UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
FACULTAD 3

"DESARROLLO DE LA CAPA DE PRESENTACIÓN DE LOS MÓDULOS ADMINISTRACIÓN Y NOMENCLADORES, PARA EL SISTEMA DE GESTIÓN DE INVENTARIO Y ALMACÉN: SIGIA"

TRABAJO PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS.

AUTOR:

Carlos Felipe Pérez León.

TUTORES:

Ing. Diosmani Meriño Hechavarría.
Lic. Arnaldo Utria Azan.

La Habana
Julio, 2007.


"La ciencia es la combinación de la voluntad, el amor y la dedicación."
CF.

DECLARACIÓN DE AUTORÍA

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

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

_____	_____	_____
Carlos Felipe Pérez León	Lic. Arnaldo Utria Azan	Ing. Diosmani Meriño Hechavarría

AGRADECIMIENTOS

A mis padres y mi hermana, por todo su apoyo.

Al arquitecto del SIGIA, José Raul, por su importante ayuda.

A todos los programadores y analistas del SIGIA, en especial a mis amigos Junier, Jorge, Elian, Danis, Figueroa, Dietmar, Grethel y Mayuli.

A mis tutores y amigos.

A la mujer que: *en Su-sana discresión, hallé mis sueños*.

DEDICATORIA

A la memoria del Comandante Ernesto Ché Guevara.

A mis padres y mi hermana.

A mi tía Conchi, mi tío Humbe, mis abuelas y todos mis primos.

A Susana.

RESUMEN

El presente trabajo de diploma expone la estructura de la capa de Presentación para los módulos: “Administración y Nomencladores” del SIGIA y muestra el resultado del trabajo del programador, acorde con la planificación para el cumplimiento de la implementación y prueba, de cada interfaz de usuario.

Las interfaces de usuario desarrolladas para los módulos Administración y Nomencladores del SIGIA deben ser funcionales, amigables, rápidas y que cumplan con los requerimientos del sistema, integradas a otras unidades funcionales del mismo. Una implementación eficiente de estas interfaces de usuario, va a contribuir en gran medida, al éxito del software.

PALABRAS CLAVE

- ***Interfaz:*** Conexión física y funcional entre dos aparatos o sistemas independientes.
- ***Capa:*** Conjunto de subsistemas que comparten la misma generalidad y de volatilidad de interfaces. Este término hace referencia a la forma como una solución es segmentada desde el punto de vista lógico: Presentación/ Lógica de Negocio/ Datos.
- ***Implementación:*** Acción de implementar. Este término se refiere a la forma en que se van a codificar las interfaces.

ÍNDICE

INTRODUCCIÓN

El mundo, hoy en día, se ve inmerso en un gran desarrollo científico-técnico, donde las Tecnologías de la Información y las Comunicaciones (TICs) desempeñan un papel trascendental. Estas tecnologías se han convertido en una herramienta al servicio de las estrategias de las empresas, posibilitando un mayor carácter competitivo, así como la producción de una nueva gama de sistemas que van a satisfacer las demandas de rentabilidad, desarrollo sostenible y calidad en cada uno de sus procesos.

En la producción de los distintos sistemas informáticos para la gestión de los procesos empresariales con calidad y acorde con las necesidades de la entidad, hay que destacar que el encargado de gestionar los inventarios a los almacenes de la empresa es uno de los más importantes, porque permite, entre otras funcionalidades, el control de los recursos tangibles así como información fehaciente que favorecerá la toma de decisiones por parte de los directivos y todo ello va a contribuir con el mejor funcionamiento y manejo de los recursos, en función del desarrollo de la empresa y del país.

En nuestro país se acometen una serie de proyectos, regidos en su mayoría, por la Universidad de las Ciencias Informáticas (*UCI*) y entre los más relevantes se encuentra un nuevo Sistema de Gestión de Inventario y Almacén *(SIGIA)*, que se ajuste a las actuales legislaciones económicas y que pueda ser utilizado por cualquier empresa, con el fin de erradicar los problemas que presentan en cuanto: la rentabilidad, que en su mayoría son de organización y control, precisamente por carecer de un sistema como este, que agilice todos estos procesos.

En la construcción de los sistemas informáticos, en la actualidad, se aboga por la utilización de alguna de las metodologías de desarrollo de software, que sirven para dirigir, organizar y controlar el proceso de desarrollo de estos sistemas y brindan mayores posibilidades de cumplimentar el trabajo de forma eficiente y con calidad. Para la elaboración de este sistema en específico, en función de resolver los problemas existentes, se ha empleado la metodología: el Proceso Unificado de Rational que en el inglés original es *Rational Unified Process (RUP)*, donde se definen un conjunto de roles que van a conformar el equipo de desarrollo del software. Entre los roles se encuentra el rol de arquitecto, que es aquella persona encargada de definir la arquitectura que se va a utilizar para desarrollar el sistema y en este caso, se emplean dos estilos arquitectónicos

que va a dar solidez a la misma, como eje central del sistema: el *Modelo Vista Controlador* y la *Arquitectura en Capas*.

En el modelo vista controlador, la vista es el medio donde se manipula toda la información de forma visual, el controlador es el encargado de interpretar todas las acciones sobre el sistema e informa tanto a la vista como al modelo para que actúen en consecuencia y el modelo es el encargado de administrar tanto los datos como el comportamiento de la aplicación. Por otra parte, la arquitectura en capas está sustentada por tres capas fundamentales: la capa de Acceso a Datos, la capa intermedia o capa de Negocio y la capa de Presentación que en este caso se divide en dos, una capa de Interfaz de Usuario y otra capa de Acciones y que además van a ser desarrolladas por los programadores, otro de los roles que define RUP. [Perera, 2007]

Para el desarrollo eficiente de cada una de estas capas, el programador debe utilizar los artefactos elaborados por el arquitecto, los analistas y los diseñadores, estos dos últimos, son roles que también define la metodología. Entre los principales artefactos que necesita el programador de la capa de Presentación, se encuentran: el *Modelo de Datos,* el *Modelo de Diseño,* el *Prototipo de Interfaz.*

Teniendo en cuenta todo lo expuesto, podemos definir el **problema:**
¿Cómo implementar un conjunto de interfaces que sean amigables, funcionales y rápidas, que además respondan a las necesidades de los módulos “Administración” y “Nomencladores” del SIGIA y se integren con la capa de Negocio desarrollada?

Objeto de estudio: Capa de Presentación para el SIGIA.

Para lograr todo ello, se plantea como **objetivo de la investigación:** Implementar un conjunto de interfaces funcionales, amigables y rápidas, en la capa de Presentación de los módulos “Administración” y “Nomencladores” para el SIGIA, que se integren con la capa de Negocio desarrollada.

Campo de acción: Capa de Presentación de los módulos “Administración” y “Nomencladores” para el SIGIA.

Las **tareas** para desarrollar la investigación, son las siguientes:

1. Valorar y analizar el estado del arte del rol de programador fundamentalmente en el desarrollo de la capa de Presentación de los principales sistemas de gestión.
2. Utilizar el Proceso Personal del Software o en inglés, Personal Software Process (PSP) para lograr un aprovechamiento máximo del tiempo que se dispone para implementar las interfaces.
3. Desarrollar un conjunto de métodos en las clases de la capa de Acciones, que favorezcan las validaciones y el trabajo con los distintos tipos de variables, brindando mayor eficiencia y uniformidad en la implementación de las interfaces.
4. Analizar los requisitos que deben cumplir las interfaces acorde con la descripción de los casos de usos y el prototipo de interfaz.
5. Implementar las interfaces en la capa de Presentación de los módulos “Administración” y “Nomencladores” para el SIGIA.
6. Integrar la capa de Presentación con la capa de Negocio desarrollada para el SIGIA utilizando la filosofía de programación multicapa.
7. Desarrollar las pruebas, con los casos de uso de prueba correspondiente, que posibilitarán la validación de las interfaces.

Para dar cumplimiento a las distintas tareas antes mencionadas, se pusieron en práctica, los siguientes **métodos:**

Métodos teóricos:

- *Analítico – sintético:* este método posibilita el procesamiento de toda la información y poder sintetizar y diferenciar cada una de ella, enfocada hacia la investigación, además de poder determinar las características que debe tener el desarrollo de la capa de Presentación.
- *Inductivo – deductivo:* permite elaborar los estándares, tanto de codificación y diseño gráfico, como las clases a utilizar por las interfaces y a su vez, determinar las especificaciones de cada una de ellas.
- *Histórico – lógico:* este método posibilita conocer los antecedentes y tendencias actuales del las interfaces gráficas y del desempeño de un programador de la capa de Presentación, principalmente en los software de gestión.

- *Sistémico:* este método facilita la integración de cada uno de los elementos desarrollados, en la conformación del objeto.

Métodos empíricos:

- *Medición:* este método posibilita establecer comparaciones en el desarrollo del PSP del programador de la capa de Presentación.
- *Experimento:* este método favorece el desarrollo de las pruebas para la validación de las interfaces.

Aportes prácticos que se esperan del trabajo:

El trabajo desarrollado debe dar la posibilidad de contar con:

- La implementación y certificación de un conjunto de interfaces funcionales, amigables y rápidas, para la capa de Presentación del SIGIA.
- Una integración de la capa de Presentación con otras unidades funcionales desarrolladas para el SIGIA.
- La implementación de componentes que posibiliten reusabilidad y calidad a estas interfaces.

Estructura de la tesis:

La tesis se encuentra estructurada en tres capítulos.

En el Capítulo 1 se aborda el surgimiento y desarrollo de la interfaz gráfica en los principales software de gestión de inventario. Se presenta además, el resultado de un estudio realizado sobre las principales técnicas o estilos de programación que se utilizan tanto en la UCI, como en el país y a nivel internacional. También se exponen un conjunto de conceptos que acercarán al lector, hacia las principales tareas que debe realizar un programador, enfocadas fundamentalmente en los lineamientos que rigen su desarrollo dentro del proyecto productivo. Se explica la estrategia de investigación y se hace una breve crítica y valoración, de las técnicas, plataforma, librerías y procedimientos utilizados para dar cumplimiento a las tareas del programador.

En el Capítulo 2 se expone la estructura que presenta la capa de Presentación en integración con la capa de Negocio. Se analizan y describen cada uno de los artefactos desarrollados por el programador en cada uno de los Casos de Uso (CU) implementados, que conforman los módulos Administración y Nomencladores, así como los distintos algoritmos utilizados y la eficiencia del programador determinada por el trabajo con el PSP.

En el Capítulo 3 se presentan los resultados de las pruebas de validación de interfaces realizadas a cada uno de los CU implementados por el programador en la capa de Presentación de los módulos Administración y Nomencladores.

Además de poseer Conclusiones Generales, Recomendaciones, Referencias bibliográficas, Bibliografía y Anexos.

1

Capítulo 1: Fundamentación Teórica.

1. 1 Introducción al capítulo.

En este capítulo se realiza un estudio acerca del surgimiento y desarrollo de la interfaz gráfica, fundamentalmente en los principales software de gestión de inventario, así como de las principales técnicas o estilos de programación que se utilizan tanto en la UCI, como en el país y a nivel internacional.

Además se exponen un conjunto de conceptos de las terminologías, que representan las principales tareas que debe realizar un programador dentro de un proyecto productivo. Se realiza una breve crítica y valoración, de las técnicas, plataforma, librerías y procedimientos utilizados para dar cumplimiento a las tareas del programador.

1. 2 ¿Qué es una Interfaz Gráfica de Usuario (GUI, Graphical User Interface)?

El término *interfaz*, proviene del inglés *interface* y pudiera definirse, de forma general, como el medio que posibilita la interacción entre dos elementos para que puedan trabajar juntos. Sin embargo, en la esfera de la informática, hay que destacar que existen diversos tipos de interfaces que trabajan a distintos niveles. Existen las interfaces que hacen posible el trabajo del software con el hardware, es decir, que posibilita a los sistemas operativos el funcionamiento con los dispositivos de hardware, tanto interno como externos. También existe la interfaz de línea de comandos para interpretar las sentencias introducidas por el usuario, la interfaz controlada por menús, que le brinda al usuario mayores funcionalidades para la interpretación de combinaciones de teclas en el trabajo con la aplicación y la interfaz gráfica de usuario o GUI, que además de controlar las opciones de los menús, también posibilita controlar la posición, el tamaño y el contenido de las áreas de trabajo, en pantalla. [Stephanidis, 2001]

La GUI es definida por Lewis y Rieman de la siguiente forma:

"Las interfaces gráficas de usuario son aquellas que incluyen cosas como menús, ventanas, teclado, ratón y algunos otros sonidos que la computadora hace, en general, todos aquellos canales por los cuales se permite la comunicación entre el hombre y la computadora." [Lewis-Rieman, 1993]

También existen otras definiciones tales como:

"…tipo de entorno que permite al usuario elegir comandos, iniciar programas, ver listas de archivos y otras opciones utilizando las representaciones visuales (iconos) y las listas de elementos del menú…" [Stephanidis, 2001]

"…es aquella parte de un programa que comunica al usuario con el programa mediante representaciones gráficas…" [Stephanidis, 2001]

La definición más acertada para esta terminología, según la opinión del autor, acorde con la esfera informática y el desarrollo de software, es la siguiente:

"La Interfaz Gráfica de Usuario es un tipo de interfaz que mediante el uso del lenguaje visual, sustentado por un conjunto de imágenes y componentes gráficos, muestra en la pantalla una información específica así como un conjunto de acciones posibles, posibilitando a los usuarios una interacción con un sistema informático."

Surgimiento y desarrollo de las GUIs.

La idea de una interfaz gráfica de usuario para un computador u ordenador, surge incluso, mucho antes de que la tecnología estuviera disponible para ello. El Memex (Memory Expander) era una máquina que Vannevar Bush había ideado, pero nadie pudo materializarlo y en síntesis, posibilitaba la búsqueda y reproducción de archivos microfilmados y daba además la posibilidad a los usuarios de tomar anotaciones en los márgenes. Luego, en 1945 Bush retoma sus ideas que inspiraron a Douglas Engelbart, quien construyó la máquina. [Bush, 1945]

Douglas recibe el apoyo por parte de las Fuerzas Aéreas de los Estados Unidos e inventa el ratón del ordenador y la primera interfaz gráfica en 1960. Culminó su trabajo en 1968

con una demostración ante miles de personas y es considerado como el padre de la Interfaz Gráfica de Usuario. [Reimer, 2005]

Esa interfaz gráfica creada por Douglas en los laboratorios de Xerox, fue introducida en las computadoras de Apple - Macintosh en 1984 y no fue hasta el 1993 y dado por la necesidad de eliminar muchas de las barreras existentes hombre – ordenador, que se llevó a la población, con la primera versión popular de Windows 3.0 como sistema operativo. [Reimer, 2005]

En la actualidad la GUI es un elemento fundamental para la producción de software, porque refleja el rostro del mismo y en cierto grado, delimita su carácter competitivo. Las GUIs de los sistemas de gestión de inventarios, son interfaces muy amigables, rápidas, funcionales y flexibles, porque su misión fundamental es brindar mayores facilidades a los usuarios en el trabajo con el sistema. Ejemplo de ello tenemos el sistema francés Geode GX, perteneciente a la Sage Adonix, líderes en el mercado de aplicaciones especializadas en gestión logística, con instalaciones en más de 350 plantas en Francia, Europa y Norteamérica; también existen otros sistemas importantes producidos por los italianos, mexicanos y argentinos, como el WMS (Warehouse Management Software) de la System, el ADMAN de la Siman (Sistemas Maestros de Negocios) y el sistema ISIS de la ALIANZA DATAHOUSE-COMPANY respectivamente.

Características de las GUIs.

En la actualidad, una buena interfaz gráfica de usuario debe contar con un conjunto de características que podrían sintetizarse en:

1. Facilidad de comprensión, aprendizaje y uso.
2. Representación fija y permanente de un determinado contexto de acción.
3. Facilidad de identificación del objeto de interés.
4. Diseño funcional, mediante el establecimiento de menús, barras de acciones e iconos de fácil acceso.
5. Las interacciones se basarán en acciones físicas sobre elementos de código visual o auditivo (iconos, botones, imágenes, mensajes de texto o sonoros, barras de desplazamiento y navegación...) y en selecciones de tipo menú con sintaxis y órdenes.

6. Las operaciones serán rápidas, incrementales y reversibles, con efectos inmediatos.
7. Existencia de herramientas de Ayuda y Consulta.
8. Tratamiento del error bien cuidado y adecuado al nivel de usuario.

1. 2. 1 Desarrolladores de las GUIs.

Para desarrollar cada uno de las capas que definen los estilos arquitectónicos del SIGIA, la metodología RUP define grupos de trabajadores: *Analistas, Desarrolladores, Probadores, entre otros,* así como un conjunto de artefactos y actividades a realizar, que precisan las responsabilidades de estos trabajadores. Entre los que clasifican como desarrolladores se encuentran los *Programadores,* que son los encargados de implementar y probar los componentes que necesita el software, acorde con los estándares adoptados por el proyecto, para la integración en subsistemas más grandes. [RUP, 2003]

Es preciso que el programador reúna un conjunto de características que le posibilite personalizar su trabajo, sin aislarse de las guías adoptadas para el mismo. La originalidad y creatividad, son aspectos claves para desempeñar este trabajo, así como también lo son la capacidad de superación con facilidad, la investigación de forma independiente, la capacidad de analizar y observar los sucesos dentro y fuera de su entorno de trabajo y sobre todo, la aplicación de la lógica en todo momento.

Según RUP:
Las actividades fundamentales que realiza el programador son:

- Implementar los elementos de diseño y prueba.
- Desarrollar pruebas de implementación y ejecución.
- Analizar el comportamiento de componentes en tiempo de ejecución.

Los artefactos principales a desarrollar por el programador son:

- Elementos y subsistemas de implementación.
- Elementos de prueba.
- Pruebas *Stub*.[RUP, 2003]

Según el Instituto de Ingeniería Eléctrica y Electrónica (IEEE):

"Stub es una implementación de un módulo de software con un objetivo específico, utilizado para desarrollar o probar un módulo que depende de este." [IEEE, 1990]

Para que las interfaces gráficas de usuario reúnan todas las características que le posibiliten la obtención del éxito al software que representan, deben ser implementadas cuidadosamente. Con frecuencia estas implementaciones se realizan en un tiempo considerablemente corto, pero si no se tienen en cuenta aspectos como patrones de diseño, reusabilidad, seguridad y optimización, entre otros, las interfaces se tornan poco eficientes. En esta capa los programadores deben hacer gala de todas sus habilidades y conocimientos, porque de ello depende en un por ciento elevado, la aceptación del software.

1. 3 Principios básicos de la programación.

La programación, en términos informáticos, se muestra como la codificación de las órdenes y datos que permiten la creación de un programa o aplicación, pero antes de llevar a cabo la programación de una aplicación o sistema, es importante que el programador realice una serie de actividades que posibiliten su desarrollo de forma eficiente. En todo proceso de construcción o elaboración de un producto, existe un elemento fundamental, donde se van a trazar los lineamientos para lograr el objetivo del proceso: el *diseño del producto*.

En el caso de los programas y aplicaciones, el diseño consta de distintas etapas de desarrollo. La realización de un **análisis** exhaustivo de cada una de las situaciones tanto explícitas como implícitas que contenga el problema a resolver. El **diseño** y creación o utilización de la base de la programación: **los algoritmos**, apoyado en el análisis. Una vez que se tienen las bases del programa, se procede a realizar la **codificación** o **implementación** de los algoritmos en un lenguaje de programación y reflejando las ideas desarrolladas en las etapas de análisis y diseño. Posterior a la codificación, es preciso que el programa sea llevado al lenguaje de la máquina, para su ejecución, es decir, que sea **interpretado** o **compilado** y **ejecutado**. [Figueroa, 2005]

Durante el desarrollo de las etapas anteriores pueden existir errores que serán buscados y eliminados en las etapas de **verificación y depuración**, etapas fundamentales para la culminación de un programa eficiente, así como la **documentación** o etiquetado del programa, para hacerlo más comprensible. [Figueroa, 2005]

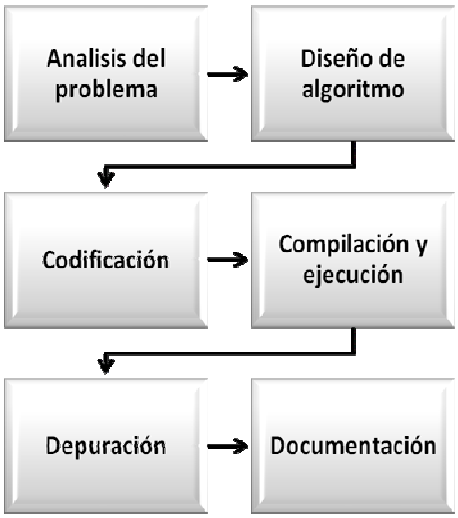


Figura 1. 1: Esquema de las etapas para el diseño de un programa.

Existe una octava etapa que está sujeta principalmente a programas o aplicaciones de mayor envergadura, es decir, que presentan mayor grado de complejidad, tanto en la programación como en la utilización o vinculación con otros sistemas. Este tipo de programas o aplicaciones necesitan, en la mayoría de los casos, de un **mantenimiento** donde puedan realizarse nuevos ajustes y modificaciones, siempre teniendo en cuenta que no deben alterar el funcionamiento de los mismos. [Figueroa, 2005]

1. 3. 1 Algoritmos y Estructuras de datos.

El diseño de algoritmo, es la etapa donde se va a resolver el problema en cuestión, aunque no por sí solo, sino con la aplicación correcta de todas las operaciones que son dictadas por el algoritmo.

La palabra *algoritmo* se deriva del nombre del gran matemático árabe Mahommed ibn Musa al-Jwarizmi (c. 780-c. 835), debido a sus avances en el trabajo con el álgebra y los algoritmos, con fines matemáticos. Es preciso destacar que los algoritmos no se centran en una ciencia o materia específica, son aplicables a la mayoría de las acciones que realiza el hombre. [Cormen-Leiserson-Rivest-Stein, 2001]

Existen diversas definiciones de algoritmo entre las cuales se encuentran:

"...es cualquier procedimiento computacional bien definido, que toma algún valor, o juego de valores, como la entrada y produce algún valor, o juego de valores, como el

rendimiento. Un algoritmo es así una sucesión de pasos que transforman la entrada en el rendimiento." [Cormen-Leiserson-Rivest-Stein, 2001]

"Un algoritmo es un juego finito de instrucciones que, si se siguen, logra cumplimentar una tarea particular. " [Pinson - Wiener, 2000]

Teniendo en cuenta todos estos criterios sobre los algoritmos, se puede definir a un algoritmo como un método sustentado por un conjunto de instrucciones finitas, que utilizando un grupo de elementos de entradas, produce otro grupo de elementos resultantes para dar solución a un problema determinado, independientemente del lenguaje de programación que se utilice para desarrollarlo.

Características de los algoritmos:

- Debe tener un punto particular de inicio.
- No debe ser ambiguo.
- Debe ser general.
- Debe ser finito en tamaño y en tiempo de ejecución.
- Legible y sencillo de modificar-ampliar.
- Eficiente, definido ("n ejecuciones - igual resultado") y óptimo. [Horta-Ostos-Gutiérrez, 1991]

Es importante destacar que los algoritmos, como eje central de la metodología de la programación, frecuentemente demandan una compleja organización de sus datos y una de las formas más eficiente y organizada de manipular estos datos es precisamente con la utilización de las estructuras de datos. Las estructuras de datos van a facilitar todo un marco lógico para el trabajo de los datos en función del problema que se vaya a tratar y el algoritmo para resolverlo.

