vistas, las 3 interfaces con las

que contará la aplicación, englobadas en el fichero (GestionarSubordinacion.js), descritas en el diseño de la solución propuesta, dentro de los modelos estarán un conjunto de clases que manejarán toda la información necesaria (DatSubordinacionModel.php, NomHistsubordinacionModel.php, NomSubordinacionModel.php, NomFilaestruc.php) que se mostrará en las vistas y serán controladas por una única clase controladora (SubordinacionController.php). La estructura que propone este patrón para la implementación del sistema se ve evidenciada en el diagrama que representa la estructura física de las clases, anteriormente descrita. Ver Fig. 19.

2.6.3 Nomenclatura

Clases en las vistas

Se definió para las clases que se encuentran dentro de **Views** que el nombre se escribirá sin ningún sufijo o prefijo. Los nombres se escribirán en notación camello (Camel Case). Ejemplo: GestionarSubordinacion.js.

Clases Controladoras.

Se definió para las clases controladoras que después del nombre se les colocará como sufijo la palabra:"Controller". Ejemplo: SubordinacionController.php

Clases de los modelos.

Business (Negocio)

Se definió para las clases que se encuentran dentro de **Business** que después del nombre se les colocará como sufijo la palabra:"Model" Ejemplo: DatSubordinacionModel.php

Domain (Dominio) y Bases del Dominio (Generated).

Se definió para las clases que se encuentran dentro de **Domain** que el nombre se escribirá sin ningún sufijo. Ejemplo: NomFilastruct.php.

Nomenclatura de las funciones.

Se definió que el nombre de las funciones se escribe con la primera palabra en minúscula, para el caso, específico de que sea un nombre compuesto se empleará la notación CamelCasing (notación camello) y

de forma tal que al leerlo se conozca el objetivo de la misma. Ejemplo: activarSubordinacion(). En caso particular que la función sea una acción de la clase controladora u otra se escribe después del nombre de la función la palabra “Action” como sufijo. Ejemplo: buscarhijosAction().

La clase **SubordinacionController** es la clase controladora encargada de unir la vista (gestionarSubordinacion.js) con las clases modelos **nom_subordinacion**, **dat_subordinacion**, **nom_histsubordinacion** y **nom_filaestruct**, realizando llamadas a las funciones de estas.

Nombre: SubordinacionController	
Tipo de clase: Controladora	
Nombre:	**Descripción:**
init ()	**Constructor de la clase**
mostarNivelsubordinacionAction()	**Muestra los tipos de subordinaciones existentes**
insertarSubordinacionAction()	**Inserta una nueva subordinación**
mostrarListadojerarquiaAction()	**Muestra un listado de las subordinaciones activas**
cargarEstructurasAction()	**Muestra las estructuras pertenecientes a una subordinación**
mostrarInactivasAction()	**Muestra un listado de las subordinaciones inactivas**
activarSubordinacionAction()	**Activa los cambios guardados en los historiales**
cambiarHijosAction()	**Cambia la forma de subordinarse una estructura a otra**

Tabla 2 Descripción de la clase controladora SubordinacionController

Esta es la clase encargada de gestionar los tipos de subordinaciones. Aquí se definen las jerarquías a las que van a pertenecer las estructuras.

Nombre: NomSubordinacionModel	
Tipo de clase: Entidad	
Nombre:	Descripción:
NomSubordinacionModel()	Constructor de la clase
insertar($denom, $abreviatura)	Inserta un tipo de subordinación
buscarIdproximo( )	Asigna un número de id al insertarse una nueva tupla
buscarCampos($limit, $start, $idtabla)	Muestra los datos de la tupla q tenga el id pasado
buscarNomSubordinacionSB($limite = 1000)	Muestra un listado de los tipos de subordinaciones

Tabla 3 Descripción de la clase entidad NomSubordinacionModel

Esta es la clase encargada de gestionar las subordinaciones. Aquí se relacionan y agrupan las estructuras con el nomenclador de subordinación formando las subordinaciones lógicas.

Nombre: DatSubordinacionModel	
Tipo de clase: Entidad	
Nombre:	Descripción:
DatSubordinacionModel()	Constructor de la clase
insertarSubordinacion($idhijo, $idpadre, $idnomsubordinacion, $descripcion)	Muestra los tipos de subordinaciones existentes
mostrarListadojerarquia($activa,$idsubordinacion,$fechaini,$limit, $start, $fechafin)	Inserta una nueva subordinación
obtenerHijosSubordinacion($idsubordinacion)	Muestra un listado de las subordinaciones activas
obtenerHijosPadre($idPadre,$idsubordinacion)	Muestra las estructuras pertenecientes a una subordinación
obtenerTodos($idsubordinacion)	Muestra un listado de las subordinaciones inactivas
actualizarFila($idpadreviejo,$idpadrenuevo,$idhijo,$idsubordinacion,$descripcion,$fechaini)	Activa los cambios guardados en los historiales
verInactivas($idsub,$fechaini,$fechafin,$limit,$start)	Cambia la forma de subordinarse una estructura a otra

Tabla 4 Descripción de la clase entidad DatSubordinacionModel

Esta es la clase encargada de guardar los historiales de las subordinaciones, para su posterior recuperación.

Nombre: NomHistsubordinacionModel	
Tipo de clase: Entidad	
Nombre:	Descripción:
NomHistsubordinacionModel()	Constructor de la clase
insertarHistorial($idpadreviejo,$idpadrenuevo,$versionactual, $versionanterior, $idhijo, $idsubordinacion)	Inserta un nuevo historial
buscarIdproximo( )	Asigna un número de id al insertarse una nueva tupla
buscarUltimaversion($idsubordinacion)	Devuelve el número de la mayor versión
versionAnterior($idsubordinacion)	Devuelve el número de la versión anterior a la que se está.
mostrarInactivas($idsub, $fechaini, $fechafin, $limit, $start)	Devuelve un listado de las subordinaciones inactivas
obtenercambiosSubordinacion($idnomsubordinacion)	Devuelve un listado de los cambios existentes dado un tipo.
obtenerultimaVersion($version,$changes)	Devuelve el ultimo id de una versión
IrAtras($camino,$changes)	Va adicionando nodos al camino hasta que se intercepten
intercepto($camino1,$camino2)	Chequea si un camino se intercepta con otro
caminoHastaIntercepto($camino,$intercepto)	Le asigna un atributos al cambio que sea el intercepto
crearConexion($camino1,$camino2,$intercepto)	Devuelve el listado real de la cantidad de cambios que tiene el camino

Tabla 5 Descripción de la clase entidad NomHistsubordinacionModel

Nomenclatura de las variables.

Se definió que el nombre a emplear para las variables se escribe en minúscula y en caso de ser un nombre compuesto se escriben ambas palabras en minúscula separadas por un guión bajo. Ejemplo:

variable

variable_compuesta.

2.6.4 Normas de los comentarios

En pos de garantizar un código más legible y reutilizable se convierte en una necesidad el comentariado de todo el código que se implemente dentro del desarrollo, con el objetivo de aumentar su mantenibilidad a lo largo del tiempo.

Nomenclatura de los comentarios.