Se considera que la definición más acertada de una estructura de datos y a la cual se adscribe el autor, es la siguiente:

"…una manera de almacenar y organizar los datos de forma ordenada, para facilitar el acceso y las modificaciones a los mismos. Es importante tener conocimiento de las ventajas y desventajas de cada una de las estructuras de datos, para enfocar su uso eficiente en un problema, acorde con sus características, porque no todas las estructuras

de datos funcionan bien con todos los tipos de problemas." [Cormen-Leiserson-Rivest-Stein, 2001]

Entre las estructuras de datos de uso más frecuente, se encuentran los Arreglos, Listas Enlazadas, Pilas, Colas, Árboles y Grafos. Cada una de ellas con sus respectivas especificaciones y variantes.

Las estructuras de datos y los algoritmos están indisolublemente unidos. Es importante identificar y comprender correctamente los algoritmos que se vayan a utilizar en la solución a un problema determinado, para poder seleccionar y ajustar en caso necesario, estructuras de datos que organicen apropiadamente el trabajo.

1. 4 Paradigmas y Lenguajes de programación.

La palabra *paradigma* proviene del griego *paradeigma*, que significaba originalmente: *ejemplo ilustrativo* y fue el filósofo Thomas Samuel Kuhn (1922-1996), quien extendió la definición de la palabra:

"Considero a los paradigmas como realizaciones científicas universalmente reconocidas que, durante cierto tiempo, proporcionan modelos de problemas y soluciones a una comunidad científica". [Fuente, 2000]

Los paradigmas de programación según Bobrow y Stefik son: *una forma de organizar programas sobre las bases de algún modelo conceptual de programación y un lenguaje apropiado para que resulten claros los programas escritos en ese estilo.* [Bobrow-Stefik, 1986]

Según R. Floyd, existen tres tipos de paradigmas:

a) *Los que soportan técnicas de programación de bajo nivel.*
b) *Los que soportan métodos de diseño de algoritmos.*
c) *Los que soportan soluciones de programación de alto nivel.*

Teniendo en cuenta que un paradigma va a definir un conjunto de reglas, patrones y estilos de programación que son usados por los lenguajes de programación que usan ese paradigma y acorde con los tipos de paradigmas definidos por Floyd, se pueden destacar cuatro paradigmas fundamentales:

1. ***Paradigma funcional:*** *en este paradigma el proceso computacional se lleva a cabo a través de la evaluación de expresiones.*

   Características:

   - *Definición de funciones.*
   - *Funciones como datos primitivos.*
   - *Valores sin efectos laterales, no existe la asignación.*
   - *Programación declarativa.*

2. ***Paradigma lógico:*** *en este paradigma la programación trata de representar el conocimiento mediante relaciones (predicados) entre objetos (datos).*

   Características:

   - *Definición de reglas.*
   - *Unificación como elemento de computación.*
   - *Programación declarativa.*

3. ***Paradigma imperativo o procedural:*** *los lenguajes que cumplen con este paradigma presentan un estado implícito que es modificado mediante instrucciones o comandos del lenguaje.*

   Características:

   - *Definición de procedimientos.*
   - *Definición de tipos de datos.*
   - *Chequeo de tipos en tiempo de compilación.*
   - *Cambio de estado de variables.*
   - *Pasos de ejecución de un proceso*

4. ***Paradigma orientado a objetos:*** *este paradigma es uno de los más empleados en la actualidad. Define los programas en términos de clases y objetos.*

   Características:

   - *Definición de clases y herencia.*

- *Objetos como abstracción de datos y procedimientos.*
- *Polimorfismo y chequeo de tipos en tiempo de ejecución.* [Floyd, 1979]

Es preciso destacar que no hay un paradigma de programación que sea el mejor para todo tipo de aplicaciones. Por ejemplo, la programación orientada a reglas sería la mejor para el diseño de una base de conocimiento, y la programación orientada a procedimientos sería la más indicada para el diseño de operaciones de cálculo intensivo. Actualmente uno de los paradigmas más utilizado tanto a nivel internacional como en nuestro país, es el orientado a objetos, porque teniendo en cuenta las abstracciones que implementa este paradigma de programación, se puede afirmar que se adecua, en mayor grado, a una gran diversidad de aplicaciones. [Booch, 1996]

Los paradigmas de programación que se emplean en el desarrollo del SIGIA son: Paradigmas *Orientado a Objetos y Orientado a Aspectos,* este último, como una alternativa a la solución de algunos de esos problemas que provocarían una disminución de la calidad final del software, como lo son la abstracción y el encapsulamiento de conceptos, que no forman parte de la funcionalidad básica de los sistemas, tales como debugging, sincronización, distribución, seguridad, administración de memoria, y otros.

Los lenguajes y los paradigmas de programación se encuentran estrechamente relacionados. De la misma forma en que existen paradigmas distintos, con ideas comunes, existen lenguajes que soportan más de un paradigma.

Según la definición de la Encyclopedia of Computer Science (Encyclopedia of Computer Science, 4th Edition, Anthony Ralston (Editor), Edwin D. Reilly (Editor), David Hemmendinger (Editor), Wiley, 2007. Disponible en la biblioteca politécnica con identificador: POE R0/E/I/ENC/RAL):

"A programming language is a set of characters, rules for combining them, and rules specifying their effects when executed by a computer, which have the following four characteristics:

1. *It requires no knowledge of machine code on the part of the user*
2. *It has machine independence*
3. *Is translated into machine language*

4. *Employs a notation that is closer to that of the specific problem being solved than is machine code."*

Es decir:

"Un lenguaje de programación es un conjunto de caracteres, reglas para combinarlos, y reglas que especifican sus efectos cuando son ejecutados por una computadora, que tienen las cuatro características siguientes:

1. *No requieren del conocimiento del código de la máquina para la parte de los usuarios.*
2. *Tiene independencia de la máquina.*
3. *Se traduce en el idioma de la máquina.*
4. *Empleos de una notación que se acerca al problema específico a resolverse, que es el código de la máquina."*

También es preciso destacar ideas como las de Abelson y Sussman acerca de los lenguajes de programación:

"A powerful programming language is more than just a means for instructing a computer to perform tasks. The language also serves as a framework within which we organize our ideas about processes. Thus, when we describe a language, we should pay particular attention to the means that the language provides for combining simple ideas to form more complex ideas."

"Un lenguaje de programación poderoso es más que un simple medio para instruir a una computadora en la realización de las tareas. El lenguaje también sirve como un medio donde podemos organizar nuestras ideas sobre los procesos. Así, cuando nosotros describimos un lenguaje, debemos prestar particular atención a los recursos que brinda el lenguaje, para combinar ideas simples con ideas más complejas." [Abelson - Sussman, 1996]

Según la definición de los doctores en ciencias Montse Ibañez y Germán Gutiérrez y a la cual se adscribe el autor, un lenguaje de programación es: un tipo de lenguaje que permite comunicar conceptos entre dos elementos fundamentales (máquina-humanos) y como cualquier lenguaje consta de:

- ***Alfabeto:*** caracteres alfanuméricos (alfabéticos + numéricos) y otros especiales como **=**, **+**, **-** , ***** , **(** , **)** , **.** , **:** , **{** , etc.
- ***Léxico:*** vocabulario

  - Palabras predefinidas (reservadas)
  - Resto palabras: reglas léxicas (identificadores válidos)

- ***Sintaxis:*** reglas gramaticales que determinan de manera estricta como formar las instrucciones (sentencias).

  - Errores sintácticos: el programa no compila y/o no se puede interpretar

- ***Semántica:*** Significado tanto del léxico como de las instrucciones. El programa, al ejecutarlo, produce el resultado deseado. [Ibañes – Gutiérrez, 1998]

Entre los años 1954 y 2000, se documentaron alrededor de 2500 lenguajes de programación, pero de ellos hubo una decena que influyeron en el desarrollo de los lenguajes posteriores y que se elaboraron entre 1954 y 1972, es decir, en la primera época de los lenguajes de programación. Anexo I

Los lenguajes de programación se dividen en tres grandes grupos: los lenguajes de máquina, que son aquellos escritos en código binario, los de bajo nivel o ensamblador, que son escritos en nemotécnicos y los de alto nivel que han sido diseñados para lograr una mejor comprensión en la escritura y lectura de programas. [Bonanata, 2003]

Entre los lenguajes de alto nivel, que han posibilitado la creación de aplicaciones, con mayores prestaciones y facilidades en su desarrollo, apoyándose en una característica fundamental que es la independencia entre ellos y la arquitectura de la máquina u ordenador; podemos destacar al Visual Basic, C, C++, Delphi, CSharp y Java, destacándose en mayor medida C, CSharp y Java. En la actualidad, nuestra universidad consta de un conjunto de proyectos productivos donde un gran número de aplicaciones, son desarrolladas con la utilización de estos lenguajes.

1. 5 ¡Para desarrollar el SIGIA!

El desarrollo del SIGIA, atendiendo a la programación, se ve sustentado por el estilo de programación ***Orientado a Objetos*** y un lenguaje que cumple con las características de este estilo, como lo es ***Java***. Una plataforma que en este caso es la ***J2SE*** o Java 2 Platform Standard Edition, y que está conformada por una Interface de Programación de Aplicaciones ***(API)*** para este lenguaje y un Kit de Desarrollo para Java que en este caso es el ***JDK v6.0*** *(Java Development Kit)* y que además incluye la Máquina Virtual de Java (JVM).

"Conocido originalmente como Kit de desarrollo de Java (JDK), el Kit de desarrollo de software para Java 2 (SDK), es lo que se usa para crear programas para Java 2 Platform Standard Edition (J2SE)" [Zukowsky, 2003] Anexo II

Un entorno de desarrollo integrado o *integrated development environment* (IDE), como lo es el ***Eclipse v3.2***, donde el lenguaje a utilizar, pueda ejecutarse correctamente. Además se le agregan algunos programas (plug-ins), al IDE, para aumentar sus funcionalidades y potencializar de manera eficiente, los frameworks a utilizar en el desarrollo de la programación multicapa. Estos frameworks se definen como el diseño reutilizable de todo o parte de un sistema, representado por un conjunto de componentes abstractos, y la forma en la que dichos componentes interactúan [Fayad et al., 1999]. Para la implementación de la capa de Presentación con el framework ***Swing*** específicamente, los plug-ins que se agrega son los del ***WindowBuilder Pro v5.0***. Además se realizan pruebas de validaciones de interfaces, para certificar cada uno de los formularios que se desarrollan en esta capa.

Todos estos elementos fueron seleccionados atendiendo las necesidades que debe suprimir el sistema en desarrollo, la complejidad que presenta este sistema y las características que debe cumplir el mismo, definidas por el arquitecto. [Perera, 2007].

1. 5. 1 La Programación Orientada a Objetos (POO) y Java.

Antes de definir aspectos que determinen lo que llamamos como POO, es preciso entender: **¿qué es un objeto?**

Un objeto, en términos computacionales, puede definirse según Armando Canchala, como: *Componente o código de software que contiene en sí mismo tanto sus*

características (campos) como sus comportamientos (métodos); se accede a través de su interfaz o signatura. [Canchala, 2006]

El objeto es algo que tiene fronteras claramente definidas, pero ello no basta para diferenciar un objeto de otro, por tanto: "Un objeto tiene estado, comportamiento e identidad; la estructura y comportamiento de objetos similares están definidos en su clase común; los términos instancia y objeto son intercambiables" [Booch, 1996].

Según el autor: un objeto es la clave para entender cualquier lenguaje orientado a objetos. Se encuentra bien delimitado y representa algo, de forma específica, determinando su identidad, estado y comportamiento.

La POO presenta algunas características peculiares, pero una de las cosas que más la distinguen es que proporciona conceptos y herramientas con las cuales se modela y representa el mundo real tan fielmente como sea posible.

Según Luis Armando Canchala Fernández, diplomado en POO, la programación orientada a objetos es: *una técnica para desarrollar soluciones computacionales utilizando componentes de software.* [Canchala, 2006]

Según la definición de Grady Booch, a la cual se adscribe el autor: *La tecnología orientada a objetos se apoya en los sólidos fundamentos de la ingeniería, cuyos elementos reciben el nombre global de* ***modelo de objetos****. El modelo de objetos abarca los principios de abstracción, encapsulación, modularidad, jerarquía, tipos, concurrencia y persistencia. Ninguno de estos principios es nuevo por sí mismo. Lo importante del modelo de objetos es el hecho de conjugar todos estos elementos de forma sinérgica.* [Booch, 1996]

El modelo de objetos, como marco de referencia del estilo de POO, abarca los principios anteriormente mencionados, como expone Grady Booch, pero de ellos, hay cuatro que son imprescindibles para que el modelo sea orientado a objetos, ellos son:

- ***Abstracción:*** denota las características esenciales de un objeto, que lo distinguen de todos los demás tipos de objetos y proporciona así fronteras conceptuales nítidamente definidas respecto a la perspectiva del observador.

- ***Encapsulamiento:*** es el proceso de almacenar en un mismo compartimento los elementos de una abstracción que constituyen su estructura y su comportamiento; sirve para separar el interfaz contractual de una abstracción y su implantación.
- ***Modularidad:*** es la propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y débilmente acoplados.
- ***Jerarquía:*** La jerarquía es una clasificación u ordenación de abstracciones. Entre los tipos de jerarquías más importantes se encuentra: *la jerarquía de clases*. [Booch, 1996]

Los principios secundarios que abarca el modelo de objeto, tales como:

- ***Tipos (tipificación):*** son la puesta en vigor de la clase de los objetos, de modo que los objetos de tipos distintos no pueden intercambiarse o, como mucho, pueden intercambiarse sólo de formas muy restringidas.
- ***Concurrencia:*** se centra en la abstracción de procesos y la sincronización de los mismos mediante la programación en paralelo.
- ***Persistencia:*** es la propiedad de un objeto por la que su existencia trasciende el tiempo.[Booch, 1996]

Otros conceptos que caracterizan la POO y que están estrechamente relacionados con los cuatro principios fundamentales del modelo de objetos, son:

- ***Clase:*** es la estructura para la creación de los objetos. Se define como contenedora de los atributos (campos y métodos) de los objetos.
- ***Herencia:*** consiste en que una clase puede heredar sus variables y métodos a varias subclases (la clase que hereda es llamada clase hija). De esta manera se crea una ***jerarquía*** de herencia, que en esta tecnología, se considera dos formas: simple y múltiple.
- ***Polimorfismo:*** propiedad que indica que un elemento puede tomar distintas formas. Podemos definirlo como el uso de varios ***tipos*** en un mismo componente o función.
- ***Envío de Mensajes:*** los mensajes son el medio para la colaboración entre los objetos, que de estar aislados, se torna muy poco útiles. Son invocaciones a los métodos de los objetos.

En el lenguaje de programación “Java”, es preciso destacar que la herencia, como ejemplo más fehaciente de la jerarquía de clases, en la clasificación de múltiple, no existe, es decir, que una clase sólo puede tener una superclase directa. Aunque se utiliza una alternativa para resolver esta problemática y no es más que la creación de *Interfaces*, donde encontramos una colección de definiciones de métodos solamente, cuya implementación se desarrolla en otras clases de esta jerarquía; pero hay elementos que define la herencia múltiple, que no se cumplen con esta alternativa, ya que no se pueden heredar variables desde un interface ni implementaciones de métodos. Anexo III

Por otra parte, la clase *Object* es la superclase de todas las clases de *Java*. Todas las clases derivan, directa o indirectamente de ella. Si al definir una nueva clase, no aparece la cláusula *extends*, Java considera que dicha clase desciende directamente de *Object*. [Arnold - Gosling, 1997]

Seguridad en Java.

El lenguaje Java fue diseñado con las siguientes ideas en mente:

- Evitar errores de memoria.
- Imposibilitar acceso al SO.
- Evitar que caiga la máquina sobre la que corre.

y para lograr la fiabilidad y robustez de las aplicaciones desarrolladas con este lenguaje, basada en el cumplimiento de estas ideas, se desarrolló la primera línea de defensa que atiende todos los problemas que fundamentalmente, C y C++, presentan respecto a las siguientes características:

- Ausencia de punteros.
- Gestión de memoria.
- Recogida de basura.
- Arrays con comprobación de límites.
- Referencias a objetos fuertemente tipadas.
- Casting seguro.
- Control de métodos y variables de clases.
- Métodos y clases ***final***.

Además, se crearon otras dos líneas de defensa, para eliminar posibles problemas que se generen con la utilización del *código bytes* y el *cargador de clases*, así como un *gestor y un cúmulo de restricciones de seguridad*. [Marañón, 1998]

Java, al igual que otros lenguajes orientados a objetos, brinda un conjunto de ventajas como son:

1. Fomenta la reutilización y extensión del código.
2. Permite crear sistemas más complejos.
3. Relacionar el sistema al mundo real.
4. Facilita la creación de programas visuales.
5. Construcción de prototipos.
6. Agiliza el desarrollo de software.
7. Facilita el trabajo en equipo.
8. Facilita el mantenimiento del software. [Marañón, 1998]

1. 5. 2 J2SE y Eclipse v3.2.

Java 2 Plataform, Standard Edition es una de las tecnologías básicas para muchos estilos diferentes de desarrollo de software, incluidos applets, aplicaciones clientes y aplicaciones servidores individuales. Java es, desde su diseño, un lenguaje capaz de superar las diferencias que hay entre los ordenadores de una red heterogénea, al ser independiente del sistema operativo.

La última versión de J2SE (1.6 ó 6.0) incorpora nuevas funciones, un mayor rendimiento y mejoras en el manejo de las interfaces de usuario, que representan un paso adelante para la tecnología Java. El cambio del número de la versión radica en darle un carácter de madurez a las tecnologías de Java. Con esta versión, se podrá utilizar la tecnología Java para desarrollar aplicaciones más complicadas con menos esfuerzo y en menos tiempo. Permite, al igual que versiones anteriores, aplicaciones escalables y de gran rendimiento para su implantación en cualquier plataforma. [Lindsey- Tolliver- Lindblad, 2005]

Los IDEs o entornos de desarrollo integrado, son programas conformados por un conjunto de herramientas útiles para el programador y que pueden estar enfocados a un lenguaje de programación específico o a varios lenguajes.

El IDE va a ofrecer el ambiente o entorno de desarrollo, como bien su nombre lo indica, para desarrollar aplicaciones complejas, apoyado en las facilidades que brinda un JDK en el proceso de compilación. Además posibilita la creación de WAR's (Web-Archives) así como un ambiente gráfico con acceso a herramientas que los JDKs no contienen. [Lago, 2006]

Entre los IDE's más importantes para desarrollar proyectos en Java, tenemos:

- ***Eclipse:*** Open-Sourse.
- ***NetBeans:*** Open-Sourse.
- ***JBuilder:*** de Borland.
- ***JDeveloper:*** de Oracle.
- ***Sun Java Studio:*** de Sun.
- ***Microsoft Visual J++:*** para aplicaciones java en ***.Net***
- ***WebSphere Studio:*** de IBM.

"Eclipse es un proyecto de desarrollo de software de código fuente abierto (open source) cuyo objetivo es la construcción de herramientas integradas para el desarrollo de aplicaciones." [Lago, 2006]

Eclipse es una gran estructura formada por un núcleo y muchos plug-ins que van conformando la funcionalidad final. Estos plug-ins interactúan mediante interfaces o puntos de extensión; así, las nuevas aportaciones se integran sin dificultad ni conflictos. Este IDE a pesar de que está escrito en su mayoría, en Java (excepto el núcleo) y que su uso más popular sea como un IDE para Java ejecutándose sobre máquina virtual de ésta, es neutral y flexible a cualquier tipo de lenguaje, por ejemplo C/C++, Cobol, C#, XML, etc. [Puebla, 2006]

La arquitectura de este IDE es sumamente fuerte y está conformada por cinco elementos fundamentales:

- ***Núcleo:*** determina cuales son los plug-ins disponibles en el directorio de plug-ins de Eclipse. Es recomendable agregar solamente los plug-ins que se vayan a utilizar en el momento, para su mejor funcionamiento

- ***Entorno de trabajo:*** maneja los recursos del usuario, organizados en uno o más proyectos. Cada proyecto corresponde a un directorio en el directorio de trabajo de Eclipse, y contienen archivos y carpetas.
- ***Interfaz de usuario:*** muestra los menús y herramientas. En los proyectos de Java puede usarse AWT y Swing con solo agregar los plug-ins para Eclipse.
- ***Ayuda al grupo:*** plug-in que facilita el uso de un sistema de control de versiones para manejar los recursos en un proyecto del usuario y define el proceso necesario para guardar y recuperar de un repositorio. Eclipse incluye un cliente para un Sistema de Control de Versiones (CVS).
- ***Documentación:*** al igual que el propio Eclipse, el componente de ayuda es un sistema de documentación extensible. Los proveedores de herramientas pueden añadir documentación en formato HTML y, usando XML, definir una estructura de navegación. [Puebla, 2006]

Se destaca que el Eclipse, como plataforma universal para la integración de herramientas de desarrollo, brinda soporte para (AWT/SWING), WEB (Servlets, jsp, ect) y otros frameworks; con la utilización de su filosofía de componentes añadidos o plug-ins. Además, se puede ejecutar en diferentes sistemas operativos como Windows, Mac OSX y Linux, entre otros, teniendo en cuenta la necesidad de que la computadora conste de un procesador de 200 MHz o superior, 256 MB de memoria RAM y un Entorno de Ejecución de Java (JRE) o JDK en una versión 1.4 o superior. [Clayberg, 2003]

1. 5. 3 Swing y WindowBuilder Pro v5.0.

Swing es la solución actual que brinda Java, para la creación de las GUIs. Es un paquete que forma parte de la Java Foundation Class (JFC) y abarca un grupo de componentes que se identifican porque pertenecen al paquete *javax.swing,* como botones, tablas, marcos, etc…

Antes de la existencia de Swing, el manejo de las interfaces gráficas de usuario, se realizaba con AWT (Abstract Window Toolkit), de quien Swing hereda todo el manejo de eventos, es por eso que la mayoría de los programas Swing necesitan importar dos paquetes AWT*: java.awt.** y *java.awt.event.**, aunque para todos los componentes de AWT, existe un componente Swing, que lo remplaza y que siempre comienza con “J”. [Calderón - Davis, 2000]

Ejemplo: clase ***Button*** de AWT es reemplazada por la clase ***JButton*** de Swing.

Es preciso enfatizar en la filosofía que plantea Swing, donde expone dos reglas de suma importancia para el desarrollo de una aplicación con este paquete:

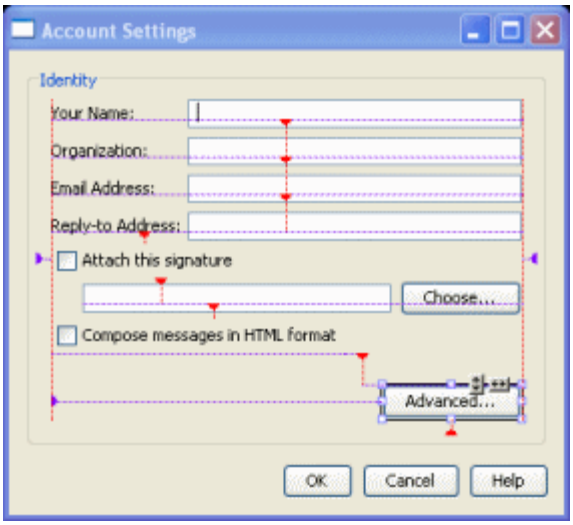
1. Debe existir, al menos, un contenedor de alto nivel (*Top-Level Container*), que provee el soporte que las componentes Swing necesitan para el pintado y el manejo de eventos.
2. Otras componentes colgando del contenedor de alto nivel (éstas pueden ser contenedores o componentes simples). [Calderón - Davis, 2000] Anexo IV

Cumpliendo con los principios de la programación multicapa, Swing es el framework seleccionado por el arquitecto, para crear las GUIs que necesita el SIGIA y así cumplir con el aspecto *Vista,* dentro del Modelo Vista Controlador (MVC), patrón de arquitectura utilizado en la capa de Presentación; en total integración con la capa de Negocio.[Perera, 2007]

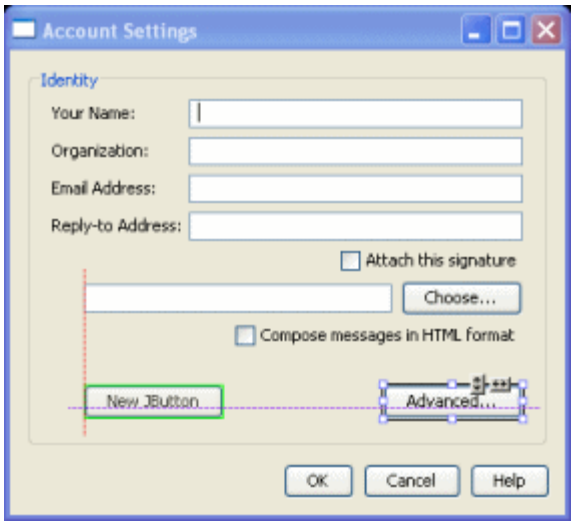
El WindowBuilder Pro, como herramienta para el diseño visual, en su versión 5.0 soporta las tecnologías Swing, Google Web Toolkit (GWT) y Eclipse Standard Widget Toolkit (SWT), otro conjunto de herramientas para construir interfaces gráficas en Java, que retoma la idea original de AWT. Brinda grandes facilidades en el desarrollo de las GUIs en Java, así como aplicaciones Eclipse Rich Client Platform (RCP) y GWT/Ajax ("*Asíncrono Javascript y XML*"), sin emplear mucho código de escritura.

A continuación se muestran algunos ejemplos del trabajo de forma sencilla y rápida en la creación de las interfaces, gracias a las tecnologías que soporta esta herramienta, así como la configuración de la paleta de componentes para diseñar las interfaces.

Ejemplo 1: GroupLayout es uno de los componentes que permite ubicar el resto de los componentes dentro del formulario, en el lugar deseado.

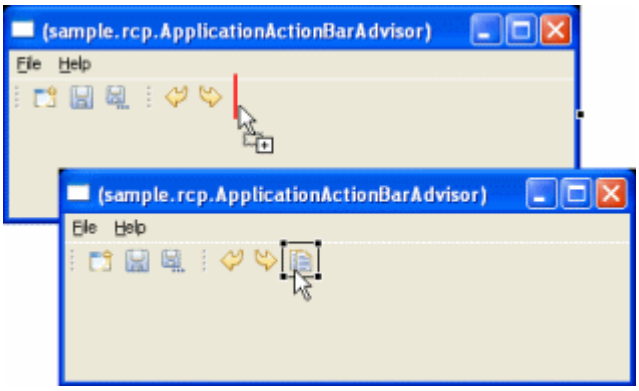


SWT GroupLayout



SWING GroupLayout

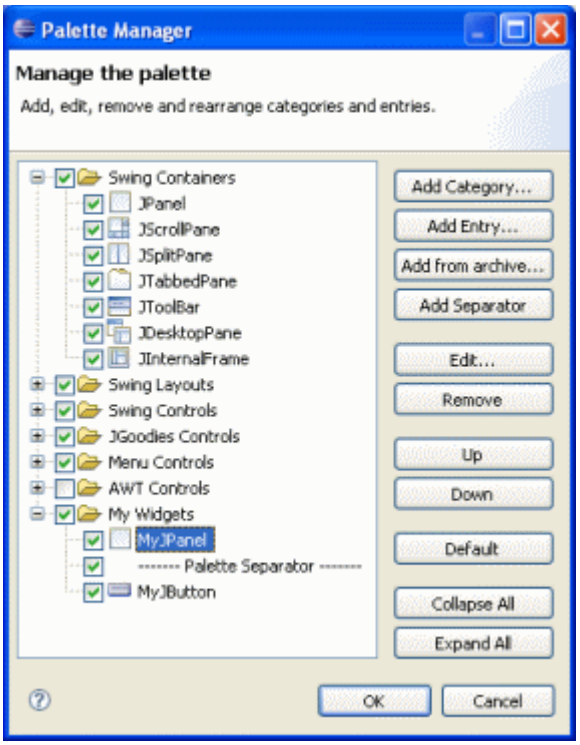
Ejemplo 2: Como crear y editar el menú y las barras de herramientas de aplicaciones RCP.



RCP Menú y Barra de Herramientas

Ejemplo 3: Administración de la Paleta de Componentes.

Paleta de Componentes



Administración de la Paleta

Con esta herramienta, como se puede observar en los ejemplos, el desarrollador puede añadir fácilmente los controles, usando drag-and-drop (arrastrar-y-gota), agrega eventos a sus controles, puede realizar cambios a las propiedades de estos controles, con la utilización de un editor de propiedades, e incluso, es capaz de revertir códigos creados en otras herramientas de diseño para Java, tales como:

- ***JBuilder.***
- ***NetBeans.***
- ***JFormDesigner.***
- ***VisualAge.***
- ***Eclipse VE.*** [EclipseCorner, 2006]

1. 6 Pruebas.

"La prueba es el proceso de un programa con la intención específica de encontrar errores previos a la entrega al usuario final." [Juristo - Moreno –Vegas, 2006]

Las pruebas de software son la comprobación del software, de modo general. Esta comprobación se basa fundamentalmente en cuatro aspectos:

1. Demostración *(proof)*: manual o semiautomática
2. Inspección manual del código
3. Prueba o ensayo *(testing)*: ejecutar y ver resultados
4. Caso de prueba: ensayo individual

El objetivo principal de las pruebas de software es encontrar defectos en el mismo. El éxito de las pruebas radica en encontrar un defecto y el fracaso, en la no detección de defectos existentes. [Collado, 2005]

Para el desarrollo de la capa de Presentación del SIGIA, se realizan pruebas de validación de las interfaces de usuario, en la certificación de cada una de las interfaces de usuario con las que cuentan los módulos Administración y Nomencladores.

"Las pruebas de interfaz de usuario verifican la interacción del usuario con el software. La meta de estas pruebas es asegurar que la interfaz de usuario provee la navegación y el acceso apropiados dentro de la aplicación." [Mañas, 2007]

1. 7 Conclusiones del capítulo.

En el presente capítulo se hizo una descripción de los conceptos asociados a las principales actividades que debe desarrollar un programador de acuerdo con la metodología RUP y se valoraron las herramientas que utiliza el programador en el desarrollo de la capa de Presentación de los módulos Administración y Nomencladores, para concluir con la necesidad del desarrollo en esta capa, un conjunto de interfaces amigables, funcionales y rápidas, teniendo en cuenta la el empleo eficiente de estas herramientas, framework, plataforma, IDE y lenguaje de programación.

2

Capítulo 2: Descripción de la solución propuesta.

2. 1 Introducción al capítulo.

En el presente capítulo se hace una descripción de la solución propuesta de este trabajo, y para ello se expone la estructura que presenta la capa de Presentación en integración con la capa de Negocio. Se realiza una valoración crítica del diseño de clases propuesto por los diseñadores, se analizan y describen cada uno de los artefactos desarrollados por el programador en cada uno de los CU implementados, que conforman los módulos Administración y Nomencladores, así como los algoritmos utilizados y el trabajo con el Proceso Personal del Software (PSP).

2. 2 Capa de Presentación y Estrategias de Integración.

Atendiendo a los estilos arquitectónicos definidos por el arquitecto a utilizar en el desarrollar el SIGIA, los módulos que lo conforman se encuentran estructurados de la siguiente forma:

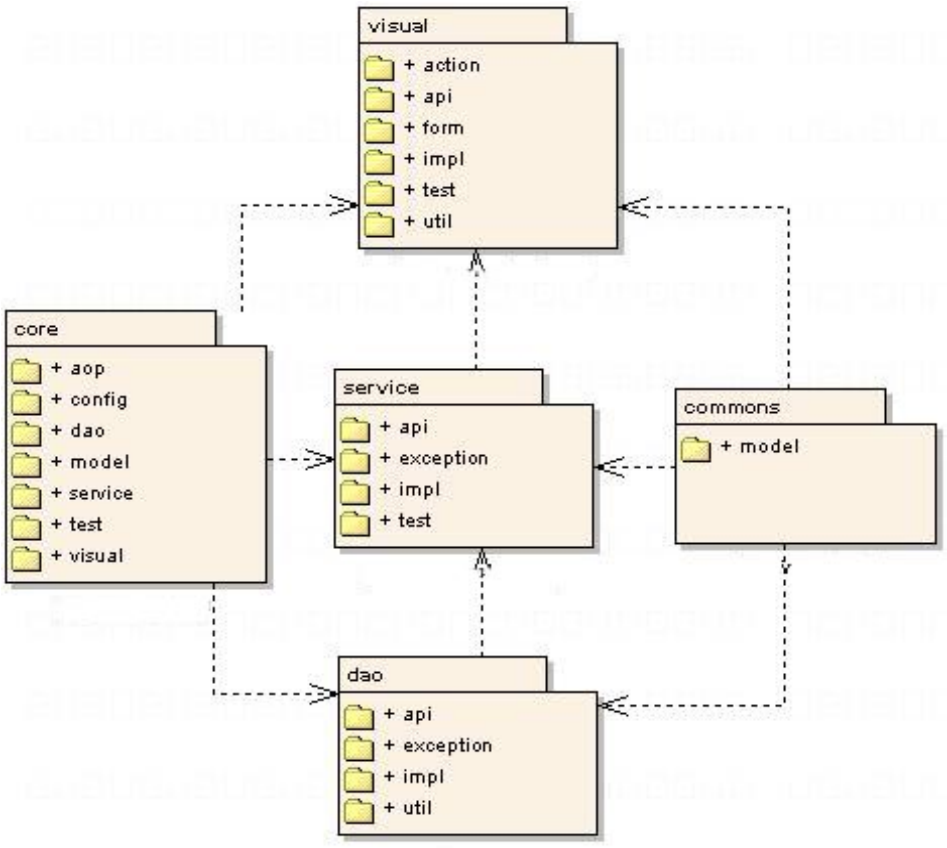


Figura 2. 1: Vista de implementación de los módulos del SIGIA. [Perera, 2007]

La capa de Presentación se encuentra compuesta por dos capas: la capa de Interfaz de Usuario y la capa de Acciones. En la capa de Interfaz de Usuario o paquete ***form***, se encuentran los formularios diseñados atendiendo el Prototipo de Interfaz que elaboran los analistas del sistema para la implementación del mismo, mientras que en la capa de Acciones van a estar todas las clases de los paquetes ***impl, api, action,*** y ***xml*** que contienen las interfaces, implementaciones a esas interfaces, las configuraciones y otras clases referentes a cada uno de los casos de uso que conforman el módulo. Esta capa, al igual que la capa de Negocio y la capa de Acceso a datos, está relacionada con dos paquetes que contienen clases y configuraciones comunes: ***Core*** y ***Commons***.

En el paquete ***Core*** se encuentran un conjunto de clases que implementan métodos de validaciones, conversiones entre distintos tipos de datos, trabajo con cadenas y otros algoritmos que le proporcionen a las interfaces de usuario, mayor rapidez y funcionalidad. Además se encuentran otras clases que configuran los elementos generales de cada capa, así como la interfaz principal del sistema. Anexos V y VI.

En el paquete ***Commons*** se encuentran las clases persistentes, mapeadas de la base de datos, por la capa de Acceso a Dato, con las que interactúa cada una de las capas, en la solución de cada CU.

Figura 2. 2: Estructura de la capa de Presentación y relación con los paquetes Core y Commons. [Perera, 2007]

La integración entre las tres capas y de cada uno de los módulos que conforma el sistema, está regida por un ***XML*** de configuración que integra cada uno de los ***XML*** de cada una de las capas, de la siguiente forma:

```
<beans>

    <import
        resource="classpath:zun/inventario/core/service/applicationContext-AOP-Services.xml" />

    <import
        resource="classpath:zun/inventario/core/visual/applicationContext-visual.xml" />

    <import
        resource="classpath:zun/inventario/core/aop/applicationContext-AOP-Aspects.xml" />

    <import
        resource="classpath:zun/inventario/core/config/applicationContext-configuration.xml"/>

</beans>
```

La capa de Presentación consta de un conjunto de ***XML*** de configuración, para la clase padre del formulario principal *(BaseForm)* y la clase padre de los formularios secundarios *(BaseInternalForm)*, como son:

```
<bean id="mainWindow"

class="zun.inventario.core.visual.main.MainWindow"parent="baseForm" />
 <bean id="baseForm" abstract="true"
       class="zun.inventario.core.visual.util.form.BaseForm">
       <property name="zunAction">
             <ref bean="zunAction" />
       </property>
       <property name="accAction">
             <ref bean="accAction" />
       </property>

 </bean>

 <bean id="baseInternalForm" abstract="true"
       class="zun.inventario.core.visual.util.form.BaseInternalForm">
       <property name="zunAction">
             <ref bean="zunAction" />
       </property>
       <property name="accAction">
             <ref bean="accAction" />
       </property>

 </bean>
```

En la integración de la capa de Presentación y la capa de Negocio, los ***XML*** de configuración principal se muestran de la siguiente forma:

```
<!-- ZUN ACTION -->
<bean id="zunAction"
    class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="interceptorNames">
        <list>
            <idref local="throwsAdvice" />
        </list>
    </property>
    <property name="target">
        <ref bean="zunBaseAction" />
    </property>
</bean>

<!-- ACC ACTION -->
<bean id="accAction"
    class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="interceptorNames">
        <list>
            <idref local="throwsAdvice" />
        </list>
    </property>
    <property name="target">
        <ref bean="accBaseAction" />
    </property>
</bean>
```

```
<bean id="zunBaseAction"
    class="zun.inventario.core.visual.impl.ZunBaseActionImpl">
    <constructor-arg>
        <ref bean="zunService" />
    </constructor-arg>
</bean>

<bean id="accBaseAction"
    class="zun.inventario.core.visual.impl.AccBaseActionImpl">
    <constructor-arg>
        <ref bean="accService" />
    </constructor-arg>
</bean>
```

Donde los ***beans*** (Componente de software que tiene la particularidad de ser reutilizable) que tienen como identificadores *(zunAction) y (accAction)* de las clases padres del formulario principal y los formularios secundarios, referencian a los ***beans*** con los identificadores *(zunBaseAction) y (accBaseAction)* de la clase de implementación: ZunBaseActionImpl y AccBaseActionImpl y estos a su vez, referencian a los beans de servicio *zunService y accService,* respectivamente.

A continuación se muestra un ejemplo de uno de los CU del módulo Nomencladores, donde se evidencia la integración entre estas capas:

```
<!-- PRODUCTO FORMS -->
<bean id="enProductoForm"
    class="zun.inventario.nomencladores.visual.form.producto.EnProductoForm"
    parent="baseInternalForm">
    <property name="productoAction">
        <ref bean="productoNomAction"/>
    </property>
</bean>

<bean id="productoNomAction" class="zun.inventario.nomencladores.visual.impl.ProductoActionImpl">
    <constructor-arg>
        <ref bean="pService"/>
    </constructor-arg>
</bean>
```

2. 3 Diseño de Clases.

Para realizar una correcta implementación en la capa de Presentación, de los CU que conforman los módulos Administración y Nomencladores, para el SIGIA, es necesario contar con el Modelo de Diseño.

Un diseño bien elaborado y sin ambigüedades para cada uno de los CU, que se subordine a un análisis previo como plantea la metodología utilizada, va a posibilitar que el programador tenga una mayor visión de cómo debe realizar la implementación y de las clases que debe crear, independientemente de las clases del diseño, para dar una solución eficiente al problema en cuestión.

La herramienta utilizada por los diseñadores para la elaboración del diseño de los CU fue el ***Enterprise Architect v6.1***. Esta es una herramienta *CASE* para el diseño y la construcción de los sistemas de software. Soporta UML 2.0 y es una herramienta progresiva que cubre todos los aspectos del ciclo de desarrollo.

El diseño de los CU propuesto por los diseñadores, cumple con los requisitos funcionales definidos por los analistas, creando de forma apropiada, un punto de partida para la realización de las actividades de implementación. Se encuentra elaborado de forma específica para un lenguaje, en este caso, para el lenguaje Java. Además brinda mayor solidez y estabilidad a la arquitectura y mayor comprensión de algunos aspectos relacionados con los requisitos no funcionales y restricciones relacionadas el lenguaje de programación, componentes reutilizables y tecnologías de interfaz de usuario, entre otros.

Este diseño posibilita que las actividades de la implementación se puedan realizar por diferentes equipos de desarrollo, es decir, permite descomponer el trabajo en partes manejables. También identifica y captura las interfaces entre las capas con anterioridad a la implementación. Este es un aspecto importante para el desarrollo del SIGIA porque se utilizan las interfaces como elementos de sincronización entre los diferentes equipos de desarrollo.

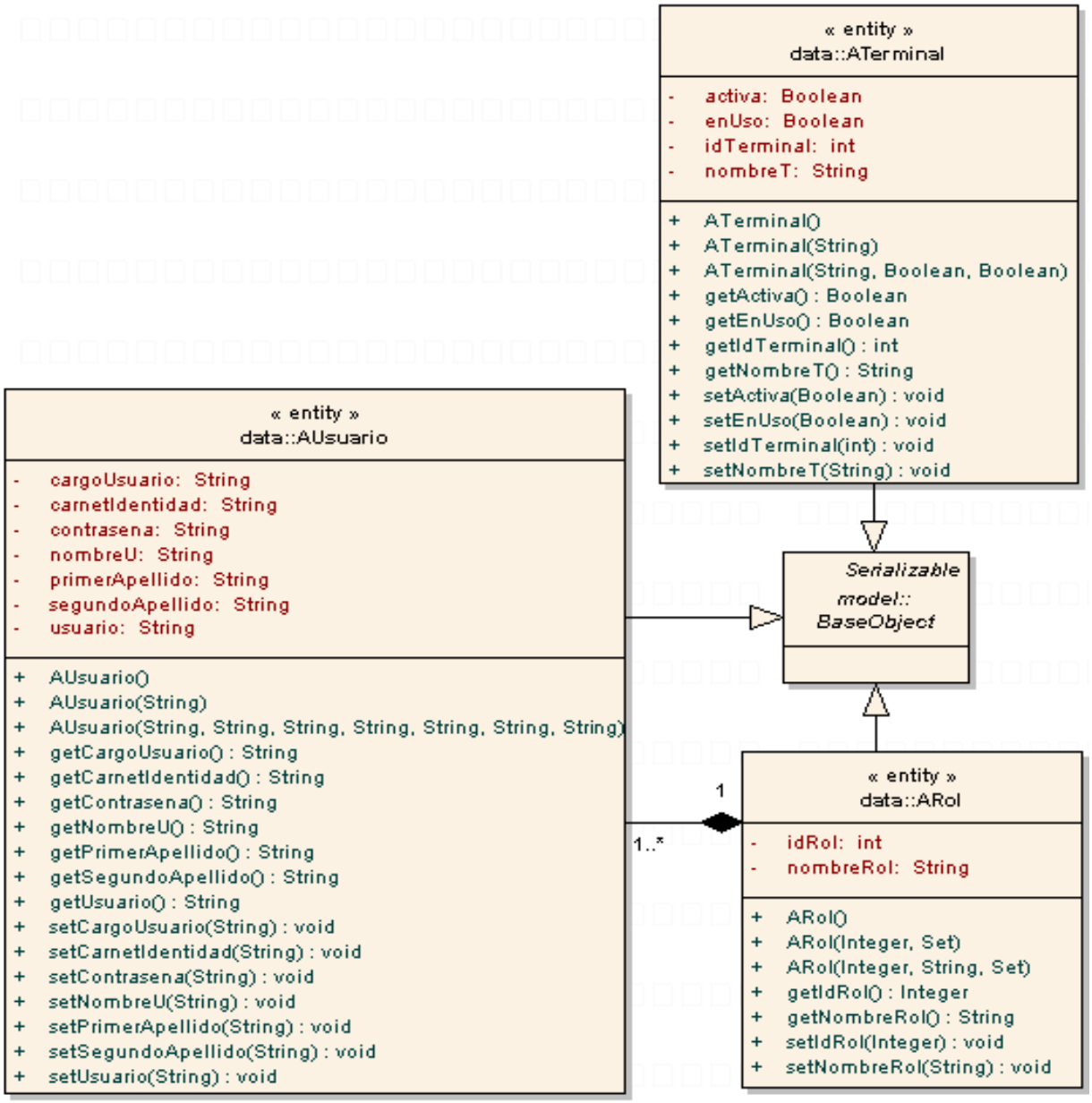
A continuación se muestra el diseño de las clases persistentes, para cada caso de uso, que conforman los módulos de Administración y Nomencladores, seleccionadas del Modelo de Diseño del SIGIA.

2. 3. 1 Diseño de las clases persistentes.

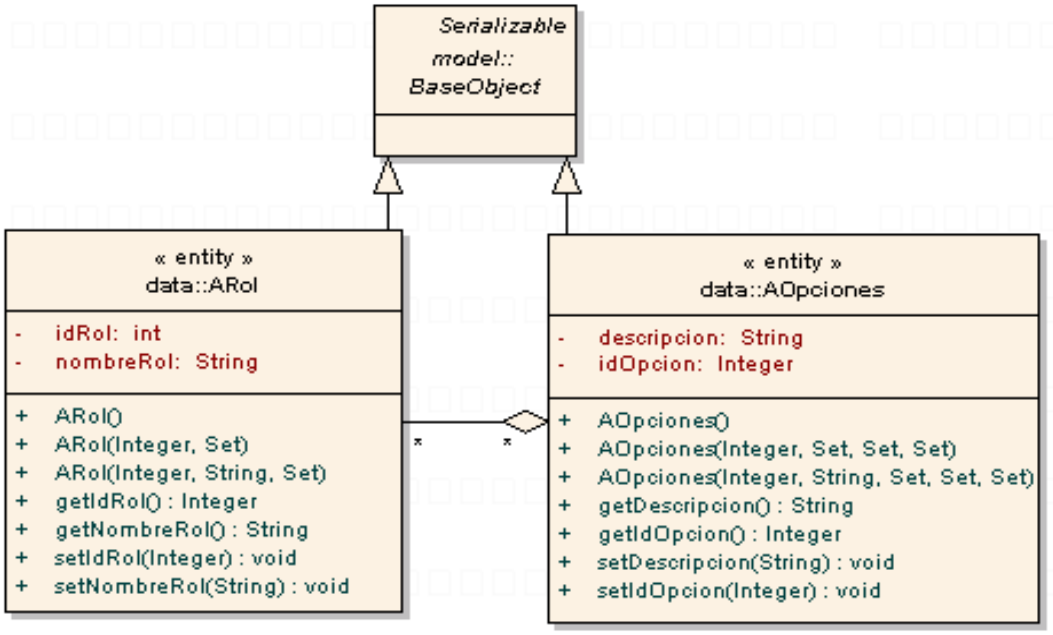
2. 3. 1. 1 Módulo Administración.