Se definió el uso de comentarios claros y precisos que ayudarán a una mejor comprensión del código implementado para que se entiendan sus objetivos específicos.

En las clases

Para implementar una clase se escribe una pequeña descripción donde se expliquen los propósitos de la misma de forma general, así como su autor y sistema al que pertenece. Dicha descripción se escribe de la siguiente forma:

```
1   <?php
2   /*
3   * Esta clase ha sido generada por Doctrine Generator
4   */
5   /**
6   * Nombre de la clase NomHistSubordinacion
7   * Descripcion *Esta es la clase encargada de guardar
8   * los cambios *  en las subordinaciones respectivas
9   * @author *Lázaro E. Dominguez Suarez
10  * @package *E y C
11  * @subpackage *(sub módulo)
12  * @copyright *ERP
13  * @version 2.0
14  */
```

Fig. 17 Ejemplo de comentario de clase

Comentarios por líneas

Se utilizan también los comentarios para líneas de código en específico que ayudan a aclarar la complejidad de algunos algoritmos. Para ellos se utiliza //, y seguidamente el comentario que describe la línea de código, tal y como se representa a continuación.

```
if (respuesta == "true") {  //Este if es para cuando tiene
                            //  ya una subordianación de ese tipo
    var loader = ColumnaArbolDer.getLoader();
    loader.url = 'loadStructures';
```

Fig. 18 Ejemplo de comentario por líneas de código.

2.6.5 Tratamiento de errores

Los errores se manejan de forma sencilla, tanto en las interfaces de usuario como en la lógica del negocio, para garantizar una respuesta a los comportamientos no esperados del sistema o la mala operatividad del usuario.

En el caso de las interfaces, se utilizan mensajes para validar campos en blancos, o para realizar acciones que requieran seleccionar una fila de un grid. Los errores son manejados mediante sentencias de condición if, else y la muestra de mensajes al usuario. A continuación se muestra un ejemplo de código:

```
852  ////////////////////////////////////ACTIVAR SUBORDIANCION//////////////////////
853  function Act_Sub() {
854
855      nombre= record.get('NomSubordinacion.text');
856      if(SM_DES.hasSelection()){//si el grid
857                                //tiene selección entonces
858                                //muestra el mensaje de confirmación.
859      mostrarMensaje(2,'Está seguro que desea activar la subordinación'+" "+nombre
860      + " " + '?',Activa)
861
862  }
863      else mostrarMensaje(3,'Debe seleccionar una subordinación',Ext.emptyFn());
864
```

Fig. 19 Tratamiento de errores en la interfaz

Por otra parte, los errores en la lógica de negocio validan nuevamente los datos que reciben de las interfaces y además chequean que el resultado de una función determinada sea el esperado, apoyándose en las sentencias de condición try – catch utilizando la clase excepción de ZendExt_Exception.

```
public function eliminarFila( $pidFila ){
    try
    {
        $mg = Doctrine_Manager::getInstance();
        $conn = $mg->getConnection('metadatos');
        $query = Doctrine_Query::create($conn);

        $result = $query ->delete('idfila')->from('NomFilaestruc')
        ->where("idfila = '$pidFila'")
        ->execute ();
        return true;
    }
    catch(Doctrine_Exception $ee)
    {
        throw new ZendExt_Exception('EC22', $ee);
    }
}
```

Fig. 20 Tratamiento de errores al código

Conclusiones del capítulo 2

- La aplicación del modelo seleccionado, en el capítulo anterior, refleja el análisis, diseño e implementación de la solución desarrollada en este trabajo.
- Con el fin de resolver los problemas que presentaba el subsistema Estructura y Composición se le dio cumplimiento a los nuevos requerimientos funcionales definidos.
- Se desarrollaron los artefactos necesarios en cada etapa, brindando una descripción de cada uno de ellos, con el objetivo de apoyar, junto a la descripción de los distintos tipos de clases definidas y la nomenclatura de sus variables y atributos, la implementación por parte de los desarrolladores y conformar la documentación del software para el proyecto.
- Para lograr la reutilización y optimización del código se planteó el uso de patrones arquitectónicos y de diseño y se detalló la forma y el lugar en que fueron utilizados dentro de la solución, en conjunto con la aplicación de las nomenclaturas definidas para el comentariado.
- Las novedades que propone este sistema con respecto a la versión anterior se ven reflejadas dentro de la utilización de nuevas funcionalidades del subsistema las cuales se aplicarán en el proceso de Gestión de Subordinación del proyecto ERP-Cuba, las cuales facilitarán el trabajo de los usuarios a la hora de organizar y visualizar la información de las entidades.

Capítulo 3: Validación de la solución propuesta

Introducción

En este capítulo se hace un análisis de los resultados obtenidos durante el desarrollo de la solución propuesta con la utilización de métricas de diseño TOC y RC, de usabilidad y pruebas de implementación de caja blanca. Para ello se siguieron una serie de métricas y pruebas para medir la calidad del diseño, de la implementación y la usabilidad del software las cuales se muestran a continuación.

3.1 Métricas de software

Las **métricas** aplicadas al diseño para la evaluación de la solución propuesta son las siguientes:

1. Tamaño operacional de la clase **(TOC):** Está dado por el número de métodos asignados una clase.

Atributo que afecta	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 en el grado de reutilización de la clase.

Tabla 6 Tamaño operacional de la clase (TOC)

2. Relaciones entre clases **(RC):** Está dado por el número de relaciones de uso de una clase con otras.

Atributo que afecta	Modo en que lo afecta
Acoplamiento	Un aumento del RC implica un aumento del Acoplamiento de la clase.
Complejidad del 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.

Tabla 7 Relaciones entre clases (RC)

*3.1.1 Resultados del instrumento de evaluación de la métrica Tamaño operacional de la clase (**TOC**)*

La gráfica siguiente muestra los resultados de la métrica Tamaño Operacional de la Clase (TOC), donde en la gráfica **cantidad de operaciones**, se mide el **porciento** de cantidad de procedimientos que contiene una clase. Como se puede apreciar existe casi un 60% de clases que contienen menos de 6 procedimientos y un 25 % que presentan menos de 10 operaciones, lo que significa un valor aceptable y satisfactorio para la métrica.

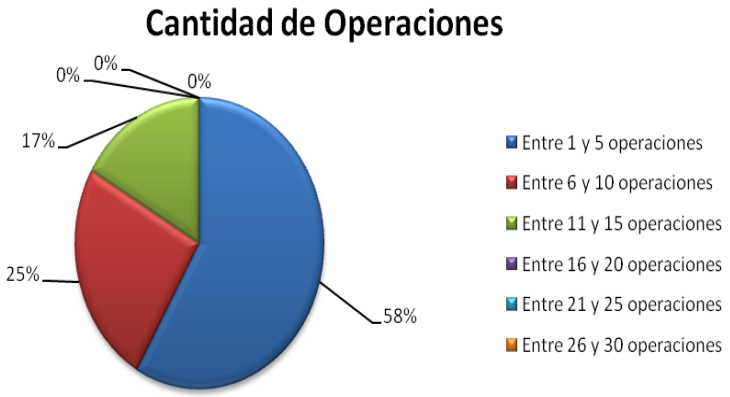


Fig. 21 Representación de los resultados obtenidos en el instrumento, agrupados en los intervalos definidos

A continuación se muestra la gráfica que mide el atributo Responsabilidad de la métrica, obteniéndose resultados satisfactorios mostrando una responsabilidad baja con un valor casi del 60%.