El módulo Administración está conformado por cuatro casos de uso: *CU_Autenticar*, *CU_Gestionar_Roles*, *CU_Gestionar_ Estaciones_de_Trabajo ó Terminales* y *CU_Gestionar_Usuarios*.

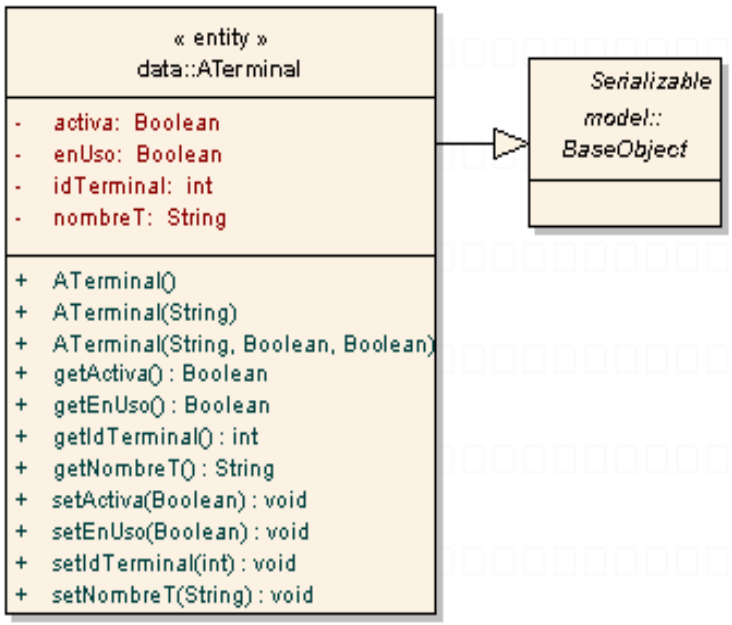
CU_Autenticar:



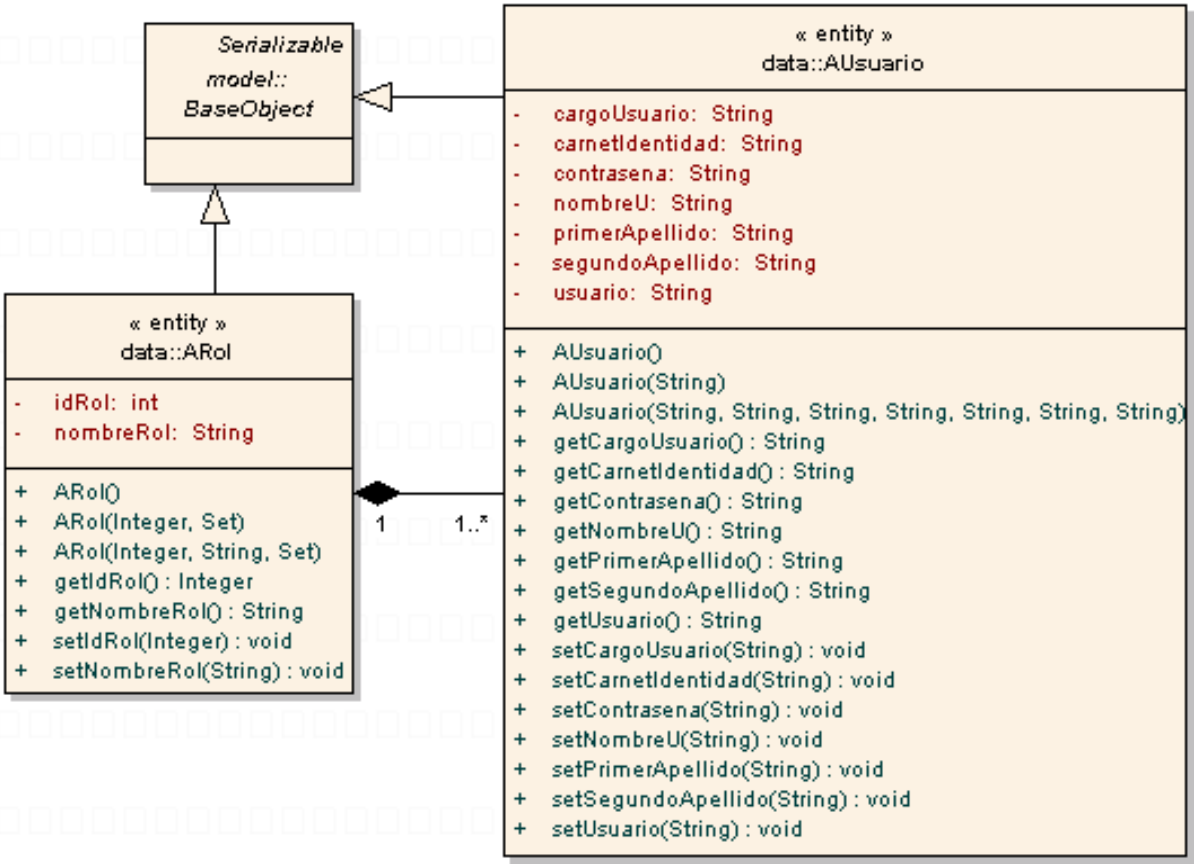
CU_Gestionar_Roles:



CU_Gestionar_Estaciones_de_Trabajo:



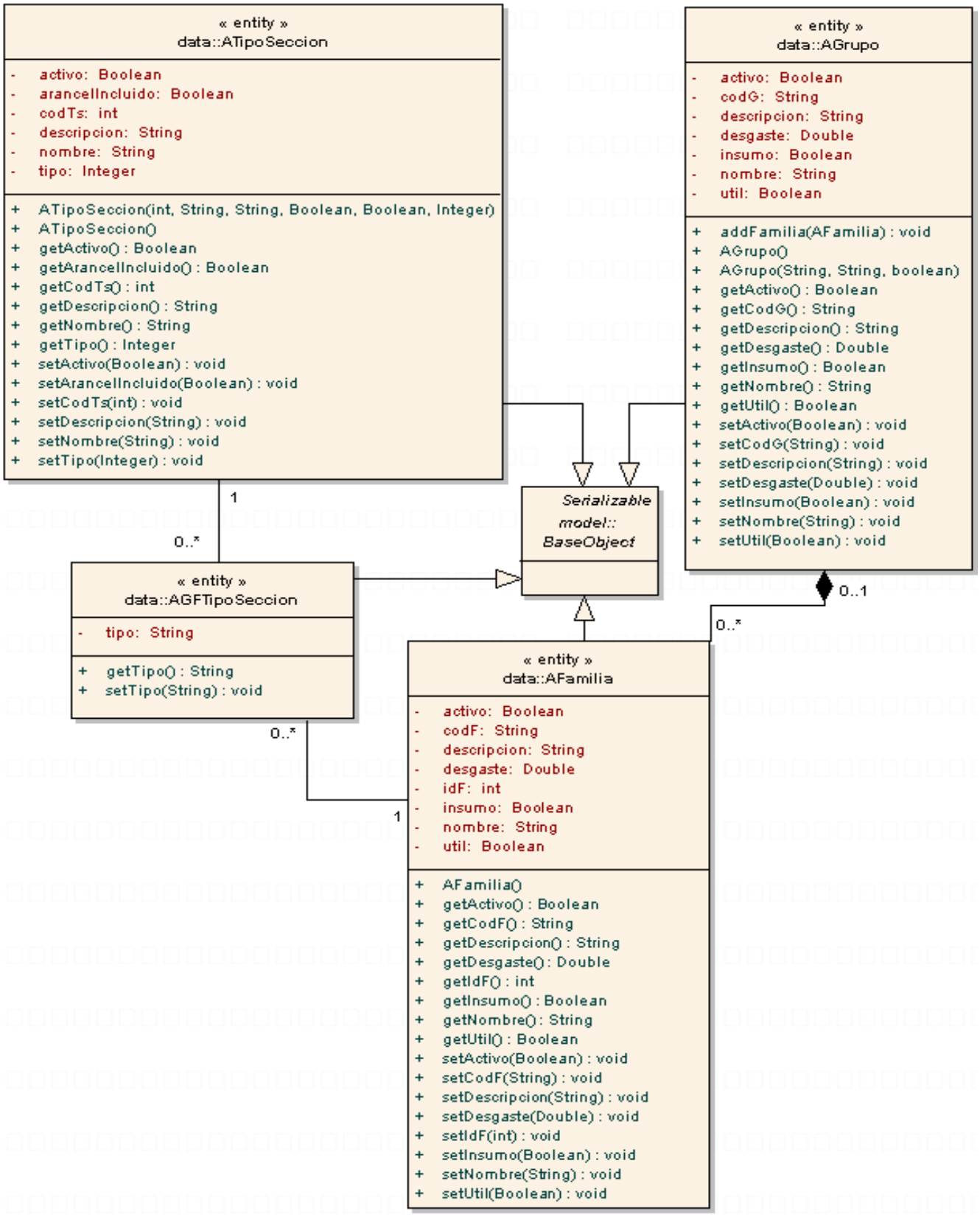
CU_Gestionar_Usuarios:



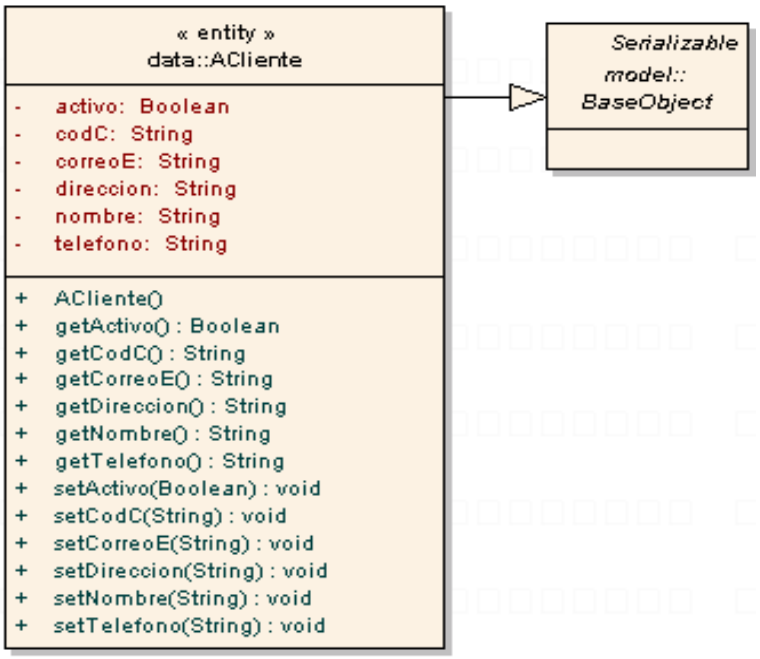
2. 3. 1. 2 Módulo Nomencladores.

El módulo Nomencladores está conformado por nueve casos de uso: CU_Clasificación_de_Secciones, CU_Clientes, CU_Especialidad_de_Proveedor, CU_Grupo/Familia, CU_Producto, CU_Proveedor, CU_Sección_de_Inventario, CU_Tipo_de_Producto y CU_Unidad_de_Medida.

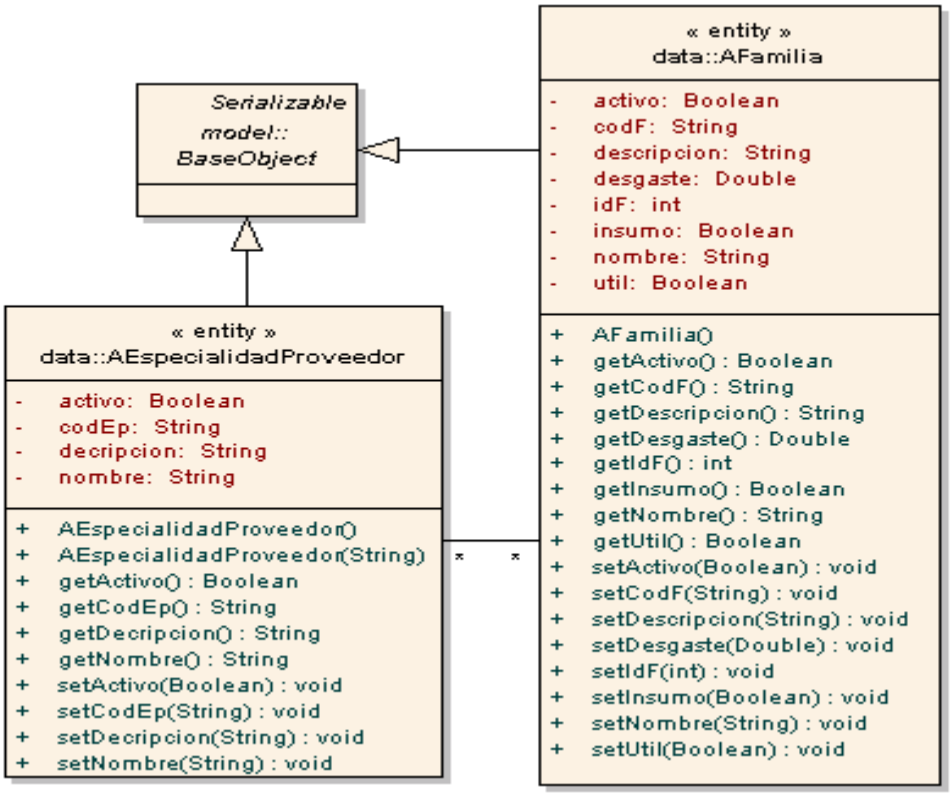
CU_Clasificación_de_Secciones:



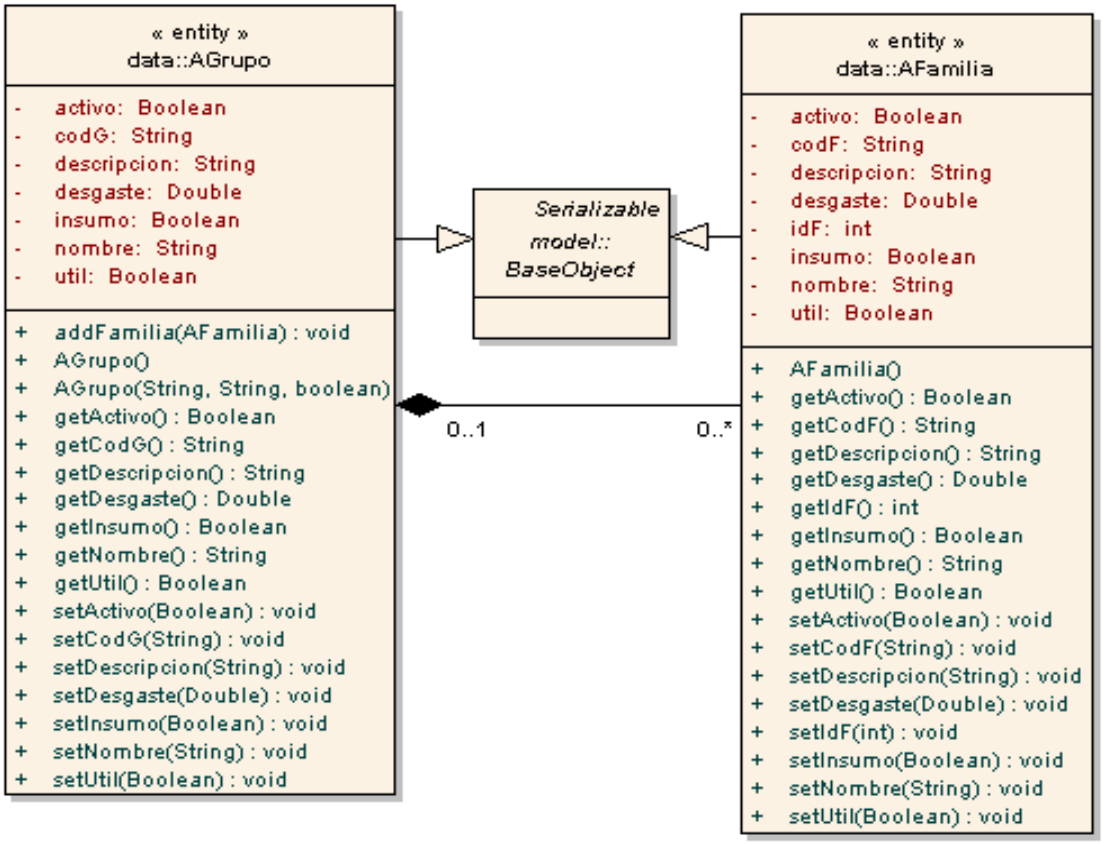
CU_Clientes:



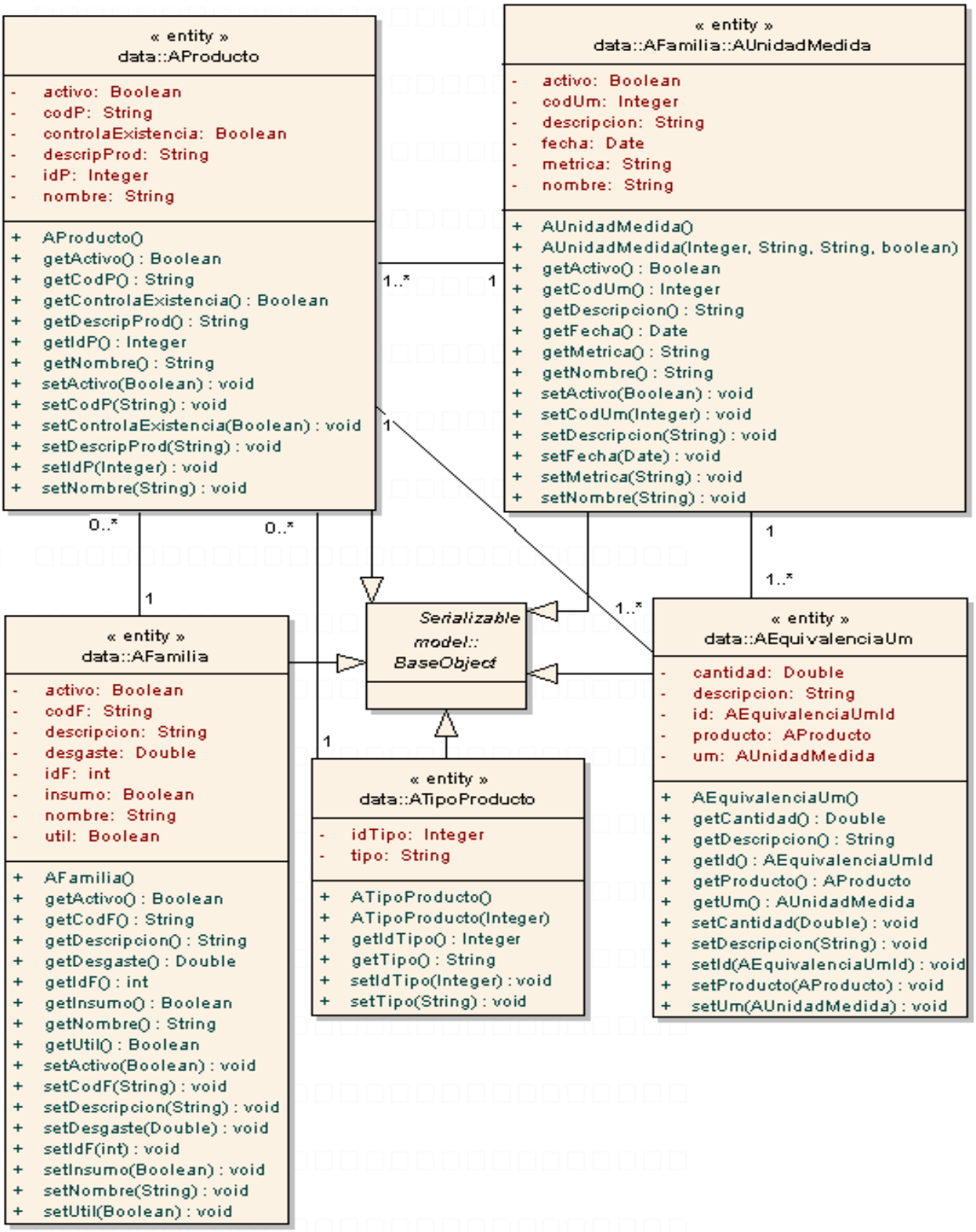
CU_Especialidad_de_Proveedor:



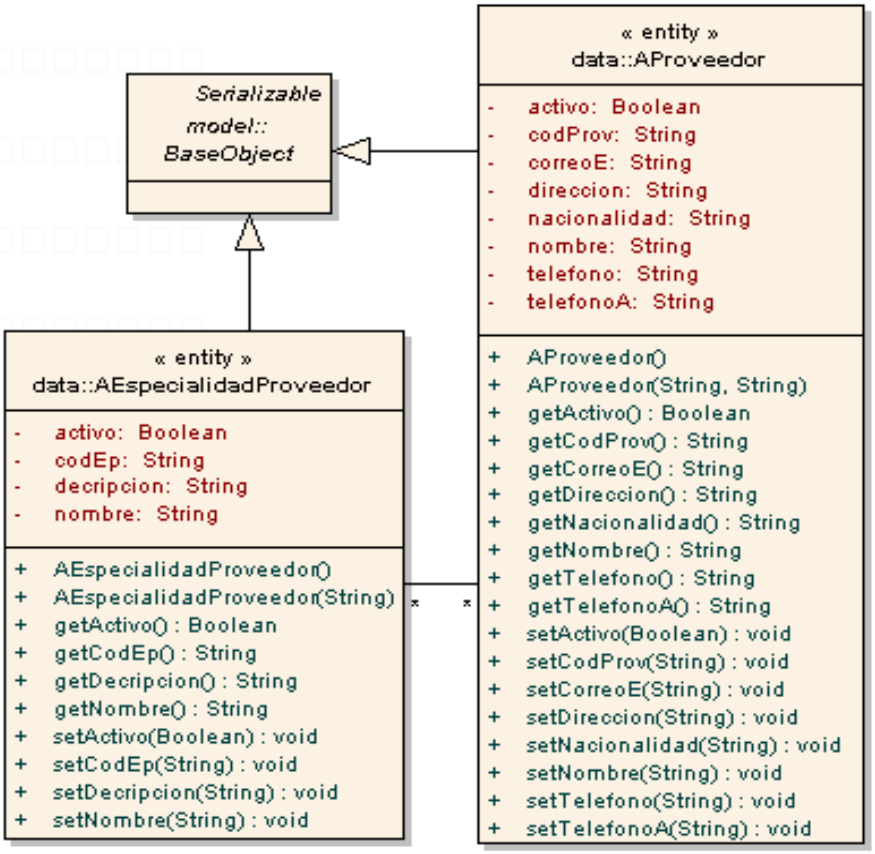
CU_Grupo/Familia:



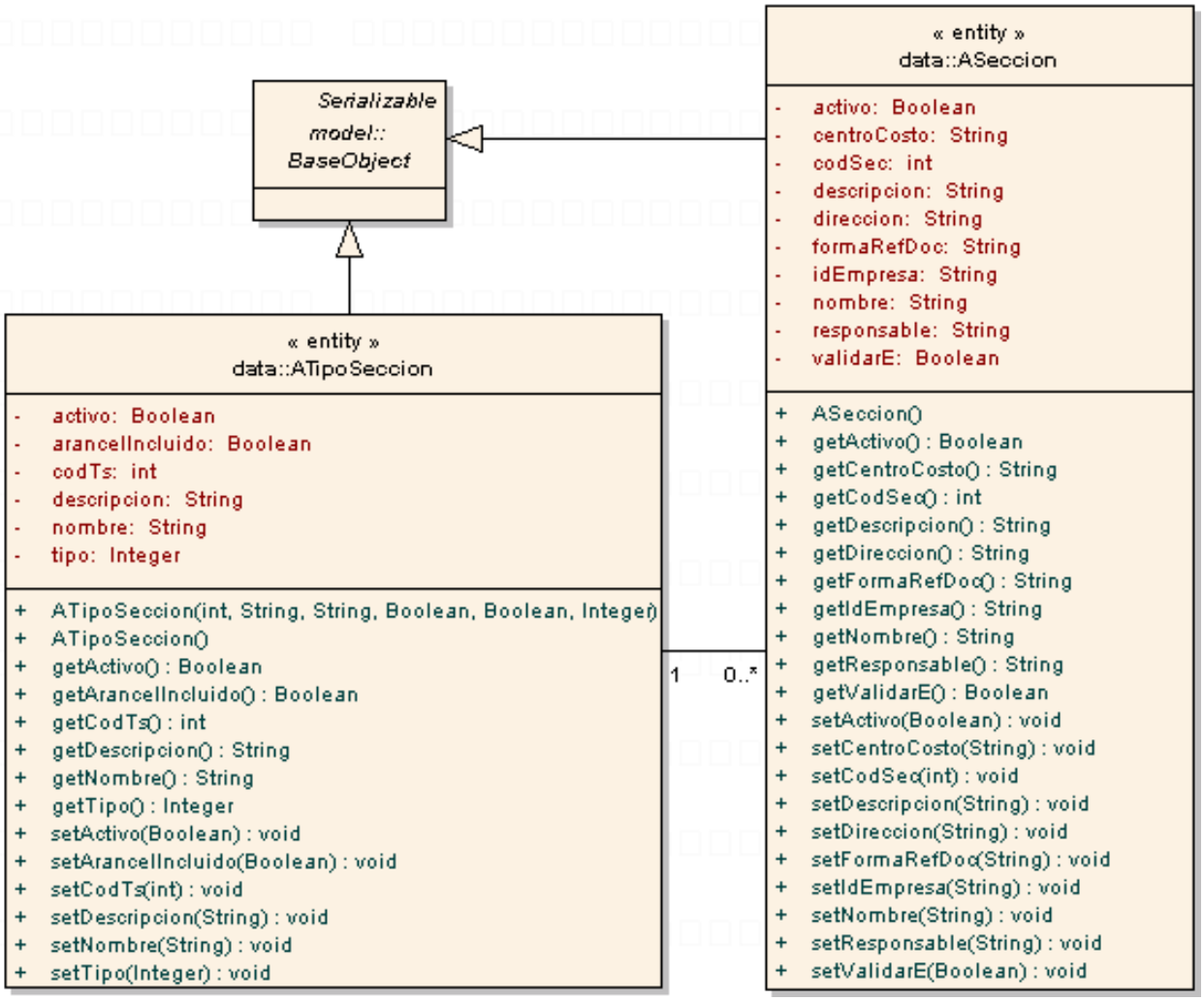
CU_Producto:


« entity »
data::AProducto
- activo: Boolean
- codP: String
- controlaExistencia: Boolean
- descripProd: String
- idP: Integer
- nombre: String
+ AProducto()
+ getActivo() : Boolean
+ getCodP() : String
+ getControlaExistencia() : Boolean
+ getDescripProd() : String
+ getIdP() : Integer
+ getNombre() : String
+ setActivo(Boolean) : void
+ setCodP(String) : void
+ setControlaExistencia(Boolean) : void
+ setDescripProd(String) : void
+ setIdP(Integer) : void
+ setNombre(String) : void
« entity »
data::AFamilia::AUnidadMedida
- activo: Boolean
- codUm: Integer
- descripcion: String
- fecha: Date
- metrica: String
- nombre: String
+ AUnidadMedida()
+ AUnidadMedida(Integer, String, String, boolean)
+ getActivo() : Boolean
+ getCodUm() : Integer
+ getDescripcion() : String
+ getFecha() : Date
+ getMetrica() : String
+ getNombre() : String
+ setActivo(Boolean) : void
+ setCodUm(Integer) : void
+ setDescripcion(String) : void
+ setFecha(Date) : void
+ setMetrica(String) : void
+ setNombre(String) : void
1..*
1
1
0..*
0..*
1
1
1
1..*
1..*
« entity »
data::AFamilia
- activo: Boolean
- codF: String
- descripcion: String
- desgaste: Double
- idF: int
- insumo: Boolean
- nombre: String
- util: Boolean
+ AFamilia()
+ getActivo() : Boolean
+ getCodF() : String
+ getDescripcion() : String
+ getDesgaste() : Double
+ getIdF() : int
+ getInsumo() : Boolean
+ getNombre() : String
+ getUtil() : Boolean
+ setActivo(Boolean) : void
+ setCodF(String) : void
+ setDescripcion(String) : void
+ setDesgaste(Double) : void
+ setIdF(int) : void
+ setInsumo(Boolean) : void
+ setNombre(String) : void
+ setUtil(Boolean) : void
Serializable
model::
BaseObject
« entity »
data::ATipoProducto
- idTipo: Integer
- tipo: String
+ ATipoProducto()
+ ATipoProducto(Integer)
+ getIdTipo() : Integer
+ getTipo() : String
+ setIdTipo(Integer) : void
+ setTipo(String) : void
« entity »
data::AEquivalenciaUm
- cantidad: Double
- descripcion: String
- id: AEquivalenciaUmId
- producto: AProducto
- um: AUnidadMedida
+ AEquivalenciaUm()
+ getCantidad() : Double
+ getDescripcion() : String
+ getId() : AEquivalenciaUmId
+ getProducto() : AProducto
+ getUm() : AUnidadMedida
+ setCantidad(Double) : void
+ setDescripcion(String) : void
+ setId(AEquivalenciaUmId) : void
+ setProducto(AProducto) : void
+ setUm(AUnidadMedida) : void

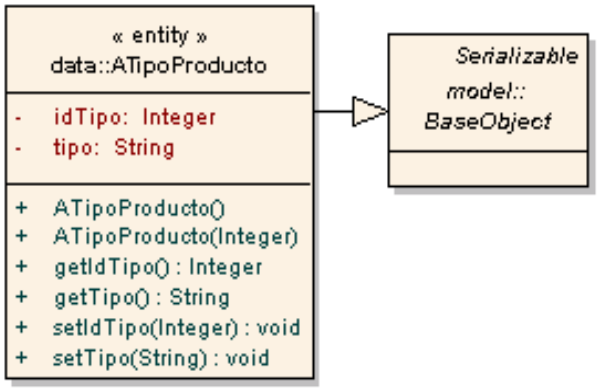
CU_Proveedor:



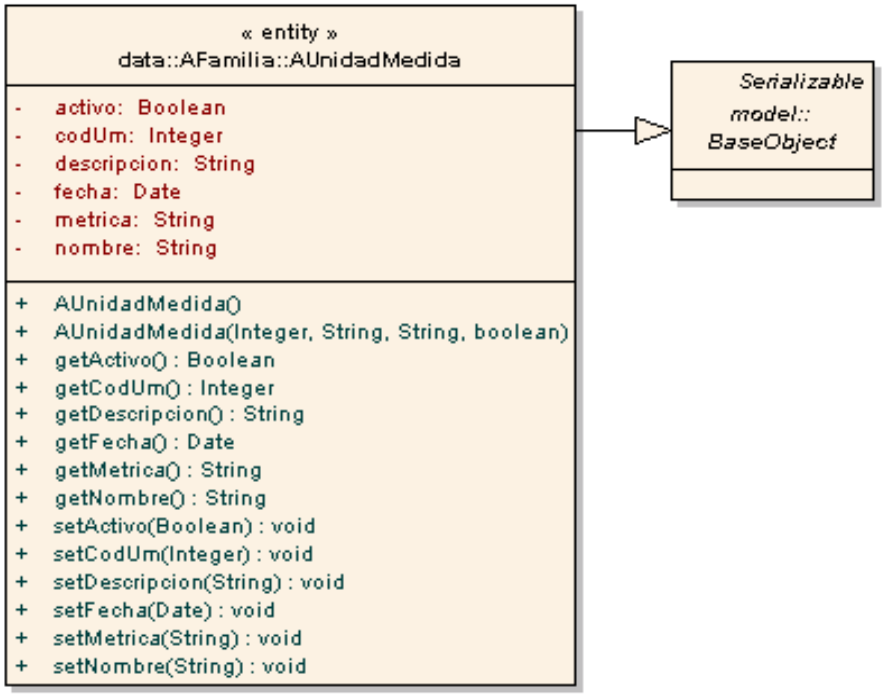
CU_Sección_de_Inventario:



CU_Tipo_de_Producto:



CU_Unidad_de_Medida:



2. 4 Implementación de las clases del diseño.

Para la implementación de la capa de Presentación, se cumple con el patrón *Modelo Vista Controlador*, definido por el arquitecto; independientemente de los patrones que proveen el framework. Se cumplen además, con un grupo de patrones definidos por el diseñador, que rigen la implementación, tales como: *Alta Cohesión, Bajo Acoplamiento, Experto y Estado;* brindado la solución adecuada a problemas específicos. También se tiene en cuenta el Prototipo de Interfaz de Usuario propuesto por los analistas, así como la identificación de las clases persistentes, para cada CU de los módulos de Administración y Nomencladores y el estándar de codificación elaborado por el arquitecto.

Para esta capa, se desarrollaron un conjunto de clases comunes que brindan una mayor reutilización del código. Clases como la interfaz *BaseAction* y su implementación *ZunBaseActionImpl*, que intervienen directamente en la integración con la capa de Negocio, engloban un conjunto de métodos que facilitarán el trabajo con las acciones de las interfaces de usuario. Además está la clase *BaseBean*, de la cual heredan todas las clases controladoras que implementan las interfaces. Esta clase contiene el objeto *serviceLocator*, que va a posibilitar que el servicio instanciado se cree una sola vez,

debido al patrón de diseño que tiene implementado: *"Singleton".* Entre las clases comunes se encuentran además la clase padre del formulario principal *(BaseForm)* y la clase padre de los formularios secundarios *(BaseInternalForm).*Anexo VII

A continuación se muestran las descripciones de las clases que intervienen en la implementación de la capa de Presentación, para cada caso de uso que conforman los módulos de Administración y Nomencladores.

2. 4. 1 Descripción de las clases de la capa de Presentación.

2. 4. 1. 1 Módulo Administración

En la realización del *CU_Autenticar* es preciso destacar que solamente intervienen la interfaz *AutenticarAction* y la implementación a dicha interfaz *(AutenticarActionImpl)*, en la integración con la capa de Negocio, mientras que en los tres casos de uso restantes, encontramos la interfaz e implementación correspondiente a cada uno de ellos y la interfaz *BaseAction* con su implementación *(ZunBaseActionImpl),* para lograr esta integración. Anexo VIII

CU_Autenticar:

Nombre: EnAutenticarForm	
Tipo de clase: interfaz	
Atributo	**Tipo**
autenticarAction	private AutenticarAction
autenticarForm	private JFrame
btnAceptar	private JButton
btnCancelar	private JButton
lbContrasena	private JLabel
lbUsuario	private JLabel
pnAutenticar	private JPanel
tfContrasena	private JPasswordField
tfUsuario	private JTextField
Para cada responsabilidad:	
Nombre:	**EnAutenticarForm***():*
Descripción:	Constructor de la clase
Nombre:	**getAutenticarAction***():AutenticarAction*
Descripción:	Devuelve el valor del atributo autenticarAction en dicha clase.
Nombre:	**getAutenticarForm***():JFrame*
Descripción:	Devuelve el valor del atributo autenticarForm en dicha clase.
Nombre:	**initialize***():void*
Descripción:	Método para inicializar los componentes.
Nombre:	**setAutenticarAction***(AutenticarAction autenticarAction):void*
Descripción:	Modifica el valor del atributo autenticarAction en dicha clase.

Nombre:	**setAutenticarForm***(JFrame autenticarForm):void*
Descripción:	Modifica el valor del atributo autenticarForm en dicha clase.

Nombre: AutenticarAction	
Tipo de clase: interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**autenticarUsuario***(String user, String pass):String*
Descripción:	Declaración
Nombre:	**getService***():AutenticarService*
Descripción:	Declaración
Nombre:	**isTerminalActive***():String*
Descripción:	Declaración
Nombre:	**setService***(AutenticarService service):void*
Descripción:	Declaración

Nombre: AutenticarActionImpl	
Tipo de clase: controladora	
Atributo	**Tipo**
service	private AutenticarService
Para cada responsabilidad:	
Nombre:	**AutenticarActionImpl***(AutenticarService service):*
Descripción:	Constructor de la Clase
Nombre:	**autenticarUsuario***(String user, String pass):String*
Descripción:	Método para la autenticación de un usuario
Nombre:	**getService***():AutenticarService*
Descripción:	Devuelve el valor del atributo service en dicha clase.
Nombre:	**isTerminalActive***():String*
Descripción:	Chequea si una Terminal o Estación de trabajo está activa
Nombre:	**setService***(AutenticarService service):void*
Descripción:	Modifica el valor del atributo service en dicha clase.

Nombre: MainWindow	
Tipo de clase: interfaz	
Atributo	**Tipo**
enClasifSeccionForm	private EnClasificacionSeccionForm
enClientesForm	private EnClientesForm
enEnlaceContableForm	private EnEnlaceContableForm
enEntradaCompraForm	private EnEntradaCompraForm
enEntradaMovimientoForm	private EnEntradaMovimientoForm
enEspecialidadForm	private EnEspecialidadForm
enGrupoForm	private EnGrupoForm
enMovEntrada	private EnMovimientoEntrada
enProductoForm	private EnProductoForm
enProveedorForm	private EnProveedorForm
enRegistrarOperacionForm	private EnRegistrarOperacionForm
enRolesForm	private EnRolesForm
enSalidaMovimientoForm	private EnSalidaMovimientoForm
enSeccionForm	private EnSeccionForm
enSolicitudMaterialesForm	private EnSolicitudMaterialesForm
enTerminalForm	private EnTerminalForm

enTipoProductoForm	private EnTipoProductoForm
enUnidadMedidaForm	private EnUnidadMedidaForm
enUsuarioForm	private EnUserForm
idop	private ArrayList
listOpcion	ArrayList
mainWindow	public JFrame
Para cada responsabilidad:	
Nombre:	**getListOpcion***():List*
Descripción:	Devuelve el valor del atributo listOpcion en dicha clase.
Nombre:	**getMainWindow***():JFrame*
Descripción:	Devuelve el valor del atributo mainWindow en dicha clase.
Nombre:	**initialize***():void*
Descripción:	Inicialización de los componentes de la clase
Nombre:	**MainWindow***()*
Descripción:	Constructor de la clase
Nombre:	**setListOpcion***(List listOpcion):void*
Descripción:	Modifica el valor del atributo listOpcion en dicha clase.
Nombre:	**setMainWindow***(JFrame mainWindow):void*
Descripción:	Modifica el valor del atributo mainWindow en dicha clase.
Nombre:	**Listaop***(AUsuario user):void*
Descripción:	Devuelve la lista de opciones de un usuariopasando el usuario como parámetro.

CU_Gestionar_Roles:

Nombre: EnRolesForm	
Tipo de clase: interfaz	
Atributo	**Tipo**
btnCancelar	private JButton
btnCancelaradmin	private JButton
btnCancelarnomencl	private JButton
btnCrear	private JButton
btnEliminarRol	private JButton
btnIr	private JButton
cbCampo	private JComboBox
cbModulos	private JComboBox
chbAActivardesactt	private JCheckBox
chbACreareditarr	private JCheckBox
chbACreareditaru	private JCheckBox
chbAEliminarr	private JCheckBox
chbAEliminart	private JCheckBox
chbAEliminaru	private JCheckBox
chbNCreareditarcs	private JCheckBox
chbNCrearediteprov	private JCheckBox
chbNCreareditgfdep	private JCheckBox
chbNCreareditp	private JCheckBox
chbNCreareditprov	private JCheckBox
chbNCreareditsecc	private JCheckBox
chbNCreareditum	private JCheckBox
chbNEliminarcs	private JCheckBox
chbNEliminargfdep	private JCheckBox

chbNEliminarp	private JCheckBox
chbNEliminarprov	private JCheckBox
chbNEliminarsecc	private JCheckBox
chbNEliminarum	private JCheckBox
codRol	Integer
ifAdministracion	private JInternalFrame
ifNomencladores	private JInternalFrame
lbBuscarPor	private JLabel
lbModulos	private JLabel
lbNombreRol	private JLabel
opadmin	ArrayList
opnomencl	ArrayList
pnAddRol	private JPanel
pnAdministracion	private JPanel
pnBuscarRol	private JPanel
pnNomenclar	private JPanel
pnRoles	private JPanel
rolesAction	private RolesAction
spRolesRegistrados	private JScrollPane
tfNombreRol	private JTextField
tfValor	private JTextField
tRoles	private JTable
Para cada responsabilidad:	
Nombre:	**buscar***():void*
Descripción:	Método para buscar los roles de la BD
Nombre:	**createJTable***(List list):void*
Descripción:	Método para crear la tabla con los roles de la BD
Nombre:	**EnRolesForm***():*
Descripción:	Constructor de la clase
Nombre:	**getRolesAction***():RolesAction*
Descripción:	Devuelve el valor del atributo rolesAction en dicha clase.
Nombre:	**limpiar***():void*
Descripción:	Método para limpiar los componentes del formulario
Nombre:	**limpiarif***():void*
Descripción:	Método para limpiar los componentes del internalFrame de las opciones.
Nombre:	**nuevoRoles***():void*
Descripción:	Este método se utiliza para mostrar el formulario.
Nombre:	**opciones***():void*
Descripción:	Este método se utiliza en la captura de las opciones de un rol.
Nombre:	**rolop***(ARol rol):void*
Descripción:	Este método se utiliza en la creación de las opciones de un rol.
Nombre:	**setRolesAction***(RolesAction rolesAction):void*
Descripción:	Modifica el valor del atributo rolesAction en dicha clase.

Nombre: RolesAction	
Tipo de clase: interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**deleteRolUsuario***(ARol obj):boolean*
Descripción:	Declaración
Nombre:	**getService***():GestionarRolesService*

Descripción:	Declaración
Nombre:	**setService***(GestionarRolesService service):void*
Descripción:	Declaración

Nombre: RolesActionImpl	
Tipo de clase: controladora	
Atributo	**Tipo**
service	private GestionarRolesService
Para cada responsabilidad:	
Nombre:	**deleteRolUsuario***(ARol obj):boolean*
Descripción:	Método para eliminar un rol, pasando el rol como parámetro.
Nombre:	**getService***():GestionarRolesService*
Descripción:	Devuelve el valor del atributo service en dicha clase.
Nombre:	**RolesActionImpl***(GestionarRolesService service):*
Descripción:	Constructor de la clase
Nombre:	**setService***(GestionarRolesService service):void*
Descripción:	Modifica el valor del atributo service en dicha clase.

CU_Gestionar_Estaciones_de_Trabajo:

Nombre: EnTerminalForm	
Tipo de clase: interfaz	
Atributo	**Tipo**
btnCancelar	private JButton
btnCrear	private JButton
btnEliminar	private JButton
btnIr	private JButton
cbBuscar	private JComboBox
chbTerminal	private JCheckBox
codTerminal	private int
lbActivar	private JLabel
lbBuscarPor	private JLabel
lbNombre	private JLabel
pnListar	private JPanel
pnTerminalPanel	private JPanel
spListaterminal	private JScrollPane
terminalAction	private TerminalAction
tfNombre	private JTextField
tfParametro	private JTextField
tListaterminal	private JTable
Para cada responsabilidad:	
Nombre:	**buscar***():void*
Descripción:	Método para buscar las estaciónes de trabajo de la BD
Nombre:	**createJTable***(List list):void*
Descripción:	Método para crear la tabla con las estaciónes de trabajo de la BD
Nombre:	**EnTerminalForm***():*
Descripción:	Constructor de la clase
Nombre:	**getTerminalAction***():TerminalAction*
Descripción:	Devuelve el valor del atributo terminalAction en dicha clase.
Nombre:	**limpiar***():void*

Descripción:	Método para limpiar los componentes del formulario
Nombre:	**nuevaTerminal***():void*
Descripción:	Este método se utiliza para mostrar el formulario.
Nombre:	**setTerminalAction***(TerminalAction terminalAction):void*
Descripción:	Modifica el valor del atributo terminalAction en dicha clase.

Nombre: TerminalAction	
Tipo de clase: interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**deleteTerminal***(ATerminal terminal):boolean*
Descripción:	Declaración
Nombre:	**getService***():GestionarEstacionesService*
Descripción:	Declaración
Nombre:	**setService***(GestionarEstacionesService service):void*
Descripción:	Declaración

Nombre: TerminalActionImpl	
Tipo de clase: controladora	
Atributo	**Tipo**
service	private GestionarEstacionesService
Para cada responsabilidad:	
Nombre:	**deleteTerminal***(ATerminal terminal):boolean*
Descripción:	Método para eliminar una estación de trabajo, pasando el rol como parámetro.
Nombre:	**getService***():GestionarEstacionesService*
Descripción:	Devuelve el valor del atributo service en dicha clase.
Nombre:	**setService***(GestionarEstacionesService service):void*
Descripción:	Modifica el valor del atributo service en dicha clase.
Nombre:	**TerminalActionImpl***(GestionarEstacionesService service):*
Descripción:	Constructor de la clase

CU_Gestionar_Usuarios:

Nombre: EnUserForm	
Tipo de clase: interfaz	
Atributo	**Tipo**
autenticarAction	private AutenticarAction
btnCancelar	private JButton
btnCancelarPassw	private JButton
btnCrear	private JButton
btnEliminar	private JButton
btnIr	private JButton
btnModificarPassw	private JButton
btnPassw	private JButton
cbCampo	private JComboBox
cbRoles	private JComboBox
ifPassw	private JInternalFrame
lbApellido1	private JLabel
lbApellido2	private JLabel

lbBuscarPor	private JLabel
lbCargo	private JLabel
lbCI	private JLabel
lbNombre	private JLabel
lbPassword	private JLabel
lbPasswordAnterior	private JLabel
lbPasswordConf	private JLabel
lbPasswordNuevo	private JLabel
lbRol	private JLabel
lbUsuario	private JLabel
lbUsuario_1	private JLabel
lUsuarios	private JList
pfPassw	private JPasswordField
pfPasswant	private JPasswordField
pfPasswconf	private JPasswordField
pfPasswnew	private JPasswordField
pnBuscarUser	private JPanel
pnUserPanel	private JPanel
spUsersRegistrados	private JScrollPane
tfApellido1	private JTextField
tfApellido2	private JTextField
tfCargo	private JTextField
ftfCI	private JFormattedTextField
tfNombre	private JTextField
tfUsuario	private JTextField
tfUsuario_1	private JTextField
tfValor	private JTextField
tUsers	private JTable
user	private String
userAction	private UserAction
Para cada responsabilidad:	
Nombre:	**buscar***():void*
Descripción:	Método para buscar los usuarios de la BD
Nombre:	**createJTable***(List list):void*
Descripción:	Método para crear la tabla con los usuarios de la BD
Nombre:	**EnUserForm***():*
Descripción:	Constructor de la clase
Nombre:	**getAutenticarAction***():AutenticarAction*
Descripción:	Devuelve el valor del atributo autenticarAction en dicha clase.
Nombre:	**getUserAction***():UserAction*
Descripción:	Devuelve el valor del atributo userAction en dicha clase.
Nombre:	**inicializar***():void*
Descripción:	Método para llenar los comboboxs de elementos de la BD
Nombre:	**limpiar***():void*
Descripción:	Método para limpiar los componentes del formulario
Nombre:	**nuevoUsuario***():void*
Descripción:	Este método se utiliza para mostrar el formulario.
Nombre:	**setAutenticarAction***(AutenticarAction autenticarAction):void*
Descripción:	Modifica el valor del atributo autenticarAction en dicha clase.
Nombre:	**setUserAction***(UserAction userAction):void*
Descripción:	Modifica el valor del atributo userAction en dicha clase.