Fig. 22 Representación de la incidencia de los resultados de la evaluación de la métrica TOC para el atributo Responsabilidad

El atributo que se mide en la grafica que se muestra a continuación, después de haberse realizado la medición de la métrica, arrojo también resultados positivos ya que la complejidad de las clases es un 58% baja.

Fig. 23 Representación de la incidencia de los resultados de la evaluación de la métrica TOC para el atributo Complejidad

Al igual que el atributo complejidad la reutilización del código presenta resultados satisfactorios al brindar valores por encima del 50% en una reutilización alta.

Fig. 24 Representación de la incidencia de los resultados de la evaluación de la métrica TOC para el atributo Reutilización

3.1.2 Resultados del instrumento de evaluación de la métrica Relaciones entre Clases (RC)

La grafica siguiente muestra el resultado de la métrica Relación entre Clases (RC), donde se mide el porciento de cantidad de dependencias que contiene una clase. Como se puede apreciar existe más de un 60% de clases que no contienen dependencias y no alcanza el 30 % del total de clases que presenta solo una dependencia, lo que significa un valor aceptable y satisfactorio para esta métrica.

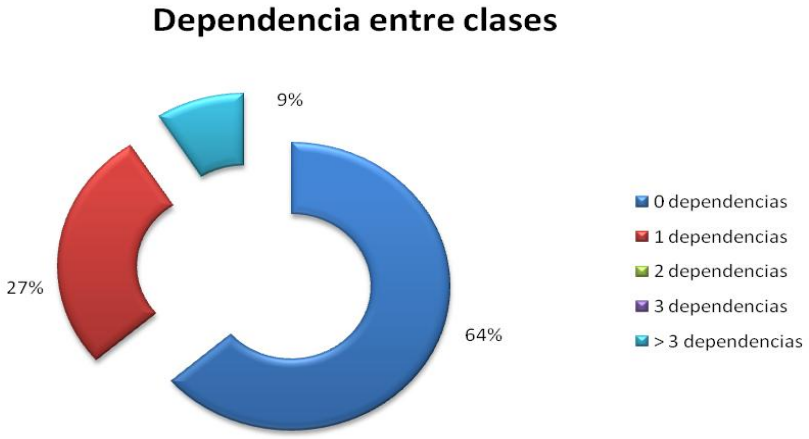


Fig. 25 Representación de los resultados obtenidos en el instrumento, agrupados en los intervalos definidos

El siguiente grafico muestra el resultado del atributo acoplamiento entre las clases del sistema, el cual presenta un 64% con valor ninguno, lo que quiere decir que las clases presentan una dependencia casi nula, lo que significa que la medición de la métrica está dando frutos satisfactorios.

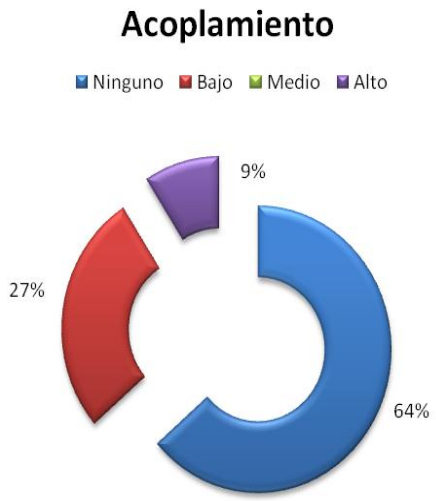


Fig. 26 Representación de la incidencia de los resultados de la evaluación de la métrica RC para el atributo Acoplamiento

Por otra parte los resultados del atributo complejidad de mantenimiento son bastantes satisfactorios, con un 91% de baja complejidad, lo que significa fácil soporte y poca complejidad para mantener el código.

Fig. 27 Representación de la incidencia de los resultados de la evaluación de la métrica RC para el atributo Complejidad de Mantenimiento

Luego de haber analizado varios parámetros, se llevo a cabo la medición de la variable cantidad de pruebas, el que brindó resultados satisfactorios al presentar un más de un 90% de bajo esfuerzo a la hora de realizar pruebas al diseño.

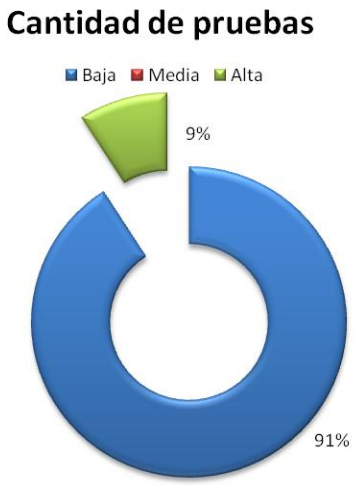


Fig. 28 Representación de la incidencia de los resultados de la evaluación de la métrica RC para el atributo Cantidad de Pruebas

Finalmente se realizó la medición del atributo **reutilización**, para comprobar el nivel de reutilización del diseño de cases, la métrica generó resultados positivos al informar que el diseño de la solución posee más del 90% de reutilización de sus clases.

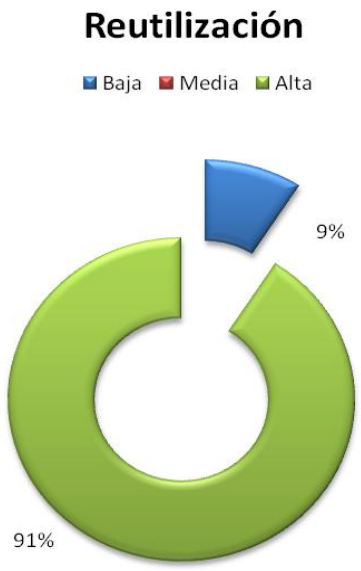


Fig. 29 Representación de la incidencia de los resultados de la evaluación de la métrica RC para el atributo Reutilización

En conclusión, la evaluación de los resultados obtenidos durante la aplicación del instrumento de medición de la métrica Relación entre Clases y Tamaño Operacional de la Clase, demuestran el uso de un buen diseño de software en el diseño de la nueva versión.

3.1.3 Métricas de Usabilidad

Para validar la variable usabilidad en el sistema se ha desarrollado el test de usabilidad que muestra los resultados que a continuación se verán. Para realizar la misma y verificar que la versión puede ser usada fácilmente por personas que no posean o no hayan tenido nunca un contacto directo con el subsistema Estructura y Composición, se seleccionaron un grupo de catorce usuarios, pertenecientes a diferentes roles dentro del centro, entre los que se pueden mencionar: especialistas de calidad y probadores, desarrolladores y arquitectos de datos.