Nombre: UserAction	
Tipo de clase: interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**crearU***(AUsuario usuario):void*
Descripción:	Declaración
Nombre:	**deleteU***(AUsuario usuario):void*
Descripción:	Declaración
Nombre:	**getService***():GestionarUsuarioService*
Descripción:	Declaración
Nombre:	**setService***(GestionarUsuarioService service):void*
Descripción:	Declaración
Nombre:	**updateU***(AUsuario usuario):void*
Descripción:	Declaración

Nombre: UserActionImpl	
Tipo de clase: controladora	
Atributo	**Tipo**
service	private GestionarUsuarioService
Para cada responsabilidad:	
Nombre:	**crearU***(AUsuario usuario):void*
Descripción:	Método para crear un usuario, pasando el usuario como parámetro.
Nombre:	**deleteU***(AUsuario usuario):void*
Descripción:	Método para eliminar un usuario, pasando el usuario como parámetro.
Nombre:	**getService***():GestionarUsuarioService*
Descripción:	Devuelve el valor del atributo service en dicha clase.
Nombre:	**setService***(GestionarUsuarioService service):void*
Descripción:	Modifica el valor del atributo service en dicha clase.
Nombre:	**updateU***(AUsuario usuario):void*
Descripción:	Método para modificar un usuario, pasando el usuario como parámetro.
Nombre:	**UserActionImpl***(GestionarUsuarioService service):*
Descripción:	Constructor de la clase

2. 4. 1. 2 Módulo Nomencladores

En la realización del *CU_Clientes* es preciso destacar que solamente intervienen la interfaz *BaseAction* y la implementación a dicha interfaz *(ZunBaseActionImpl)*, en la integración con la capa de Negocio, mientras que en los casos de uso restantes, encontramos la interfaz e implementación correspondiente a cada uno de ellos y la interfaz *BaseAction* con su implementación *(ZunBaseActionImpl),* para lograr esta integración. Anexo IX

CU_ Clasificación_de_Secciones:

Nombre: EnClasificacionSeccionForm
Tipo de clase: interfaz

Atributo	**Tipo**
btnCancelar	private JButton
btnCrear	private JButton
btnEliminar	private JButton
btnIr	private JButton
btnModificar	private JButton
btnVisual	private JButton
cbCampo	private JComboBox
cbGrupos	private JComboBox
cbGrupos_1	private JComboBox
cbTipoSeccion	private JComboBox
chbActivo	private JCheckBox
chbArancelIncluido	private JCheckBox
clasifsecAction	private ClasificacionSeccionesAction
codTS	Integer
enClasificacionSeccionPanel	private JPanel
familiasE	Object[]
familiasER	Object[]
familiasS	Object[]
familiasSR	Object[]
ifMostrar	private JInternalFrame
lbBuscarPor	private JLabel
lbDescripcion	private JLabel
lbFamilias	private JLabel
lbFamilias_1	private JLabel
lbGrupos	private JLabel
lbGrupos_1	private JLabel
lbNombre	private JLabel
lbTipo	private JLabel
lFamiliasE	private JList
lFamiliasER	private JList
lFamiliasS	private JList
lFamiliasSR	private JList
lister	private ArrayList
listsr	private ArrayList
pnClasificacionSeccion	private JPanel
pnMostrar	private JPanel
pnRestringirFa	private JPanel
pnRestringirFaS	private JPanel
spClasificacionSeccion	private JScrollPane
taDescripcion	private JTextArea
tClasificacionSeccion	private JTable
tfNombre	private JTextField
tfValor	private JTextField
Para cada responsabilidad:	
Nombre:	**buscar***():void*
Descripción:	Método para buscar las clasificaciones de secciones de la BD
Nombre:	**clear***():void*
Descripción:	Método para limpiar los componentes del formulario
Nombre:	**coincidentes***():Object[]*
Descripción:	Método auxiliar para encontrar clasificaciones de secciones coincidentes.

Nombre:	**copyProperties***(ATipoSeccion tp):ATipoSeccion*
Descripción:	Captura las propiedades de un objeto.
Nombre:	**createJTable***(List list):void*
Descripción:	Método para crear la tabla con las clasificaciones de secciones de la BD
Nombre:	**EnClasificacionSeccionForm***():*
Descripción:	Constructor de la clase
Nombre:	**getClasifsecAction***():ClasificacionSeccionesAction*
Descripción:	Devuelve el valor del atributo clasifsecAction en dicha clase.
Nombre:	**inicializar***():void*
Descripción:	Método para llenar los comboboxs de elementos de la BD
Nombre:	**listGFTipoSeccion**(*ATipoSeccion tipoS):Set*
Descripción:	*Método para asociar las familias de entrada, de salida o ambas*
Nombre:	**nuevaCS***():void*
Descripción:	Este método se utiliza para mostrar el formulario.
Nombre:	**setClasifsecAction***(ClasificacionSeccionesAction clasifsecAction):void*
Descripción:	Modifica el valor del atributo clasifsecAction en dicha clase.

Nombre: ClasificacionSeccionesAction	
Tipo de clase: interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**deleteTipoSeccion***(ATipoSeccion tipoSesion):boolean*
Descripción:	Declaración

Nombre: ClasificacionSeccionesActionImpl	
Tipo de clase: controladora	
Atributo	**Tipo**
service	private TipoSeccionService
Para cada responsabilidad:	
Nombre:	**ClasificacionSeccionesActionImpl***(TipoSeccionService service):*
Descripción:	Constructor de la clase
Nombre:	**deleteTipoSeccion***(ATipoSeccion tipoSesion):boolean*
Descripción:	Método para eliminar una clasificación de secciones.
Nombre:	**getService***():TipoSeccionService*
Descripción:	Devuelve el valor del atributo service en dicha clase.
Nombre:	**setService***(TipoSeccionService service):void*
Descripción:	Modifica el valor del atributo service en dicha clase.

CU_Clientes:

Nombre: EnClientesForm	
Tipo de clase: interfaz	
Atributo	**Tipo**
accion	String
btnCancelar	private JButton
btnCrear	private JButton
btnEliminar	private JButton
btnIr	private JButton
cbBuscar	private JComboBox
chbActivo	private JCheckBox

idCliente	String
lbbuscarPor	private JLabel
lbCodigo	private JLabel
lbDireccion	private JLabel
lbEmail	private JLabel
lbNombre	private JLabel
lbTelefono	private JLabel
pnClientes	private JPanel
pnListar	private JPanel
spDireccion	private JScrollPane
spListacliente	private JScrollPane
taDireccion	private JTextArea
tfCodigo	private JFormattedTextField
tfEmail	private JTextField
tfNombre	private JTextField
tfparametro	private JTextField
tfTelefono	private JTextField
tListacliente	private JTable
Para cada responsabilidad:	
Nombre:	**buscar***():void*
Descripción:	Método para buscar los clientes de la BD
Nombre:	**createJTable***(List list):void*
Descripción:	Método para crear la tabla con los clientes de la BD
Nombre:	**EnClientesForm***()*
Descripción:	Constructor de la clase
Nombre:	**limpiar***():void*
Descripción:	Método para limpiar los componentes del formulario
Nombre:	**nuevoCliente***():void*
Descripción:	Este método se utiliza para mostrar el formulario.

CU_Especialidad_de_Proveedor:

Nombre: EnEspecialidadForm	
Tipo de clase: interfaz	
Atributo	**Tipo**
accion	String
activo	boolean
btnAdicionar	private JButton
btnCancelar	private JButton
btnCrear	private JButton
btnEliminar	private JButton
btnIr	private JButton
btnQuitar	private JButton
cbBuscar	private JComboBox
cbEspecialidades	private JComboBox
chbActivo	private JCheckBox
codigoEP	private String
enEspecialidadPanel	private JPanel
espproveedorAction	private EspecialidadProveedorAction
familiasAsig	private Object[]
familiasNoAsig	private Object[]

ftfCodigo	private JFormattedTextField
lbAsociarfamilia	private JLabel
lbBuscarPor	private JLabel
lbCodigo	private JLabel
lbDescripcion	private JLabel
lbFamiliadisp	private JLabel
lbFamiliaselec	private JLabel
lbGrupos	private JLabel
lbNombre	private JLabel
lFamiliasdisp	private JList
lFamiliasselec	private JList
listemptyObject	private Object[]
pnListar	private JPanel
spEspecialidad	private JScrollPane
spFamiliasdisp	private JScrollPane
spFamiliasselec	private JScrollPane
taDescripcion	private JTextArea
tEspecialidad	private JTable
tfNombre	private JTextField
tfParametro	private JTextField

Para cada responsabilidad:	
Nombre:	**buscar***():void*
Descripción:	Método para buscar las especialidades de la BD
Nombre:	**createJTable***(List list):void*
Descripción:	Método para crear la tabla con las especialidades de la BD
Nombre:	**EnEspecialidadForm***()*
Descripción:	Constructor de la clase
Nombre:	**getEspproveedorAction***():EspecialidadProveedorAction*
Descripción:	Devuelve el valor del atributo espproveedorAction en dicha clase.
Nombre:	**inicializar***():void*
Descripción:	Método para llenar los comboboxs de elementos de la BD
Nombre:	**nuevaEspecialidad***():void*
Descripción:	Este método se utiliza para mostrar el formulario.
Nombre:	**setEspproveedorAction***(EspecialidadProveedorAction espproveedorAction):void*
Descripción:	Modifica el valor del atributo espproveedorAction en dicha clase.

Nombre: EspecialidadProveedorAction	
Tipo de clase: interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**deleteEspecialidadProveedor***(AEspecialidadProveedor especialidad):boolean*
Descripción:	Declaración

Nombre: EspecialidadProveedorActionImpl	
Tipo de clase: controladora	
Atributo	**Tipo**
service	private EspecialidadProveedorService
Para cada responsabilidad:	
Nombre:	**deleteEspecialidadProveedor***(AEspecialidadProveedor especialidad):boolean*

Descripción:	Método para eliminar una especialidad.
Nombre:	**EspecialidadProveedorActionImpl***(EspecialidadProveedorService service)*
Descripción:	Constructor de la clase
Nombre:	**getService***():EspecialidadProveedorService*
Descripción:	Devuelve el valor del atributo service en dicha clase.
Nombre:	**setService***(EspecialidadProveedorService):void*
Descripción:	Modificar el valor del atributo service en dicha clase.

CU_Grupo/Familia:

Nombre: EnGrupoForm	
Tipo de clase: interfaz	
Atributo	**Tipo**
btnCancelar	private JButton
btnCancelarFamilia	private JButton
btnCrear	private JButton
btnCrearFamilias	private JButton
btnEliminarFamilia	private JButton
btnEliminarGrupo	private JButton
btnIr	private JButton
btnIrFam	private JButton
btnSalvarFamilia	private JButton
cbCampo	private JComboBox
cbCampoFam	private JComboBox
chbActivo	private JCheckBox
chbActivo_1	private JCheckBox
chbInsumo	private JCheckBox
chbInsumo_1	private JCheckBox
chbUtil	private JCheckBox
chbUtil_1	private JCheckBox
familiaAction	private FamiliaAction
ftfCodgrupo	private JFormattedTextField
ftfFamilia	private JFormattedTextField
grupoAction	private GrupoAction
grupoinicial	private String
idFamilia	private Integer
internalFrame	private JInternalFrame
lbBuscarPor	private JLabel
lbBuscarPorFam	private JLabel
lbCodigo	private JLabel
lbCodigo_1	private JLabel
lbDescripcion	private JLabel
lbDescripcion_1	private JLabel
lbDesgaste	private JLabel
lbDesgaste_1	private JLabel
lbGrupo	private JLabel
lbNombre	private JLabel
lbNombre_1	private JLabel
pnFamilia	private JPanel
pnFamilia_1	private JPanel
pnGrupo	private JPanel

pnGrupo_1	private JPanel
spDescripcion	private JScrollPane
spDescripciongrupo	private JScrollPane
spFamilia	private JScrollPane
spGrupo	private JScrollPane
table	private JTable
taDescripcion	private JTextArea
taDescripcionFam	private JTextArea
tFamilia	private JTable
tfDesgaste	private JTextField
tfDesgaste_1	private JTextField
tfGrupo	private JTextField
tfNombre	private JTextField
tfNombre_1	private JTextField
tfValor	private JTextField
tfValorFam	private JTextField
tGrupo_1	private JTable
Para cada responsabilidad:	
Nombre:	**BuscarFamilia***():void*
Descripción:	Método para buscar las familias de la BD
Nombre:	**BuscarGrupo***():void*
Descripción:	Método para buscar los grupos de la BD
Nombre:	**clear***():void*
Descripción:	Método para limpiar los componentes del internaFrame de familia
Nombre:	**createJTable***(List list):void*
Descripción:	Método para crear la tabla con los grupos de la BD
Nombre:	**createJTableFamilia***(List lista):void*
Descripción:	Método para crear la tabla con las familias de la BD
Nombre:	**EnGrupoForm***():*
Descripción:	Constructor de la clase
Nombre:	**getFamiliaAction***():FamiliaAction*
Descripción:	Devuelve el valor del atributo familiaAction en dicha clase.
Nombre:	**getGrupoAction***():GrupoAction*
Descripción:	Devuelve el valor del atributo grupoAction en dicha clase.
Nombre:	**IniciarFamilia***():void*
Descripción:	Método para llenar los comboboxs de elementos de la BD
Nombre:	**limpiarCampos***():void*
Descripción:	Método para limpiar los componentes del formulario de grupo
Nombre:	**nuevoGrupo***():void*
Descripción:	Este método se utiliza para mostrar el formulario.
Nombre:	**setFamiliaAction***(FamiliaAction familiaAction):void*
Descripción:	Modificar el valor del atributo familiaAction en dicha clase.
Nombre:	**setGrupoAction***(GrupoAction grupoAction):void*
Descripción:	Modificar el valor del atributo grupoAction en dicha clase.

Nombre: GrupoAction	
Tipo de clase: interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**deleteGrupo***(AGrupo grupo):boolean*
Descripción:	Declaración

Nombre: GrupoActionImpl	
Tipo de clase: controladora	
Atributo	**Tipo**
service	private GrupoService
Para cada responsabilidad:	
Nombre:	**deleteGrupo***(AGrupo grupo):boolean*
Descripción:	Método para eliminar un grupo.
Nombre:	**getService***():GrupoService*
Descripción:	Devuelve el valor del atributo service en dicha clase.
Nombre:	**GrupoActionImpl***(GrupoService service)*
Descripción:	Constructor de la clase
Nombre:	**setService***(GrupoService service):void*
Descripción:	Modifica el valor del atributo service en dicha clase.

Nombre: FamiliaAction	
Tipo de clase: interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**deleteFamilia***(AFamilia familia):boolean*
Descripción:	Declaración

Nombre: FamiliaActionImpl	
Tipo de clase: controladora	
Atributo	**Tipo**
service	private FamiliaService
Para cada responsabilidad:	
Nombre:	**deleteFamilia***(AFamilia familia):boolean*
Descripción:	Método para eliminar una familia
Nombre:	**FamiliaActionImpl***(FamiliaService service)*
Descripción:	Constructor de la clase
Nombre:	**getService***():FamiliaService*
Descripción:	Devuelve el valor del atributo service en dicha clase.
Nombre:	**setService***(FamiliaService):void*
Descripción:	Modifica el valor del atributo service en dicha clase.

CU_Producto:

Nombre: EnProductoForm	
Tipo de clase: interfaz	
Atributo	**Tipo**
btnAdicionar	private JButton
btnCancelar	private JButton
btnEliminar	private JButton
btnEliminarE	private JButton
btnCrear	private JButton
btnCrearE	private JButton
btnIr	private JButton
btnUMEq	private JButton
cbCampobuscar	private JComboBox

cbFamilia	private JComboBox
cbGrupo	private JComboBox
cbTipo	private JComboBox
cbUM	private JComboBox
cbUMDisponibles	private JComboBox
chbActivo	private JCheckBox
chbControlaExistencia	private JCheckBox
codEqu	Object[]
equivalencias	Object[]
ftfCodigo	private JFormattedTextField
ftfValor	private JFormattedTextField
idProd	Integer
internalFrame	private JInternalFrame
isumsActivo	Object[]
lbBuscarPor	private JLabel
lbCodigo	private JLabel
lbDescripcion	private JLabel
lbFamilia	private JLabel
lbGrupo	private JLabel
lbTipo	private JLabel
lbUM	private JLabel
lbUMBase	private JLabel
lbUMDisponibles	private JLabel
lUMEquivalentes	private JLabel
pnGrupobuscar	private JPanel
pnProducto	private JPanel
pnUMEquivalentes	private JPanel
productoAction	private ProductoAction
spGrupo	private JScrollPane
spMostrarUMEquivalentes	private JScrollPane
taDescripcion	private JTextArea
tfNombre	private JTextField
tfUMBase	private JTextField
tfValorbuscar	private JTextField
tProductos	private JTable
Para cada responsabilidad:	
Nombre:	**Buscar** *():void*
Descripción:	Método para buscar los productos de la BD
Nombre:	**createJListE***():void*
Descripción:	Método para crear la lista con las unidades de medida equivalentes, de la BD
Nombre:	**createJTable** *(List list):void*
Descripción:	Método para crear la tabla con los productos de la BD
Nombre:	**EnProductoForm***():*
Descripción:	Constructor de la clase
Nombre:	**getProductoAction***():ProductoAction*
Descripción:	Devuelve el valor del atributo productoAction en dicha clase.
Nombre:	**inicializacion***():void*
Descripción:	Método para llenar los comboboxs de elementos de la BD
Nombre:	**limpiar***():void*
Descripción:	Método para limpiar los componentes del formulario
Nombre:	**nuevoProducto***():void*

Descripción:	Este método se utiliza para mostrar el formulario.
Nombre:	**setProductoAction***(ProductoAction productoAction):void*
Descripción:	Modifica el valor del atributo productoAction en dicha clase.

Nombre: ProductoAction	
Tipo de clase: interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**deleteProducto***(AProducto producto):boolean*
Descripción:	Declaración

Nombre: ProductoActionImpl	
Tipo de clase: controladora	
Atributo	**Tipo**
service	private ProductoService
Para cada responsabilidad:	
Nombre:	**deleteProducto***(AProducto producto):boolean*
Descripción:	Método para eliminar un producto.
Nombre:	**ProductoActionImpl** *(ProductoService service)*
Descripción:	Constructor de la clase
Nombre:	**getService***():ProductoService*
Descripción:	Devuelve el valor del atributo service en dicha clase.
Nombre:	**setService***(ProductoService):void*
Descripción:	Modifica el valor del atributo service en dicha clase.

CU_Proveedor:

Nombre: EnProveedorForm	
Tipo de clase: interfaz	
Atributo	**Tipo**
btnCancelar	private JButton
btnCrear	private JButton
btnEliminar	private JButton
btnIr	private JButton
btnMoverD	private JButton
btnMoverI	private JButton
cbCampo	private JComboBox
chbactivo	private JCheckBox
codProveedor	String
cuentasR	private Object[]
cuentasSR	private Object[]
especialidadesR	private Object[]
especialidadesSR	private Object[]
lbBuscarPor	private JLabel
lbCodigo	private JLabel
lbDireccion	private JLabel
lbEmail	private JLabel
lbEspecialidades	private JLabel
lbNacionalidad	private JLabel
lbNombre	private JLabel

lbTelefono	private JLabel
lEspecialidadesR	private JList
lEspecialidadesSR	private JList
noseleccionados	ArrayList
pnProveedor	private JPanel
pnRestringirEsp	private JPanel
proveedorAction	private ProveedorAction
scrollPane	private JScrollPane
seleccionados	ArrayList
spEspecialidad	private JScrollPane
spEspecialidadR	private JScrollPane
spProveedorPanel	private JScrollPane
taDireccion	private JTextArea
tfCodigo	private JTextField
tfEmail	private JTextField
tfNacionalidad	private JTextField
tfNombre	private JTextField
tfTelefonos	private JTextField
tfValor	private JTextField
tProveedor	private JTable
Para cada responsabilidad:	
Nombre:	**Buscar***():void*
Descripción:	Método para buscar los proveedores de la BD
Nombre:	**coincidentes***():Object[]*
Descripción:	Método auxiliar para encontrar elementos coincidentes (ProveedorEspProveedor).
Nombre:	**createJTable***(List list):void*
Descripción:	Método para crear la tabla con los proveedores de la BD
Nombre:	**EnProveedorForm***()*
Descripción:	Constructor de la clase
Nombre:	**getProveedorAction***():ProveedorAction*
Descripción:	Devuelve el valor del atributo proveedorAction en dicha clase.
Nombre:	**limpiar***():void*
Descripción:	Método para limpiar los componentes del formulario
Nombre:	**nuevoProveedor***():void*
Descripción:	Este método se utiliza para mostrar el formulario.
Nombre:	**setProveedorAction***(ProveedorAction proveedorAction):void*
Descripción:	Modifica el valor del atributo proveedorAction en dicha clase.