Para la medición de la métrica se han seleccionado una serie de tareas que el usuario debe realizar con el objetivo de analizar estos datos luego para obtener finalmente los resultados. A continuación se especifican y describen las mismas:

- **Describir** cómo funciona la métrica para medir la usabilidad del producto.
- **Localizar** la dirección donde se encuentra la aplicación para el comienzo de la métrica.
- **Describir** brevemente lo que el producto hace y cómo se hace.
- **Adicionar** una subordinación.
- **Buscar** una subordinación dado un juego de datos por el cual se filtrará la misma. Los datos son (tipo de subordinación: económica, estado: inactiva, rango de fecha: entre el 10/05/2009 y el 25/4/2010).
- **Consultar** una subordinación dado un conjunto de datos por el cual se filtrará la misma. Los datos son (tipo de subordinación: Política, estado: activa, rango de fecha: entre el 10/05/2009 y la fecha actual). En caso de existir más de una subordinación una vez introducidos los datos anteriores entonces consultar a voluntad del usuario una de ellas, sino consultar la subordinación filtrada.
- **Activar** una subordinación dado un conjunto de datos por el cual se filtrará la misma. Los datos son (tipo de subordinación: Política, rango de fecha: entre el 10/05/2009 y la fecha actual). En caso de existir más de una subordinación una vez introducidos los datos anteriores entonces activar a voluntad del usuario una de ellas, sino activar la subordinación filtrada.
- **Listar** todas las subordinaciones existentes tanto las activas como las inactivas.

3.1.4 Herramienta Plantilla empleada para medir las variables de la métrica

La plantilla **USABILITY DATALOGGER v5.0** es una herramienta basada en Excel diseñada para ayudar a anotar observaciones y mediciones durante una evaluación de la usabilidad. (deinterfaz, 2010).

3.1.5 Resultados de la métrica

La forma más fácil de reflejar los resultados de la medición de métricas es haciéndolo a través de gráficos. Los datos y los gráficos ayudan a detectar dónde están los problemas, pero además, es tarea del experto saber detectar qué problemas existen en función de la observación que se haya hecho durante las

pruebas y el análisis de los datos. Los gráficos obtenidos de la herramienta datalogger son sencillos y por lo tanto muy fáciles de comprender. A continuación se muestran los mismos.

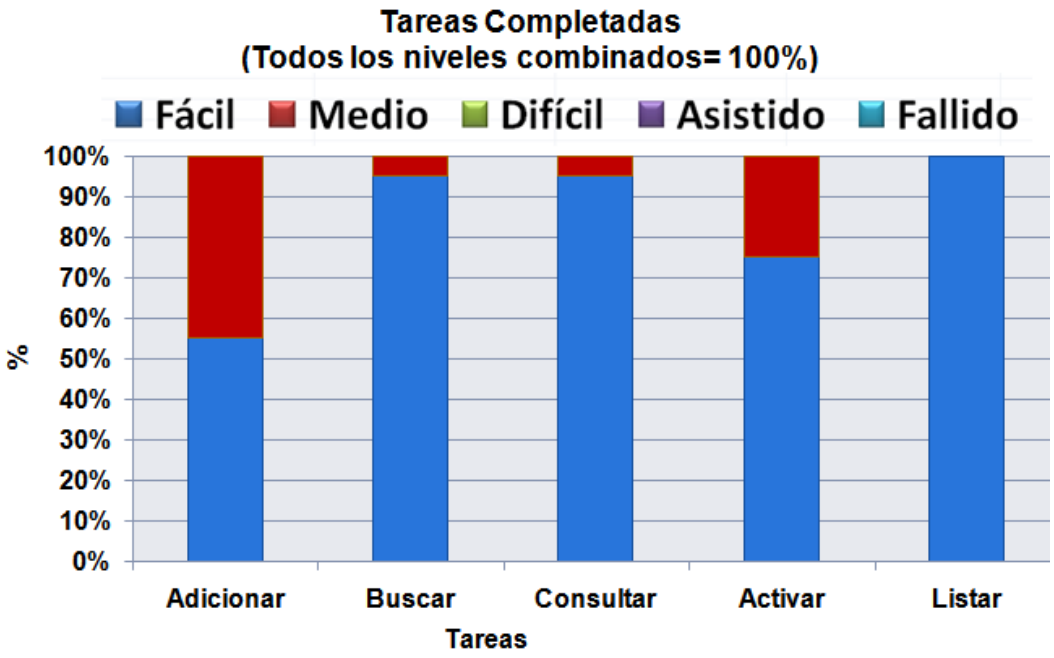


Fig. 30 Tareas completadas por los usuarios (todos los niveles de complejidad combinados)

El gráfico anterior muestra el porcentaje de tareas que fueron capaces de completar los usuarios midiendo todo los criterios que se ven en la parte superior (fácil, medio, difícil, asistido, fallido), donde se puede apreciar que todas las tareas se concluyeron a un 100% y los niveles de facilidad con que se completaron las mismas sobrepasan un 50%.

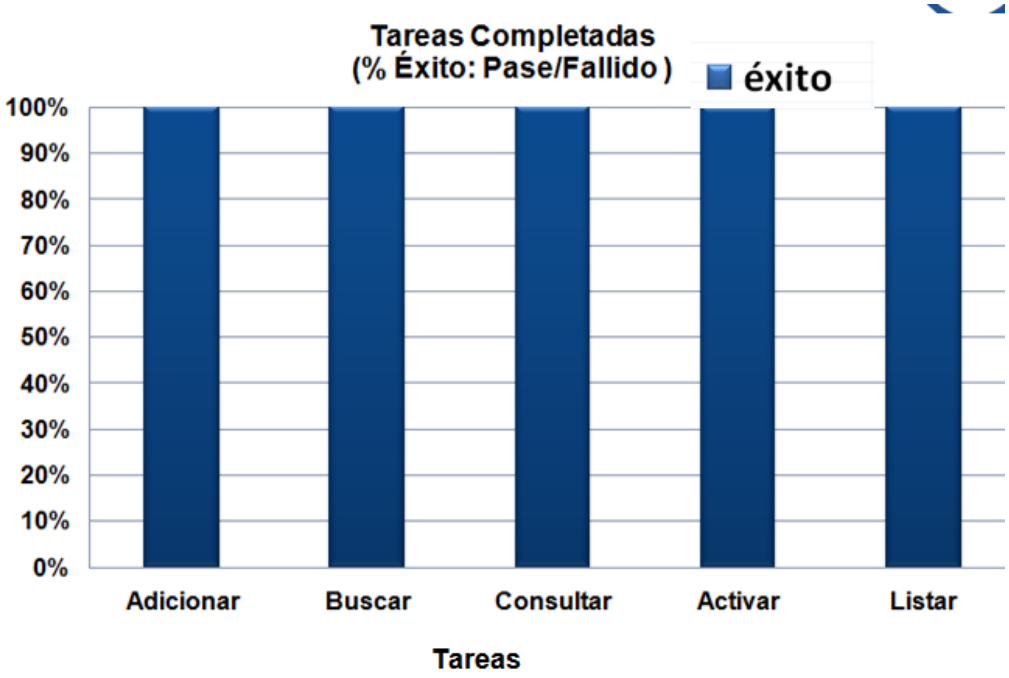


Fig. 31 Tareas completadas exitosamente por los usuarios

El gráfico anterior muestra la media de tareas completadas/fallidas (expresada en porciento) donde se ve claramente que todas las tareas se completaron exitosamente a un 100%.