Nombre: ProveedorAction	
Tipo de clase: interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**delete***(BaseObject obj):boolean*
Descripción:	Declaración

Nombre: ProveedorActionImpl	
Tipo de clase: controladora	
Atributo	**Tipo**
proveedorService	private ProveedorService
Para cada responsabilidad:	

Nombre:	**delete***(BaseObject obj):boolean*
Descripción:	Método para eliminar un proveedor.
Nombre:	**ProveedorActionImpl***(ProveedorService proveedorService)*
Descripción:	Constructor de la clase

CU_Sección_de_Inventario:

Nombre: EnSeccionForm	
Tipo de clase: interfaz	
Atributo	**Tipo**
btnCancelar	private JButton
btnCrear	private JButton
btnEliminar	private JButton
btnIr	private JButton
cbCampo	private JComboBox
cbCentroCosto	private JComboBox
cbEmpresa	private JComboBox
cbMVI	private JComboBox
cbReferenciaDoc	private JComboBox
cbTipoSeccion	private JComboBox
chbActivo	private JCheckBox
chbValidarExistencia	private JCheckBox
codSec	private Integer
lbBuscarPor	private JLabel
lbCentroCosto	private JLabel
lbDescripcion	private JLabel
lbDireccion	private JLabel
lbEmpresa	private JLabel
lbMVI	private JLabel
lbNombre	private JLabel
lbReferenciaDoc	private JLabel
lbResponsable	private JLabel
lbTipoSeccion	private JLabel
lbValidarExistencia	private JLabel
pnSeccion	private JPanel
pnSeccionEn	private JPanel
seccionAction	private SeccionAction
spSeccionPn	private JScrollPane
spDescripcionseccion	private JScrollPane
spDireccionseccion	private JScrollPane
spSeccion	private JTextArea
taDescripcion	private JTextArea
taDireccion	private JTextArea
tfNombre	private JTextField
tfResponsable	private JTextField
tfValor	private JTextField
tSeccion	private JTable
Para cada responsabilidad:	
Nombre:	**buscar***():void*
Descripción:	Método para buscar las secciones de la BD
Nombre:	**createJTable***(List list):void*

Descripción:	Método para crear la tabla con las secciones de la BD
Nombre:	**EnSeccionForm***()*
Descripción:	Constructor de la clase
Nombre:	**getSeccionAction***():SeccionAction*
Descripción:	Devuelve el valor del atributo seccionAction en dicha clase.
Nombre:	**limpiar***():void*
Descripción:	Método para limpiar los componentes del formulario
Nombre:	**newEnSeccionForm***():void*
Descripción:	Este método se utiliza para mostrar el formulario.
Nombre:	**setSeccionAction***(SeccionAction seccionAction):void*
Descripción:	Modifica el valor del atributo seccionAction en dicha clase.

Nombre: SeccionAction	
Tipo de clase: interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**deleteSeccion** *(ASeccion seccion):boolean*
Descripción:	Declaración

Nombre: SeccionActionImpl	
Tipo de clase: controladora	
Atributo	**Tipo**
service	private SeccionService
Para cada responsabilidad:	
Nombre:	**deleteSeccion** *(ASeccion seccion):boolean*
Descripción:	Método para eliminar una sección.
Nombre:	**SeccionActionImpl***(SeccionService service)*
Descripción:	Constructor de la clase

CU_Tipo_de_Producto:

Nombre: EnTipoProductoForm	
Tipo de clase: interfaz	
Atributo	**Tipo**
accion	private String
btnCancelar	private JButton
btnCrear	private JButton
btnEliminar	private JButton
btnIr	private JButton
cbbuscar	private JComboBox
idProd	private Integer
lbBuscarPor	private JLabel
lbTipoprod	private JLabel
pnListar	private JPanel
pnTipoproducto	private JPanel
spListatipoprod	private JScrollPane
spTipoprod	private JScrollPane
taTipoprod	private JTextArea
tfParametro	private JTextField
tipoprodAction	private TipoProductoAction

tlistatipoprod	private JTable
Para cada responsabilidad:	
Nombre:	**buscar***():void*
Descripción:	Método para buscar los tipos de producto de la BD
Nombre:	**createJTable***(List list):void*
Descripción:	Método para crear la tabla con los tipos de producto de la BD
Nombre:	**EnTipoProductoForm***()*
Descripción:	Constructor de la clase
Nombre:	**getTipoprodAction***():TipoProductoAction*
Descripción:	Devuelve el valor del atributo tipoprodAction en dicha clase.
Nombre:	**limpiar***():void*
Descripción:	Método para limpiar los componentes del formulario
Nombre:	**nuevoTipoProducto** *():void*
Descripción:	Este método se utiliza para mostrar el formulario.
Nombre:	**setTipoprodAction***(TipoProductoAction tipoprodAction):void*
Descripción:	Modifica el valor del atributo tipoprodAction en dicha clase.

Nombre: TipoProductoAction	
Tipo de clase: interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**DeleteTipoProducto***(ATipoProducto tipoproducto):boolean*
Descripción:	Declaración

Nombre: TipoProductoActionImpl	
Tipo de clase: controladora	
Atributo	**Tipo**
service	private SeccionService
Para cada responsabilidad:	
Nombre:	**DeleteTipoProducto***(ATipoProducto tipoproducto):boolean*
Descripción:	Método para eliminar un tipo de producto.
Nombre:	**getService***():TipoProductoService*
Descripción:	Devuelve el valor del atributo service en dicha clase.
Nombre:	**setService***(TipoProductoService service):void*
Descripción:	Modifica el valor del atributo service en dicha clase.
Nombre:	**TipoProductoActionImpl***(TipoProductoService service)*
Descripción:	Constructor de la clase

CU_Unidad_de_Medida:

Nombre: EnUnidadMedidaForm	
Tipo de clase: interfaz	
Atributo	**Tipo**
btnCancelar	private JButton
btnCrear	private JButton
btnEliminar	private JButton
btnIr	private JButton
cbCampo	private JComboBox
cbMetrica	private JComboBox
chbActivo	private JCheckBox

codUM	private Integer
enUMPanel	private JPanel
lbBuscarPor	private JLabel
lbDescripcion	private JLabel
lbMetrica	private JLabel
lbNombre	private JLabel
pnUM	private JPanel
scrollPane	private JScrollPane
spUM	private JScrollPane
taDescripcion	private JTextArea
tfNombre	private JTextField
tfValor	private JTextField
tUM	private JTable
umAction	private UnidadMedidaAction
Para cada responsabilidad:	
Nombre:	**buscar***():void*
Descripción:	Método para buscar las unidades de medida de la BD
Nombre:	**createJTable***(List list):void*
Descripción:	Método para crear la tabla con las unidades de medida de la BD
Nombre:	**EnUnidadMedidaForm***()*
Descripción:	Constructor de la clase
Nombre:	**getUmAction***():UnidadMedidaAction*
Descripción:	Devuelve el valor del atributo umAction en dicha clase.
Nombre:	**nuevaUM***():void*
Descripción:	Este método se utiliza para mostrar el formulario.
Nombre:	**setUmAction***(UnidadMedidaAction umAction):void*
Descripción:	Modifica el valor del atributo umAction en dicha clase.

Nombre: UnidadMedidaAction	
Tipo de clase interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**eliminarUnidadMedida***(AUnidadMedida um):boolean*
Descripción:	Declaración

Nombre: UnidadMedidaActionImpl	
Tipo de clase: controladora	
Atributo	**Tipo**
service	private UnidadMedidaService
Para cada responsabilidad:	
Nombre:	**eliminarUnidadMedida***(AUnidadMedida um):boolean*
Descripción:	Método para eliminar una unidad de medida.
Nombre:	**getService***():UnidadMedidaService*
Descripción:	Devuelve el valor del atributo service en dicha clase.
Nombre:	**UnidadMedidaActionImpl***(UnidadMedidaService service)*
Descripción:	Constructor de la clase

2. 5 Algoritmos y Estructuras de datos utilizados.

Durante del proceso de implementación de cada una de las interfaces de usuario y para contribuir a la rapidez y eficiencia de las mismas, se utilizaron y desarrollaron un conjunto de clases reutilizables, con diversos métodos donde se utilizan algoritmos de conversión, validaciones, búsqueda, ordenamiento y que implementan las diferentes estructuras de datos que brinda el lenguaje Java.

Entre los algoritmos más importantes, desarrollados en los métodos de la clase *VisualUtil,* podemos destacar el utilizado en el método ***moveTo***:

```
public static Object[][] moveTo(Object[] val, Object[] izq, Object[] der) {

    List list1 = convertArrayObjectToList(izq);
    List list2 = convertArrayObjectToList(der);

    for (int i = 0; i < val.length; i++) {
        list1.remove(val[i]);
        if (!list2.contains(val[i]))
            list2.add(val[i]);
    }

    Object[][] obj = new Object[2][];
    obj[0] = list1.toArray();
    obj[1] = list2.toArray();

    return obj;

}
```

Esté algoritmo posibilita mover los elementos pasados como parámetros de un arreglo de objetos hacia otro arreglo de objetos, pasando como parámetros el arreglo de donde se moverán los elementos *(izq),* el arreglo hacia donde se moverán los elementos *(der)* y un arreglo con las posiciones a mover *(val)*.

Otro de los algoritmos relevantes de esta clase, es el utilizado en el método ***fieldValues***, que permite obtener el valor de un atributo de un objeto pasando como parámetros el objeto *(obj)* y el atributo *(field)*. Esto se se hace posible con la utilización de las clases contenidas en el paquete ***java.lang.reflect***.

```
public static Object fieldValues(Object obj, Field field) {

    try {

        String fieldName = field.getName();

        Method get = obj.getClass().getMethod(
                "get" + fieldName.substring(0, 1).toUpperCase()
                        + fieldName.substring(1), (Class[]) null);

        Object result = get.invoke(obj, (Object[]) null);

        if (result instanceof Boolean) {
            result = Boolean.parseBoolean(result.toString()) ? "SI" : "NO";
        }

        return result;

    } catch (Exception e) {
        return 0;
    }

}
```

El lenguaje Java provee diferentes estructuras de datos que facilitan el trabajo de los programadores. Entre las estructuras de datos utilizadas para desarrollar esta capa, se encuentran los arreglos de una y de dos dimensiones, así como las *Listas y Sets,* incluidos en los paquetes ***java.util.List*** y ***java.util.Set***, respectivamente. Estas estructuras nos permiten coleccionar y manejar objetos de forma eficiente. En el caso de los *Sets,* es preciso aclarar que es un tipo de lista que no contiene elementos repetidos.

2. 6 Proceso Personal del Software (PSP).

El Proceso Personal del Software fue definido por Watts S. Humphrey del Software y según Humphrey, es de vital importancia para la planificación del ingeniero de forma tal que le permita alcanzar sus objetivos y cumplimentar su trabajo con la calidad requerida. Brinda una serie de principios con los cuales se pueden realizar planes precisos, establece bancos de pruebas para medir la mejora del proceso personal, y determina el impacto que los cambios del proceso tienen sobre el rendimiento del ingeniero. [Humphrey, 2001]

El desarrollador debe tener presente aspectos como:

- Realizar de planes realizables.
- Intentar cumplir a cabalidad los planes.
- Controlar de manera eficiente del tiempo.

- Detectar los errores y eliminarlos.

Para cumplimentar con cada uno de los procesos que define el PSP para los desarrolladores, Humphrey recomienda la utilización del Cuaderno del Ingeniero. Un cuaderno de trabajo donde se van a registrar los elementos y datos necesarios que apoyarán la planificación y su cumplimiento. En este cuaderno se controlan los compromisos, todas las actividades realizadas y el tiempo en correspondencia para cada actividad, entre otros elementos, además debe ser flexible en cuanto a los sucesos imprevistos. [Humphrey, 2001]

Mediante la utilización de las tablas del cuaderno de registro de defectos y resumen semanal, que incluye el cuaderno del ingeniero, el desarrollador tiene un control estricto del tiempo dedicado a cada tarea durante la semana y una visión hacia donde debe inclinar sus esfuerzos para lograr cumplimentar la implementación con calidad, de cada uno de los CU; así como un control de los defectos detectados durante la implementación y el estado de los mismos. Además, con la utilización de la tabla para estimar el tamaño de la implementación, los programadores o desarrolladores, pueden controlar las LOC y estimar, de acuerdo con el tamaño del programa, el tiempo de desarrollo del mismo.

Cuaderno de registro de defecto						
Estudiante :	Carlos Felipe Pérez León				**Fecha Pograma**	27/03/07
Profesor :	Diosmani Meriño Hechavarría					16
Fecha	**Número**	**Tipo**	**Introducido**	**Eliminado**	**Tiempo de corrección (min)**	**Defecto corregido**
25/03/07	1	40	Implementación	25/03/07	1	X
Descripción	Error en la declaración de la variable. Variable duplicada.					
Fecha	**Número**	**Tipo**	**Introducido**	**Eliminado**	**Tiempo de corrección (min)**	**Defecto corregido**
26/03/07	2	80	Implementación	26/03/07	10	X
Descripción	Error en el bucle para recorrer la lista de elementos y eliminar un elemento.					
Fecha	**Número**	**Tipo**	**Introducido**	**Eliminado**	**Tiempo de corrección (min)**	**Defecto corregido**
26/03/07	3	100	Prueba	26/03/07	5	X
Descripción	Error en la validación del código. Tipo de validación inapropiado.					

Figura 2. 3: Ejemplo del Cuaderno de registro de defectos.

Resumen de actividades sem 13				
Tarea	Investigar	Codificar	Pruebas	Total
Fecha				
D (29-5)				
Domingo	0	220	15	235
Lunes	0	220	25	245
Martes	0	230	15	245
Miercoles	0	220	15	235
Jueves	0	220	15	235
Viernes	0	220	10	230
Sabado	0	220	15	235
Total	0	1550	110	**1660**

Resumen de actividades de la semana anterior				
Total	1703	4051	5754	11508
Media	377.48486	992.13477	1369.6196	2739.2393
Max	852	2300	1102	4254
Min	293	205	35	533

Resumen incluyendo la ultima semana				
Total	0	1550	110	1660
Media	188.74243	1271.0674	739.80981	2199.6196
Max	852	2300	1102	4254
Min	0	205	35	240

Resumen de actividades sem 14				
Tarea	Investigar	Codificar	Pruebas	Total
Fecha				
D (6-12)				
Domingo	0	230	10	240
Lunes	0	230	10	240
Martes	0	235	15	250
Miercoles	0	240	15	255
Jueves	0	230	10	240
Viernes	0	230	10	240
Sabado	0	237	5	242
Total	0	1632	75	**1707**

Resumen de actividades de la semana anterior				
Total	0	1550	110	1660
Media	188.74243	1271.0674	739.80981	2199.6196
Max	852	2300	1102	4254
Min	0	205	35	240

Resumen incluyendo la ultima semana				
Total	0	3182	185	3367
Media	94.371216	1451.5337	407.40491	1953.3098
Max	852	2300	1102	4254
Min	0	205	35	240

Resumen de actividades sem 15				
Tarea	Investigar	Codificar	Pruebas	Total
Fecha				
D (13-19)				
Domingo	0	200	0	200
Lunes	0	205	5	210
Martes	0	220	10	230
Miercoles	0	225	5	230
Jueves	0	210	15	225
Viernes	0	200	10	210
Sabado	0	0	0	0
Total	0	1260	45	**1305**

Resumen de actividades de la semana anterior				
Total	0	3182	185	3367
Media	94.371216	1451.5337	407.40491	1953.3098
Max	852	2300	1102	4254
Min	0	205	35	240

Resumen incluyendo la ultima semana				
Total	0	4442	230	4672
Media	47.185608	1355.7668	226.20245	1629.1549
Max	852	2300	1102	4254
Min	0	205	35	240

Resumen de actividades sem 16				
Tarea	Investigar	Codificar	Pruebas	Total
Fecha				
D (20-26)				
Domingo		220	5	225
Lunes	0	220	10	230
Martes	0	235	5	240
Miercoles	0	220	10	230
Jueves	0	230	5	235
Viernes	0	220	10	230
Sabado	0	230	5	235
Total	0	1575	50	**1625**

Resumen de actividades de la semana anterior				
Total	0	3182	185	3367
Media	47.185608	1451.5337	407.40491	1906.1242
Max	852	2300	1102	4254
Min	0	205	35	240

Resumen incluyendo la ultima semana				
Total	0	4757	4757	9514
Media	23.592804	1513.2668	1536.8596	3073.7193
Max	852	2300	1102	4254
Min	0	205	35	240

Figura 2. 4: Tablas de Resumen de las semanas 13, 14, 15 y 16.

Tiempos de desarrollo en la Implementación					
No	**Implementación**	**Tiempo**	**LOC**	**Min/LOC**	**Funciones**
	Generales(Presentación)				
1	BaseForm.java(Impl)	7	25	0.28	Instrucción simple
2	BaseInternalForm.java(Impl)	7	28	0.25	Instrucción simple
3	BaseBean.java(Impl)	2	7	0.2857143	Instrucción simple
4	MainWindow.java(formulario)	100	351	0.2849003	Instrucción simple
	Utiles(Presentación)				
5	ColumnNames	4	18	0.2222222	Instrucción simple
6	ComponentStyles	4	19	0.2105263	Instrucción simple
7	SearchParameters	3	18	0.1666667	Instrucción simple
8	Validate	310	325	0.9538462	Bucle For y Switch complejos
9	VisualUtils	205	213	0.9624413	Bucle For y Switch complejos
10	ErrorMessagesTypes	1	4	0.25	Instrucción simple
11	FormNames	10	36	0.2777778	Instrucción simple
	Configuraciones(Presentación)				
12	Configuracion (XML)	1	4	0.25	Instrucción simple
13	applicationContext-xml	1	4	0.25	Instrucción simple
14	RunAppContext.java(Impl)	8	25	0.32	Instrucción simple
15	RunInitialization.java(Impl)	8	21	0.3809524	Instrucción simple
16	InitializationForm.java(Interfaz)	20	59	0.3389831	Instrucción simple
	BaseAction				
17	BaseAction.java(Interfaz)	5	17	0.2941176	Instrucción simple
18	ZunBaseActionImpl.java(Impl)	18	51	0.3529412	Try simple
19	applicationContext-core.xml	45	50	0.9	Instrucción simple
	Modulo Administración (Presentación)				
20	AutenticarAction.java(Interfaz)	1	4	0.25	Instrucción simple
21	GestionarTerminalesAction.java(Interfaz)	1	3	0.3333333	Instrucción simple
22	GestionarRolesAction.java(Interfaz)	1	3	0.3333333	Instrucción simple
23	GestionarUsuarioAction.java(Interfaz)	1	5	0.2	Instrucción simple
24	AutenticarActionImpl.java(Impl)	15	46	0.326087	Try simple
25	GestionarTerminalesActionImpl.java(Impl)	5	17	0.2941176	Try simple
26	GestionarRolesActionImpl.java(Impl)	4	14	0.2857143	Try simple
27	GestionarUsuarioActionImpl.java(Impl)	6	20	0.3	Try simple
28	EnAutenticarForm.java(formulario)	90	99	0.9090909	Instrucción simple
29	EnTerminalForm.java(formulario)	230	245	0.9387755	Bucle For y Switch simples
30	EnRolesForm.java(formulario)	400	422	0.9478673	Bucle For y Switch simples
31	EnUserForm.java	322	328	0.9817073	Bucle For y Switch simples
32	applicationContext--administración	1	4	0.25	Instrucción simple
33	applicationContext-.administración-form-CarlosF.xml	17	29	0.5862069	Instrucción simple

	Modulo Nomencladores (Presentación)				
34	FamiliaAction.java(Interfaz)	0.2	1	0.2	Instrucción simple
35	GrupoAction.java(Interfaz)	0.2	1	0.2	Instrucción simple
36	EspecialidadProveedorAction.java(Interfaz)	0.2	1	0.2	Instrucción simple
37	ProductoAction.java(Interfaz)	0.2	1	0.2	Instrucción simple
38	ProveedorAction.java(Interfaz)	0.2	1	0.2	Instrucción simple
39	SeccionAction.java(Interfaz)	0.2	1	0.2	Instrucción simple
40	TipoProductoAction.java(Interfaz)	0.2	1	0.2	Instrucción simple
41	TipoSeccionAction.java(Interfaz)	0.2	1	0.2	Instrucción simple
42	UnidadMedidaAction.java(Interfaz)	0.2	1	0.2	Instrucción simple
43	FamiliaActionImpl.java(Impl)	1	4	0.25	Try simple
44	GrupoActionImpl.java(Impl)	1	4	0.25	Try simple
45	EspecialidadProveedorActionImpl.java(Impl)	1	4	0.25	Try simple
46	ProductoActionImpl.java(Impl)	1	4	0.25	Try simple
47	ProveedorActionImpl.java(Impl)	1	4	0.25	Try simple
48	SeccionActionImpl.java(Impl)	1	4	0.25	Try simple
49	TipoProductoActionImpl.java(Impl)	1	4	0.25	Try simple
50	TipoSeccionActionImpl.java(Impl)	1	4	0.25	Try simple
51	UnidadMedidaActionImpl.java(Impl)	1	4	0.25	Try simple
52	EnGrupoForm.java(formulario)	420	435	0.9655172	Bucle For y Switch complejos
53	EnClientesForm.java(formulario)	235	266	0.8834586	Bucle For y Switch complejos
54	EnEspecialidadForm.java(formulario)	393	437	0.8993135	Bucle For y Switch complejos
55	EnProductoForm.java(formulario)	425	471	0.9023355	Bucle For y Switch complejos
56	EnProveedorForm.java(formulario)	395	418	0.9449761	Bucle For y Switch complejos
57	EnSeccionForm.java(formulario)	215	303	0.709571	Bucle For y Switch complejos
58	EnTipoProductoForm.java(formulario)	120	177	0.6779661	Bucle For y Switch complejos
59	EnClasificacionSeccionForm.java(formulario)	415	489	0.8486708	Bucle For y Switch complejos
60	EnUnidadMedidaForm.java(formulario)	225	246	0.9146341	Bucle For y Switch complejos
61	applicationContext-nomencladores.xml	1	6	0.1666667	Instrucción simple
62	applicationContext-nomencladores-form-CarlosF.xml	50	91	0.5494505	Instrucción simple
	Total	4757.8	5898	26.679883	
	Media	76.73871	95.129032	0.4303207	

2. 7 Conclusiones del capítulo.

En este capítulo se abordó la propuesta de solución, realizándose una descripción de la estructura de la capa de Presentación y su integración con la capa de Negocio del sistema, acorde con la arquitectura planteada para el SIGIA. Se presenta el diseño de las clases persistentes y se describen las clases de la capa de Presentación, que intervienen en la implementación de cada caso de uso que conforman los módulos Administración y Nomencladores, respectivamente. Además se hizo un análisis de algunos algoritmos utilizados en la implementación de las estructuras de datos utilizadas, para brindar mayores funcionalidades a las interfaces.

Se expone el trabajo realizado por el programador, en el Proceso Personal del Software (PSP), donde se puede medir la eficiencia del programador y estimar el tiempo de duración del flujo de trabajo.

3

Capítulo 3: Validación de la solución propuesta.

3. 1 Introducción al capítulo.

En el presente capítulo se exponen los resultados de las pruebas de validación de interfaces realizadas a cada uno de los CU implementados por el programador en la capa de Presentación de los módulos Administración y Nomencladores.

3. 2 Pruebas de Validación de Interfaces de Usuario.

Las pruebas de validación de las interfaces de usuario básicamente chequean para ver si los tipos y rangos de los valores de un campo son correctos. Por ejemplo, si un campo numérico es realmente un número o si una fecha es realmente una fecha válida, etc. La validación de datos es cuando necesitamos chequear si el valor dado es válido entre, digamos, una lista de posibles elecciones, quizás requiriendo una búsqueda en una base de datos. [Juristo - Moreno –Vegas, 2006]

A continuación se muestran las pruebas de validación de interfaces, para cada una de las interfaces de usuario de los módulos Administración y Nomencladores, que certifican la calidad de estas interfaces:

3. 2. 1 Módulo Administración.