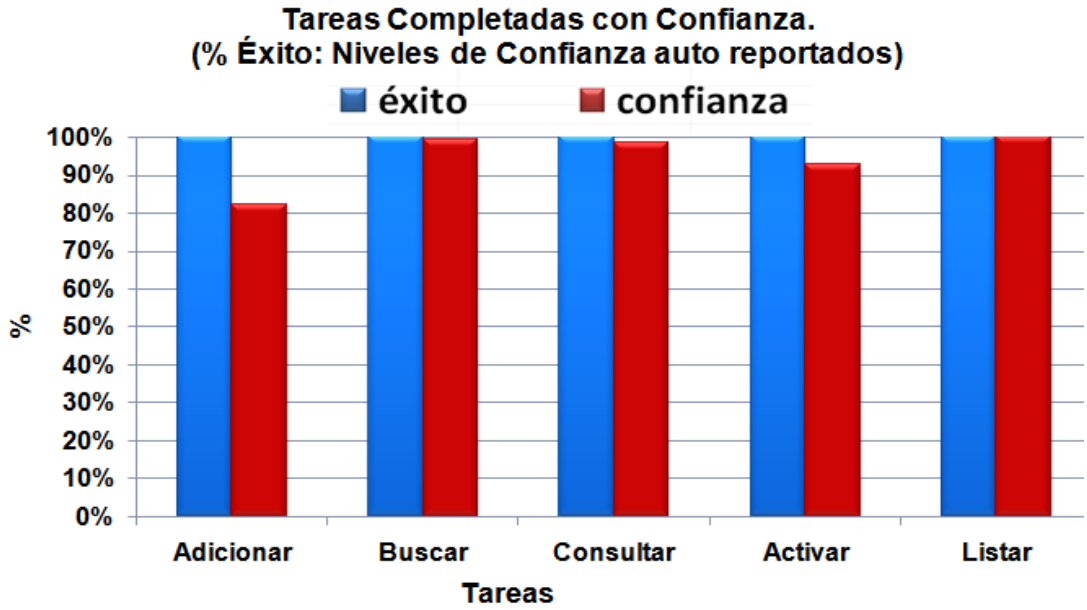


Fig. 32 Tareas completadas con confianza

La grafica anterior muestra las tareas completadas con éxito (barras azules) y las barras rojas el nivel de confianza con la que el usuario ejecutó las acciones para darle fin a la misma, como se puede apreciar los niveles de tareas concluidas con éxito están a un 100%, y la confianza con que se realizan las mismas sobrepasa el 80%, por lo que queda demostrado el nivel de facilidad con que el usuario navega por la aplicación.

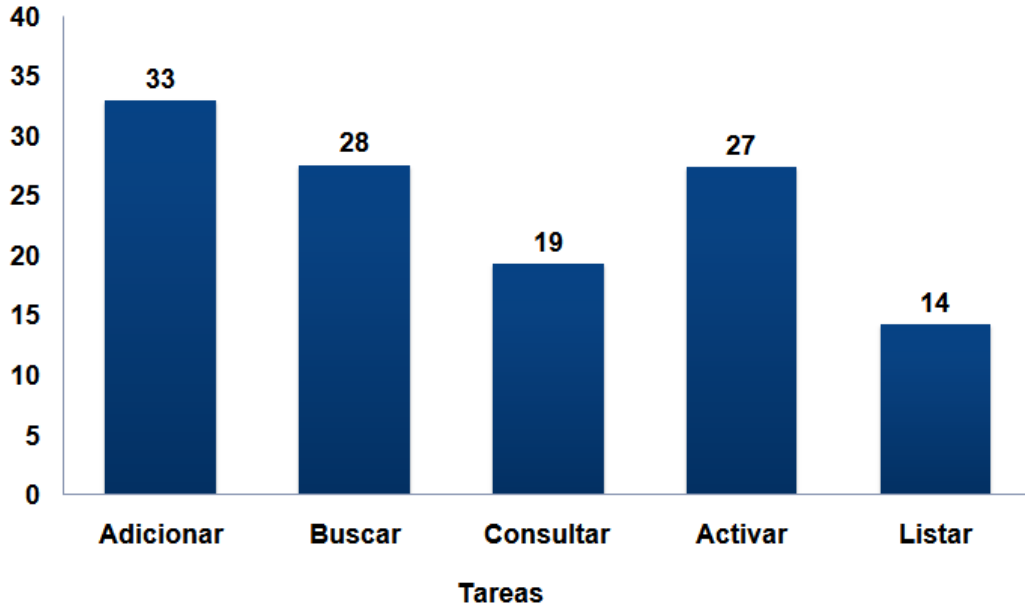


Fig. 33 Tiempo de demora en completar una tarea

En la gráfica anterior se muestran los resultados obtenidos del test de usabilidad donde se muestran en la parte superior de cada barra en la gráfica anterior, el promedio entre los tiempos máximos y los mínimos que se tarda un usuario para completar una tarea. Como se puede apreciar los resultados fueron satisfactorios ya que los tiempos en realizar estas tareas son pequeños.

3.1.6 Comparando los datos con el diseño anterior

Una vez detectados los problemas de usabilidad y habiéndolos solucionado en un rediseño, las mismas tareas serán más sencillas de llevar a cabo. Si se lleva a cabo un nuevo test sobre las tareas rediseñadas, se podrán ver los resultados al compararlas con el diseño viejo.

La siguiente tabla muestra un test de comparación de tiempo empleado para completar las tareas. Es importante aclarar que los resultados obtenidos, no son la representación del tiempo que demora la aplicación en concluir las tareas que se muestran, sino que representa el promedio entre los tiempos máximos y los mínimos que se tarda un usuario para completar la misma desde que comienza a trabajar en ella. Las siglas NI significan que las funcionalidades no están implementadas.

Tiempo empleado en realizar las tareas:

TEST DE TIEMPO		
TAREA	**DISEÑO ORIGINAL**	**REDISEÑO**
Adicionar Subordinación	160	33
Activar Subordinación	NI	27
Buscar Subordinación	NI	28
Consultar Subordinación	NI	19
Listar Subordinación	NI	14

Tabla 8 Test de tiempo para comparar los dos diseños

3.1.7 Prueba de caja blanca

A continuación se muestra el análisis de las pruebas de caja blanca para calcular los valores de la complejidad ciclomática de los procedimientos de la aplicación. Después de haber realizado el análisis de cada funcionalidad se pasa a registrar los mismos para llegar a conclusiones más exactas. Seguidamente se muestra el análisis realizado a las funcionalidades hacerCambios(), obtenerHijosPadre(), obtenerFecha(), tres de las funciones analizadas en el test de caja blanca de los cuales arrojaron los resultados que se muestran más adelante.

```
484
485 function hacerCambios($camino)
486     {
487         $deshacer = true;//1
488         foreach ($camino as $change)//2
489         {
490
491             if($change['intercepto'] )//3
492             $deshacer = false;//4
493             else
494             if($deshacer)//5
495             {
496                 $this->updatedatrow($change['idpadrenuevo'],//6
497                 $change['idpadreviejo'],$change['idhijo'],
498                 $change['idtiposub'],$change['fechaini']);
499             }//7
500             else
501             {
502                 $this->updatedatrow($change['idpadreviejo'],//8
503                 $change['idpadrenuevo'],$change['idhijo'],
504                 $change['idtiposub'],$change['fechaini']);
505             }//9
506         }
507
508
509
510     }//10
```

Fig. 34 Procedimiento hacerCambios()

Posteriormente se construye el grafo de flujo asociado al código anterior.

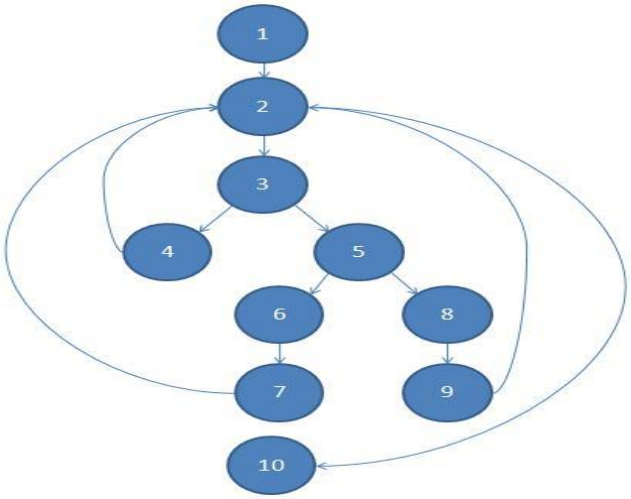


Fig. 35 Grafo de Flujo del procedimiento hacerCambios

Luego de haber construido el grafo, se realiza el cálculo de la complejidad ciclomática, mediante las tres fórmulas utilizadas para el análisis de complejidad del algoritmo, las cuales tienen que arrojar el mismo resultado para asegurar que el cálculo de la complejidad es correcto.

Fórmulas para calcular complejidad ciclomática.

1. V (G) = (A (Aristas) – N (Nodos)) + 2.

2. V (G) = P (Nodos Predicados) + 1.

3. V (G) = R.

Aplicando estas fórmulas al grafo de flujo de la figura 39 se obtienen los siguientes resultados:

Calculando mediante la fórmula 1:

V (G) = (12-10) + 2

V (G) = 4.

Calculando mediante la fórmula 2:

V (G) = 3 + 1

V (G) = 4

Calculando mediante la fórmula 3:

V (G) = 4

El cálculo efectuado mediante las fórmulas ha dado el mismo valor, por lo que se puede decir que la complejidad ciclomática del código es de 4, lo que significa que existen esa cantidad posible de caminos por donde el flujo puede circular, este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

Seguidamente es necesario representar los caminos básicos por los que puede recorrer el flujo. En estas representaciones se subrayan los elementos de cada camino que los hacen independientes a los demás.

Caso de prueba para el camino básico # 1.

Descripción: Los datos cumplirán con los siguientes requisitos: El arreglo $camino no posee ningún elemento.

Condición de ejecución: Que la cantidad de elementos del arreglo camino sea cero.

Entrada: camino []

Resultados esperados: No se realizará ningún cambio.

Resultados: No se hicieron cambios de subordinaciones porque no hubo ninguno

Caso de prueba para el camino básico # 2.

Descripción: Los datos de entrada cumplirán con los siguientes requisitos: El arreglo de cambios posee atributos con los cuales se podrá llegar a otras versiones.

Condición de ejecución: Que la cantidad de elementos del arreglo camino sea mayor que cero y que sea verdadero el atributo intercepto.

Entrada:
camino[{"idnomhistsubordinacion":"2","idpadrenuevo":"100000002","idtiposub":"1","idpadreviejo":"1000000
01","idhijo":"100000002","versionactual":"1","versionanterior":"0","fechaini":"2010-06-

13"},{"idnomhistsubordinacion":"1","idpadrenuevo":null,"idtiposub":"1","idpadreviejo":null,"idhijo":null,"versionactual":"0","versionanterior":"0","fechaini":"2010-06-13","intercepto":true},{"idnomhistsubordinacion":"4","idpadrenuevo":"100000002","idtiposub":"1","idpadreviejo":"100000001","idhijo":"100000002","versionactual":"3","versionanterior":"0","fechaini":"2010-06-13"}]

Resultados esperados: Se espera que el cambio que tenga el atributo, "intercepto": true no sea realizado.

Resultados: No se ha realizado el cambio que sea el intercepto.

Caso de prueba para el camino básico # 3.

Descripción: Los datos de entrada cumplirán con los siguientes requisitos: El arreglo de cambios posee atributos con los cuales se podrá llegar a otras versiones.

Condición de ejecución: Que la cantidad de elementos del arreglo camino sea mayor que cero.

Entrada:

camino[{"idnomhistsubordinacion":"2","idpadrenuevo":"100000002","idtiposub":"1","idpadreviejo":"100000001","idhijo":"100000002","versionactual":"1","versionanterior":"0","fechaini":"2010-06-13"},{"idnomhistsubordinacion":"1","idpadrenuevo":null,"idtiposub":"1","idpadreviejo":null,"idhijo":null,"versionactual":"0","versionanterior":"0","fechaini":"2010-06-13","intercepto":true},{"idnomhistsubordinacion":"4","idpadrenuevo":"100000002","idtiposub":"1","idpadreviejo":"100000001","idhijo":"100000002","versionactual":"3","versionanterior":"0","fechaini":"2010-06-13"}]

Resultados esperados: Se espera que se realicen los cambios que estén entre el intercepto y la versión a la que se quiere ir.

Resultados: Se han realizado satisfactoriamente los cambios que están entre la versión a la que se quiere ir y el intercepto.

Caso de prueba para el camino básico # 4.

Descripción: Los datos de entrada cumplirán con los siguientes requisitos: El arreglo de cambios posee atributos con los cuales se podrá llegar a otras versiones.

Condición de ejecución: Que la cantidad de elementos del arreglo camino sea mayor que cero.

Entrada:

camino[{"idnomhistsubordinacion":"2","idpadrenuevo":"100000002","idtiposub":"1","idpadreviejo":"100000001","idhijo":"100000002","versionactual":"1","versionanterior":"0","fechaini":"2010-06-13"},{"idnomhistsubordinacion":"1","idpadrenuevo":null,"idtiposub":"1","idpadreviejo":null,"idhijo":null,"versionactual":"0","versionanterior":"0","fechaini":"2010-06-13","intercepto":true},{"idnomhistsubordinacion":"4","idpadrenuevo":"100000002","idtiposub":"1","idpadreviejo":"100000001","idhijo":"100000002","versionactual":"3","versionanterior":"0","fechaini":"2010-06-13"}]

Resultados esperados: Se espera que se realicen los cambios que estén entre el intercepto y la versión actual.

Resultados: Se han realizado satisfactoriamente los cambios que están entre la versión actual y el intercepto.

```
280
281     public function obtenerHijosPadre($idPadre,$idsubordinacion)
282     {
283         $SQLwhere       = ($idPadre) ?"s.idpadre=$idPadre AND
284          s.idhijo<>s.idpadre":"s.idhijo=s.idpadre";//1
285         $q = Doctrine_Query::create();//2
286
287         $result         = $q->select('de.denominacion,
288         nf.idfila, s.idpadre, s.activa,false as leaf')//3
289         ->from('DatSubordinacion s ')
290         ->innerJoin('s.hijo nf')
291         ->innerJoin('nf.DatEstructura de')
292         ->where($SQLwhere)
293         ->addWhere('s.activa = ? and s.idnomsubordinacion = ?',array(1, $idsubordinacion))
294         ->orderBy('s.idhijo')
295         -> setHydrationMode ( Doctrine :: HYDRATE_ARRAY )
296         ->execute();
297         $array = array();//3
298         foreach ($result as $hijos) //4
299         {
300             $arreglohijo['id'] = $hijos['idhijo'].'#'.$hijos['idnomsubordinacion'];//5
301             $arreglohijo['text'] = $hijos['hijo'][0]['DatEstructura']['denominacion'];//5
302             $arreglohijo['leaf'] = $hijos['leaf'];//5
303             $arreglohijo['hijo'] = null;//5
304             $array[] =  $arreglohijo;//5
305         }
306         return $array;//6
307     }//7
```

Fig**. 36 Procedimiento obtenerHijosPadre()**

Posteriormente se construye el grafo de flujo asociado al código anterior:

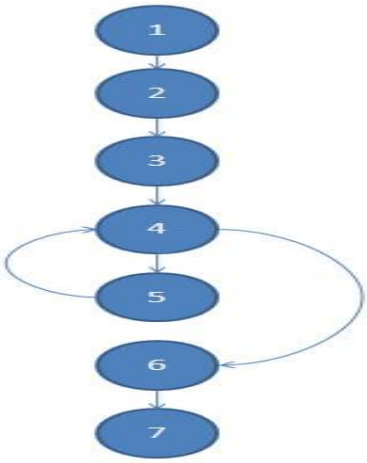


Fig. 37 Grafo de Flujo del procedimiento obtenerHijosPadre()

V (G) = (7-7) + 2

V (G) = 2.

Calculando mediante la fórmula 2:

V (G) = 1 + 1

V (G) = 2

Calculando mediante la fórmula 3:

V (G) = 2

Caso de prueba para el camino básico # 1.

Descripción: Los datos cumplirán con los siguientes requisitos: El arreglo que se obtenga al pasarle el id de la subordinación y de un padre no tendrá ningún elemento.

Condición de ejecución: Que la cantidad de elementos del arreglo sea cero.

Entrada: idpadre, idsubordinacion

Resultados esperados: Que no exista hijos que tengan ese id de padre hijos [].

Resultados: No se obtuvieron hijos con ese id.

Caso de prueba para el camino básico # 2.

Descripción: Los datos cumplirán con los siguientes requisitos: El arreglo que se obtenga al pasarle el id de la subordinación y de un padre tiene más de un elemento.

Condición de ejecución: Que exista al menos una estructura que tenga como id de padre el pasado.

Entrada: idpadre, idsubordinacion

Resultados esperados: Que devuelva un arreglo con la cantidad de estructuras que tienen como id de padre el que se le ha entrado. hijos [[{"id":"100000003#1","text":"PROCESAMIENTO DE SISTEMAS INDSUTRIALES","leaf": false,"hijo": null}]]

Resultados: Devolvió un arreglo con los hijos de una estructura.

```
242
243     function obtenerfecha($idsubordinacion,$idpadre,$idhijo)
244     {
245         $query  = Doctrine_Query::create();//1
246         if ($idsubordinacion) {//2
247             if ($idhijo) {//3
248                 if ($idpadre) {//4
249                     $result = $query->select('dt.fechaini')
250                     ->from('DatSubordinacion dt')
251                     ->where('dt.idnomsubordinacion = ?
252                     and dt.idhijo =? and dt.idpadre = ?',
253                     array($idsubordinacion,$idhijo,$idpadre))
254                     ->setHydrationMode(Doctrine::HYDRATE_ARRAY)
255                     ->execute();
256                     return $result;//5
257                 }//6
258             }//7
259         }//8
260         return 0;//9
261
262     }//10
```

Fig. 38 Procedimiento obtenerFecha()

Posteriormente se construye el grafo de flujo asociado al código anterior:

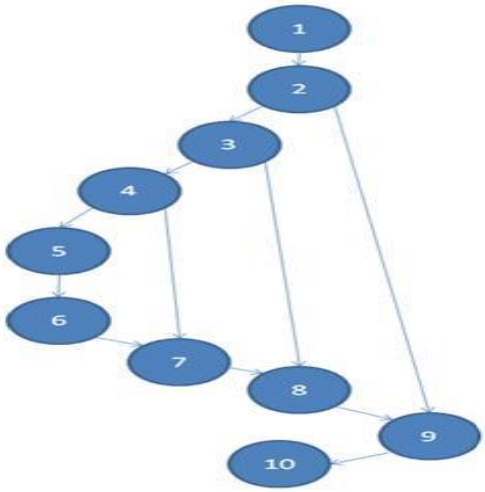


Fig. 39 Grafo de flujo del procedimiento obtenerFecha()

V (G) = (12-10) + 2

V (G) = 4.

Calculando mediante la fórmula 2:

V (G) = 3 + 1.

V (G) = 4.

Calculando mediante la fórmula 3:

V (G) = 4.

Caso de prueba para el camino básico # 1.

Descripción: Los datos cumplirán con los siguientes requisitos: No se le debe pasar id de la subordinación

Condición de ejecución: El id de la subordinación no debe estar definido.

Entrada: idpadre, idhijo

Resultados esperados: Devolverá cero

Resultados: El resultado fue el esperado

Caso de prueba para el camino básico # 2.

Descripción: Los datos de entrada cumplirán con los siguientes requisitos: No se le debe pasar id de la estructura padre.

Condición de ejecución: El idpadre no debe estar definido.

Entrada: idsubordinacion, idhijo

Resultados esperados: Se espera que resultado sea cero.

Resultados: Estuvo acorde con lo esperado.

Caso de prueba para el camino básico # 3.

Descripción: Los datos de entrada cumplirán con los siguientes requisitos: No se le debe pasar id de la estructura.

Condición de ejecución: El idhijo no debe estar definido.

Entrada: idsubordinacion, idpadre

Resultados esperados: Se espera que resultado sea cero.

Resultados: Estuvo acorde con lo esperado.

Caso de prueba para el camino básico # 4.

Descripción: Los datos de entrada cumplirán con los siguientes requisitos: Deben entrarse todos los parámetros

Condición de ejecución: Deben estar definido todos los parámetros de entrada.

Entrada: idsubordinacion, idpadre, idhijo

Resultados esperados: Se espera que devuelva la fecha de inicio de esa subordinación.

Resultados: Se ha devuelto correctamente lo esperado.

[[{"idpadre":"100000002","idhijo":"100000002","idnomsubordinacion":"1","fechaini":"2010-06-13"}]]

Conclusiones del Capítulo 3

- En este capítulo se aplicaron métricas dirigidas a evaluar la calidad del diseño del software, basadas en diferentes atributos de calidad que avalan sus resultados.
- Con la métrica Tamaño Operacional de Clase se evidencia que la implementación no es complicada, que la mayoría de las clases tiene una baja responsabilidad y que a su vez son muy reutilizables.
- Por su parte la de Relaciones entre Clases demuestra que la implementación del sistema tiene una calidad aceptable donde la mayor parte de las clases tienen 2 o menos dependencias, en cuanto al nivel de acoplamiento se comporta bajo por lo que es un factor aceptable desde el punto de vista de implementación del software.
- También se puede mencionar que la cantidad de pruebas y la complejidad para el mantenimiento se mantienen en niveles bajos según la métrica empleada, por lo que se demuestra el uso de un buen diseño de software.
- El diseño propuesto no presenta complejidad, es fácil de implementar y dar mantenimiento, tiene un bajo acoplamiento y es fácil de probar. También se realizaron métricas de usabilidad cumplen con las condiciones necesarias y garantiza un manejo fácil y de forma sencilla para los usuarios. Para demostrar el funcionamiento del código fuente, se aplicaron pruebas de caja blanca. Todo esto permitió verificar que las funcionalidades implementadas responden a las necesidades y propósitos de los clientes y que la aplicación esta lista para su explotación.

Conclusiones generales

- Luego de culminar el presente trabajo se puede concluir que los objetivos planteados en un inicio fueron alcanzados satisfactoriamente.
- Se logró la obtención de una nueva versión del subsistema Estructura y Composición, la cual ayudará al usuario a definir y organizar mejor la estructura organizativa de la empresa, garantizando que el usuario tenga el conocimiento suficiente de las relaciones existentes en cada una de las entidades que maneja, posibilitando el cubrimiento de los objetivos propuestos para mitigar los problemas de usabilidad que presentaba dicho subsistema.
- Para ello se abordó en el marco teórico conceptual asociado al objeto de estudio y campo de acción en cuestión.
- Posteriormente fue diseñada e implementada una solución apropiada en aras de dar solución a los objetivos propuestos. Finalmente se realizó un análisis con el propósito de hallar los mecanismos adecuados que permitieron llevar a cabo una evaluación del resultado los que resultaron ser positivos en todos los sentidos que se evaluaron.

Recomendaciones

El equipo de desarrollo recomienda implementar las interfaces de usuario relacionadas con el nomenclador de subordinación la cual permitirá al usuario crear un tipo de subordinación así como visualizar los tipos existentes. Otra recomendación, es integrar esta solución al Sistema de Gestión Integral de Entidades CEDRUX, dentro del framework actual del subsistema Estructura y Composición, ya que este marco de trabajo está implementado sobre una arquitectura mejorada y actualizada y la solución está desarrollada en un marco de trabajo un poco desactualizado.

Bibliografía

Runbaugh, James, Jacobson, Ivar y Booch, Gradi. 1998. *El Lenguaje Unificado de Modelado. Manual de Referencia.* 1998.

2008. visual-paradigm.com. [En línea] 2008. www.visual-paradigm.com.

White, Stehpen A y Miers, Derek. 2008. *BPMN Modeling and Reference Guide.* 2008.

Korsh, James F. and Garrett, Leonard J. Data Structures, Algorithms and Program Style Using C. *Entorno Virtual del Aprendizaje(EVA).* [Online] [Cited: 3 4, 2010.] http://eva.uci.cu/file.php/461/Libros/KorshGarret_-_Data_Structures_Algorithms_and_Program_Style_Using_C/TOC.HTM.

apache.org. 2010. http://www.httpd.apache.org/. *http://www.httpd.apache.org/.* [En línea] 2010. http://www.httpd.apache.org/.

Pressman, Roger S. 2005. *Ingenieria del Software un enfoque practico.* La Habana : Felix Varela, 2005. págs. 171-187.

Aptana, inc. 2010. *.aptana.com.* [En línea] 2010. http://www.aptana.com/.

Bergin, Joseph. 2010. *Building Graphical User Interfaces with the MVC pattern. [Online] 2 4, 2010. http://csis.pace.edu/ bergin/mvc/mvcgui.html.*

Burbeck, Steve. 2009. *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC). [Online] 10 3, 2009. http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.*

Deacon, John. 2010. *Model-View-Controller (MVC) Architecture. [Online] 1 20, 2010. http://www.jdl.co.uk/briefings/mvc.pdf.*

Departamento Central de Técnicas de Programación Universidad de las Ciencias Informáticas(UCI). Estructuras de Datos No Lineales. *Entorno virtual de Aprendizaje.* [En línea] [Citado el: 9 de 4 de 2010.] http://eva.uci.cu/mod/resource/view.php?id=20131.

doctrine-project organization development team. 2010. www.doctrine-project.org. *doctrine-project.* [En línea] 2010. http://www.doctrine-project.org/.

EAV Modeling. **Welch, Carl, et al. 2008.** [ed.] Steph Fox, et al. 6, Toronto : php|architect, Junio 2008, php|architect, Vol. 7.

Extjs, In. 2010. http://www.extjs.com/. *http://www.extjs.com/.* [En línea] 2010. http://www.extjs.com/.

php.net. 2010. *php.net.* [En línea] 2010. http://www.php.net/.

postgresql.org. 2010. http://www.postgresql.org/. *http://www.postgresql.org/.* [En línea] 2010. http://www.postgresql.org/.

subversion.apache.org. 2010. http://subversion.apache.org/. *http://subversion.apache.org/.* [En línea] 2010. http://subversion.apache.org/.

Sun Microsystem. 2008. *visual-paradigm.com.* [En línea] 2008. www.visual-paradigm.com.

Torres, Patricio Orlando Letelier y María del Carmen Panadé. 2010. [En línea] 2010. http://dialnet.unirioja.es/servlet/articulo?codigo=1983605&info=resumen&modo=popup.

Zend Company. 2010. *www.framework.zend.com.* [En línea] 2010. http://www.framework.zend.com.

http://latecladeescape.com. 2009. http://latecladeescape.com/w0/ingenieria-del-software/metodologias-de-desarrollo-del-software.html. *http://latecladeescape.com.* [En línea] 25 de 1 de 2009. [Citado el: 22 de 3 de 2010.] http://eva.uci.cu/file.php/102/Curso_2008-2009/Materiales_Complementarios/Materiales_Complementarios_Conf_1/2009-2010/materiales.rar.

Ivar Jacobson, Grady Booch y James Rumbaugh. 2000. *El proceso unificado del desarrollo del software.* Madrid : Pearson Education S.A, 2000. 84-7829-036-2.

Dpto. Central de Técnicas de Programación UCI. 2008. Estructuras de Datos No Lineales. *Entorno virtual de Aprendizaje.* [En línea] 2008. [Citado el: 9 de 4 de 2010.] http://eva.uci.cu/mod/resource/view.php?id=20131.

Departamento de Técnicas de Programación (UCI). 2010. El TDA Grafo. 2010. http://www.deinterfaz.com, 2010. http://www.deinterfaz.com/referencias/ .

http://www.deinterfaz.com/medir-metricas-de-usabilidad.htm. http://www.deinterfaz.com/. [En línea] http://www.deinterfaz.com/ , 2010. [Citado el: 13 de 5 de 2010.] http://www.deinterfaz.com/medir-metricas-de-usabilidad.htm.