CU_Autenticar:

Caso de uso: CU_Autenticar
Caso de prueba: Verificar la autenticación de un usuario.
Entrada: • Se introduce el usuario en un JTextField • Se introduce la contraseña en un JPasswordField
Resultado: • Se presiona el botón Aceptar y la aplicación muestra el menú principal. • En caso de que el usuario no exista en la Base de Datos se muestra un mensaje de error indicando que ese usuario no está registrado. • Si la contraseña introducida no coincida con la contraseña para el usuario, se muestra un mensaje de error, indicando que la contraseña es incorrecta.
Condiciones: • La estación de trabajo debe estar registrada y activa.
Elaborado por: Carlos Felipe Pérez León

CU_Gestionar_Roles:

Caso de uso: CU_Gestionar_Roles
Caso de prueba: Verificar la creación de un rol.
Entrada: • Se introduce el nombre del rol en un JTextField • Se seleccionan los módulos para obtener las opciones y dar los permisos a ese rol. • Se seleccionan las opciones para ese rol, en cada uno de los módulos.
Resultado: • Se presiona el botón Crear y se limpia el formulario para dar la posibilidad de crear un nuevo rol, además se actualiza la tabla con todos los roles existentes en la BD. • En caso de que el nombre del rol coincida con algún rol existente en la Base de Datos, se muestra un mensaje de error indicando que el rol ya existe.

Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Las opciones deben estar registradas en la BD.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Gestionar_Roles
Caso de prueba: Verificar la modificación de un rol.
Entrada: • Se seleccionan los módulos para obtener las opciones y dar los permisos a ese rol, en caso de que se vayan a modificar esos campos. • Se seleccionan las opciones para ese rol, en cada uno de los módulos.
Resultado: • Se presiona el botón Guardar y se limpia el formulario para dar la posibilidad de crear un nuevo rol, además se actualiza la tabla con todos los roles existentes en la BD.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Las opciones deben estar registradas en la BD. • Debe seleccionar un rol de la tabla de los roles registrados.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Gestionar_Roles
Caso de prueba: Verificar la eliminación de un rol.
Entrada:
Resultado: • Se presiona el botón Eliminar y se limpia el formulario para dar la posibilidad de crear un nuevo rol, además se actualiza la tabla con todos los roles existentes en la BD.

• En caso de que el rol esté asociado a un usuario, se muestra un mensaje de error indicando que no se puede eliminar el rol porque está asociado a un usuario.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar un rol de la tabla de los roles registrados.
Elaborado por: Carlos Felipe Pérez León

CU_Gestionar_Estaciones_de_Trabajo:

Caso de uso: CU_Gestionar_Estaciones_de_Trabajo
Caso de prueba: Verificar la creación de una estación de trabajo.
Entrada: • Se introduce el nombre de la estación de trabajo, en un JTextField • Se seleccionan si está activa o no, en un JCheckBox.
Resultado: • Se presiona el botón Crear y se limpia el formulario para dar la posibilidad de crear una nueva estación de trabajo, además se actualiza la tabla con todas las estaciones de trabajo existentes en la BD. • En caso de que el nombre de la estación de trabajo coincida con alguna estación de trabajo existente en la Base de Datos, se muestra un mensaje de error indicando que la estación de trabajo ya existe.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Gestionar_Estaciones_de_Trabajo
Caso de prueba: Verificar la modificación de una estación de trabajo.
Entrada: • Se seleccionan si está activa o no, en un JCheckBox, en caso de que se vaya a modificar este campo.

Resultado: • Se presiona el botón Guardar y se limpia el formulario para dar la posibilidad de crear una nueva estación de trabajo, además se actualiza la tabla con todas las estaciones de trabajo existentes en la BD.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar una estación de trabajo de la tabla de las estaciones de trabajo registradas.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Gestionar_Estaciones_de_Trabajo
Caso de prueba: Verificar la eliminación de una estación de trabajo.
Entrada:
Resultado: • Se presiona el botón Eliminar y se limpia el formulario para dar la posibilidad de crear una nueva estación de trabajo, se actualiza la tabla con todas las estaciones de trabajo existentes en la BD. • En caso de que la estación de trabajo esté activa o en uso, se muestra un mensaje de error indicando que no se puede eliminar la estación de trabajo por estas razones.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar una estación de trabajo de la tabla de las estaciones de trabajo registradas.
Elaborado por: Carlos Felipe Pérez León

CU_Gestionar_Usuarios:

Caso de uso: CU_Gestionar_Usuarios
Caso de prueba: Verificar la creación de un usuario.

Entrada:

- Se introduce el nombre del usuario en un JTextField
- Se introduce el primer apellido del usuario en un JTextField
- Se introduce el segundo apellido del usuario en un JTextField
- Se introduce el número de carnet de identidad del usuario en un JFormattedTextField.
- Se introduce el cargo del usuario en un JTextField.
- Se introduce el identificador (user) del usuario en un JTextField
- Se introduce la contraseña del usuario en un JPasswordField.
- Se seleccionan el rol para el usuario.

Resultado:

- Se presiona el botón Crear y se limpia el formulario para dar la posibilidad de crear un nuevo usuario, además se actualiza la tabla con todos los usuarios existentes en la BD.
- En caso de que el campo del nombre esté vacio, se muestra un mensaje indicando que es un valor requerido.
- En caso de que el user coincida con algún user existente en la Base de Datos, se muestra un mensaje de error indicando que el usuario ya existe.
- En caso de que el user, el nombre, el primer apellido, el segundo apellido, el cargo del usuario no cumplan con el formato correspondiente, se muestra un mensaje de error indicando no es una cadena con caracteres válidos.
- En caso de que la contraseña no contenga como mínimo 5 caracteres, se muestra un mensaje de error indicando que debe contener 5 o más caracteres.

Condiciones:

- El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción.
- Debe existir al menos un rol registrado para asignárselo al usuario.

Elaborado por:
Carlos Felipe Pérez León

Caso de uso: CU_Gestionar_Usuarios

Caso de prueba: Verificar la modificación de un usuario.

Entrada: • Se introduce el nuevo cargo del usuario en un JTextField. • Se presiona el botón Cambiar contraseña para desplegar el panel de cambiar la contraseña del usuario. • Se introduce la contraseña anterior del usuario en un JPasswordField. • Se introduce la nueva contraseña del usuario en un JPasswordField. • Se introduce la confirmación de la nueva contraseña del usuario en un JPasswordField. • Se presiona el botón Modificar en el panel de cambiar la contraseña. • Se coloca la nueva contraseña en el JPasswordField del formulario principal • Se seleccionan el rol para el usuario.
Resultado: • Se presiona el botón Guardar y se limpia el formulario para dar la posibilidad de crear un nuevo usuario, además se actualiza la tabla con todos los usuarios existentes en la BD. • En caso de que la contraseña anterior del usuario sea la incorrecta, se muestra un mensaje de error indicando que debe corregir la contraseña. • En caso de que la contraseña nueva no coincida con su confirmación, se muestra un mensaje de error indicando que debe corregir la contraseña. • En caso de que la contraseña no contenga como mínimo 5 caracteres, se muestra un mensaje de error indicando que debe contener 5 o más caracteres.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar un usuario de la tabla de los usuarios registrados.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Gestionar_Usuarios
Caso de prueba: Verificar la eliminación de un usuario.
Entrada:

Resultado: • Se presiona el botón Eliminar y se limpia el formulario para dar la posibilidad de crear una nueva estación de trabajo, además se actualiza la tabla con todas las estaciones de trabajo existentes en la BD.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar un usuario de la tabla de los usuarios registrados.
Elaborado por: Carlos Felipe Pérez León

3. 2. 2 Módulo Nomencladores.

CU_Clasificación_de_Secciones:

Caso de uso: CU_Clasificación_de_Secciones
Caso de prueba: Verificar la creación de una sección.
Entrada: • Se introduce el nombre de la clasificación de secciones, en un JTextField. • Se seleccionan si está activa o no, en un JCheckBox. • Se seleccionan si está incluye el arancel o no, en un JCheckBox. • Se seleccionan el tipo de sección. • Se introduce la descripción de la clasificación de sección en un JTextArea. • Se selecciona un grupo para asociar la(s) familia(s) de productos, de entrada y de salida para esta clasificación de secciones.
Resultado: • Se presiona el botón Crear y se limpia el formulario para dar la posibilidad de crear una nueva clasificación, además se actualiza la tabla con todas las nueva clasificaciones existentes en la BD. • En caso de que el campo del nombre esté vacio, se muestra un mensaje indicando que es un valor requerido. • En caso de que el nombre coincida con alguna clasificación registrada, se muestra un mensaje indicando que la clasificación ya existe. • En caso de que no se haya asociado al menos una familia de productos de entrada y una familia de productos de salida, se muestra un mensaje de error indicando que debe asociar las familias de productos.

Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe existir al menos un grupo registrado que contenga al menos una familia de productos registrada.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Clasificación_de_Secciones
Caso de prueba: Verificar la modificación de una sección.
Entrada: • Se seleccionan si está activa o no, en un JCheckBox en caso de que se vaya a modificar este campo. • Se seleccionan si está incluye el arancel o no, en un JCheckBox en caso de que se vaya a modificar este campo. • Se seleccionan el tipo de sección en caso de que se vaya a modificar este campo. • Se introduce la descripción de la clasificación de sección en un JTextArea en caso de que se vaya a modificar este campo.
Resultado: • Se presiona el botón Visualizar para desplegar el panel donde se muestra una tabla con todas las clasificaciones de secciones registradas en la BD. • Se presiona el botón Modificar y se cierra el panel dando paso al formulario para modificar la clasificación. • Se presiona el botón Guardar y se limpia el formulario para dar la posibilidad de crear una nueva clasificación y se actualiza la tabla en el panel de visualizar.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar una clasificación de la tabla de las clasificaciones de secciones registradas.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Clasificación_de_Secciones
Caso de prueba: Verificar la eliminación de una sección.
Entrada:
Resultado: • Se presiona el botón Visualizar para desplegar el panel donde se muestra una tabla con todas las clasificaciones de secciones registradas en la BD. • Se presiona el botón Eliminar y se actualiza la tabla con de las clasificaciones de secciones registradas. • En caso de que la clasificación esté activa, se muestra un mensaje de error indicando que no se puede eliminar.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar una clasificación de la tabla de las clasificaciones de secciones registradas.
Elaborado por: Carlos Felipe Pérez León

CU_Clientes:

Caso de uso: CU_Clientes
Caso de prueba: Verificar la creación de un cliente.
Entrada: • Se introduce el código del cliente en un JFormattedTextField. • Se introduce el nombre del cliente en un JTextField. • Se introduce la dirección del cliente en un JTextArea. • Se introduce la dirección del correo electrónico del cliente en un JTextField. • Se introduce número de teléfono del cliente en un JTextField. • Se seleccionan si está activo o no, en un JCheckBox.
Resultado: • Se presiona el botón Crear y se limpia el formulario para dar la posibilidad de crear un nuevo rol, además se actualiza la tabla con todos los roles existentes en la BD. • En caso de que el código del cliente coincida con algún código de cliente existente en la Base de Datos, se muestra un mensaje de error indicando que el cliente con ese código ya existe.

• En caso de que el campo del nombre esté vacio, se muestra un mensaje indicando que es un valor requerido. • En caso de que el campo del número telefónico contenga caracteres no numéricos, se muestra un mensaje de error indicando que no contiene caracteres numéricos válidos. • En caso de que el campo de la dirección de correo electrónico no esté en el formato correcto, se muestra un mensaje de error indicando los elementos incorrectos.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Clientes
Caso de prueba: Verificar la modificación de un cliente.
Entrada: • Se introduce la dirección del cliente en un JTextArea en caso de que se vaya a modificar este campo. • Se introduce la dirección del correo electrónico del cliente en un JTextField en caso de que se vaya a modificar este campo. • Se introduce número de teléfono del cliente en un JTextField en caso de que se vaya a modificar este campo. • Se seleccionan si está activo o no, en un JCheckBox en caso de que se vaya a modificar este campo.
Resultado: • Se presiona el botón Guardar y se limpia el formulario para dar la posibilidad de crear un nuevo cliente, además se actualiza la tabla con todos los clientes existentes en la BD. • En caso de que se haya modificado el número de teléfono y el campo del número telefónico contenga caracteres no numéricos, se muestra un mensaje de error indicando que no contiene caracteres numéricos válidos. • En caso de que se haya modificado el correo electrónico y el campo de la dirección de correo electrónico no esté en el formato correcto, se muestra un mensaje de error indicando los elementos incorrectos.

Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar un cliente de la tabla de los clientes registrados.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Clientes
Caso de prueba: Verificar la eliminación de un cliente.
Entrada:
Resultado: • Se presiona el botón Eliminar y se limpia el formulario para dar la posibilidad de crear un nuevo cliente, además se actualiza la tabla con todos los clientes existentes en la BD. • En caso de que el cliente esté activo, se muestra un mensaje de error indicando que no se puede eliminar por esta razón.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar un cliente de la tabla de los clientes registrados.
Elaborado por: Carlos Felipe Pérez León

CU_Especialidad_de_Proveedor:

Caso de uso: CU_Especialidad_de_Proveedor
Caso de prueba: Verificar la creación de una especialidad de proveedor.
Entrada: • Se introduce el código de la especialidad de proveedor, en un JFormattedTextField. • Se introduce el nombre de la especialidad de proveedor, en un JTextField.

• Se introduce la descripción de la especialidad de proveedor en un JTextArea. • Se seleccionan si está activa o no, en un JCheckBox. • Se selecciona un grupo para asociar la(s) familia(s) de productos a esta especialidad de proveedor. • Se seleccionan la(s) familia(s) a asociar a la especialidad.
Resultado: • Se presiona el botón Crear y se limpia el formulario para dar la posibilidad de crear una nueva especialidad de proveedor, además se actualiza la tabla con todas las especialidades de proveedor existentes en la BD. • En caso de que el campo del código de la especialidad esté vacio o contenga valores que no se ajustan al formato correcto, se muestra un mensaje de error indicando que es un valor requerido o que no contiene una cadena con caracteres numéricos válidos. • En caso de que el campo del nombre de la especialidad esté vacio o contenga valores que no se ajustan al formato correcto, se muestra un mensaje de error indicando que es un valor requerido o que no contiene una cadena con caracteres válidos. • En caso de que el código o el nombre coincidan con los valores de alguna especialidad registrada, se muestra un mensaje indicando que la especialidad ya existe. • En caso de que no se haya asociado al menos una familia de productos se muestra un mensaje de error indicando que debe asociar la(s) familia(s) de productos.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe existir al menos un grupo registrado que contenga al menos una familia de productos registrada.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Especialidad_de_Proveedor
Caso de prueba: Verificar la modificación de una especialidad de proveedor.

Entrada: • Se introduce la descripción de la especialidad en caso de que se vaya a modificar este campo. • Se seleccionan si está activa o no, en un JCheckBox en caso de que se vaya a modificar este campo. • Se selecciona un grupo para asociar la(s) familia(s) de productos a esta especialidad de proveedor, en caso de que se vayan a modificar la(s) familia(s) de productos. • Se seleccionan la(s) familia(s) de productos a asociar o a quitar de la especialidad.
Resultado: • Se presiona el botón Guardar y se limpia el formulario para dar la posibilidad de crear una nueva clasificación y se actualiza la tabla en el panel de visualizar.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar una especialidad de la tabla de las especialidades de proveedor registradas. • Debe existir al menos un grupo registrado que contenga al menos una familia de productos registrada.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Especialidad_de_Proveedor
Caso de prueba: Verificar la eliminación de una especialidad de proveedor.
Entrada:
Resultado: • Se presiona el botón Eliminar y se actualiza la tabla con de las especialidades de proveedor registradas. • En caso de que la especialidad esté activa, se muestra un mensaje de error indicando que no se puede eliminar.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta

acción. • Debe seleccionar una especialidad de la tabla de las especialidades de proveedor registradas.
Elaborado por: Carlos Felipe Pérez León

CU_Grupo/Familia:

Caso de uso: CU_Grupo/Familia
Caso de prueba: Verificar la creación de un grupo.
Entrada: • Se introduce el código del grupo, en un JFormattedTextField. • Se introduce el nombre del grupo, en un JTextField. • Se introduce la descripción del grupo, en un JTextArea. • Se selecciona si está activa o no, en un JCheckBox. • Se selecciona si es de insumo o no, en un JCheckBox. • Se selecciona si es útil o no, en un JCheckBox. • Se introduce el valor del desgaste con un valor real válido.
Resultado: • Se presiona el botón Crear y se limpia el formulario para dar la posibilidad de crear un nuevo grupo de productos, además se actualiza la tabla con todos los grupos existentes en la BD. • En caso de que los campos del código o nombre del grupo estén vacios o contengan valores que no se ajustan al formato correcto, se muestra un mensaje de error indicando que es un valor requerido o que no contiene una cadena con caracteres válidos. • En caso de que se seleccione útil, se desmarca el insumo si está seleccionado. • En caso de que el valor del desgaste no contenga un valor real válido, se muestra un mensaje de error indicando que debe cumplir con esta condición.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Grupo/Familia
Caso de prueba: Verificar la modificación de un grupo.
Entrada: • Se introduce la descripción del grupo, en un JTextArea, en caso de que se vaya a modificar este campo. • Se seleccionan si está activo o no, en un JCheckBox en caso de que se vaya a modificar este campo. • Se selecciona si es de insumo o no, en un JCheckBox en caso de que se vaya a modificar este campo. • Se selecciona si es útil o no, en un JCheckBox en caso de que se vaya a modificar este campo. • Se introduce el valor del desgaste con un valor real válido.
Resultado: • Se presiona el botón Guardar y se limpia el formulario para dar la posibilidad de crear un nuevo grupo, además se actualiza la tabla con todos los grupos existentes en la BD. • En caso de que se seleccione útil, se desmarca el insumo si está seleccionado. • En caso de que el valor del desgaste no contenga un valor real válido, se muestra un mensaje de error indicando que debe cumplir con esta condición.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar un grupo de la tabla de los grupos registrados.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Grupo/Familia
Caso de prueba: Verificar la eliminación de un grupo.
Entrada:
Resultado: • Se presiona el botón Eliminar y se limpia el formulario para dar la posibilidad de crear un nuevo grupo, además se actualiza la tabla con todos los grupos existentes en la

BD. • En caso de que el grupo esté activo o tenga alguna familia asociada, se muestra un mensaje de error indicando que no se puede eliminar por esta razón.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar un grupo de la tabla de los grupos registrados.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Grupo/Familia
Caso de prueba: Verificar la creación de una familia de productos.
Entrada: • Se introduce el código de la familia de productos, en un JFormattedTextField. • Se introduce el nombre de la familia de productos, en un JTextField. • Se introduce la descripción de la familia de productos en un JTextArea. • Se selecciona si está activa o no, en un JCheckBox. • Se selecciona si es de insumo o no, en un JCheckBox. • Se selecciona si es útil o no, en un JCheckBox. • Se introduce el valor del desgaste con un valor real válido.
Resultado: • Se presiona el botón Crear y se limpia el formulario para dar la posibilidad de crear una nueva familia de productos, además se actualiza la tabla con todas las familias de productos existentes en la BD. • En caso de que los campos del código o nombre de la familia de productos estén vacios o contengan valores que no se ajustan al formato correcto, se muestra un mensaje de error indicando que es un valor requerido o que no contiene una cadena con caracteres válidos. • En caso de que se seleccione útil, se desmarca el insumo si está seleccionado. • En caso de que el valor del desgaste no contenga un valor real válido, se muestra un mensaje de error indicando que debe cumplir con esta condición.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar

esta acción. • Debe existir al menos un grupo con al menos una familia de productos asociada. • Debe seleccionar un grupo para ver las familias de productos. • Debe presionar el botón Familias para desplegar el panel que contiene el formulario de las familias de productos.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Grupo/Familia
Caso de prueba: Verificar la modificación de una familia de productos.
Entrada: • Se introduce la descripción de la familia de producto, en un JTextArea, en caso de que se vaya a modificar este campo. • Se seleccionan si está activa o no, en un JCheckBox en caso de que se vaya a modificar este campo. • Se selecciona si es de insumo o no, en un JCheckBox en caso de que se vaya a modificar este campo. • Se selecciona si es útil o no, en un JCheckBox en caso de que se vaya a modificar este campo. • Se introduce el valor del desgaste con un valor real válido.
Resultado: • Se presiona el botón Guardar y se limpia el formulario para dar la posibilidad de crear una nueva clasificación y se actualiza la tabla en el panel de visualizar. • En caso de que se seleccione útil, se desmarca el insumo si está seleccionado. • En caso de que el valor del desgaste no contenga un valor real válido, se muestra un mensaje de error indicando que debe cumplir con esta condición.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe existir al menos un grupo con al menos una familia de productos asociada. • Debe seleccionar un grupo para ver las familias de productos. • Debe presionar el botón Familias para desplegar el panel que contiene el formulario de las familias de productos.

• Debe seleccionar una familia de productos de la tabla de las familias de productos registradas.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Grupo/Familia
Caso de prueba: Verificar la eliminación de una familia de productos.
Entrada:
Resultado: • Se presiona el botón Eliminar y se actualiza la tabla con de las familias de productos registradas. • En caso de que la familia de productos esté activa o asociada a un producto o a una especialidad de proveedor, se muestra un mensaje de error indicando que no se puede eliminar.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe existir al menos un grupo con al menos una familia de productos asociada. • Debe seleccionar un grupo para ver las familias de productos. • Debe presionar el botón Familias para desplegar el panel que contiene el formulario de las familias de productos. • Debe seleccionar una familia de productos de la tabla de las familias de productos registradas.
Elaborado por: Carlos Felipe Pérez León

CU_Producto:

Caso de uso: CU_Producto
Caso de prueba: Verificar la creación de un producto.
Entrada: • Se selecciona un grupo para buscar sus familias de productos. • Se selecciona una familia de productos. • Se introduce el código del producto, en un JFormatedTextField.

• Se introduce el nombre del producto, en un JTextField. • Se introduce la descripción del producto, en un JTextArea. • Se selecciona la unidad de mediad base para ese producto. • Se selecciona el tipo de producto para ese producto. • Se seleccionan si está activo o no, en un JCheckBox. • Se seleccionan si se controla la existencia o no, en un JCheckBox.
Resultado: • Se presiona el botón Crear y se limpia el formulario para dar la posibilidad de crear un nuevo producto, además se actualiza la tabla con todos los productos existentes en la BD. • En caso de que el código del cliente coincida con algún código de cliente existente en la Base de Datos, se muestra un mensaje de error indicando que el cliente con ese código ya existe. • En caso de que los campos del código o nombre del producto estén vacios o coincidan con alguno existente en la BD, se muestra un mensaje indicando que son valores requeridos o que ya existen. • En caso de que el campo del nombre contenga caracteres que no se correspondan con el formato, se muestra un mensaje de error indicando que no contiene caracteres válidos.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe existir al menos un grupo registrado que contenga al menos una familia de productos. • Debe existir al menos un tipo de producto registrado. • Debe existir al menos una unidad de medida registrada.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Producto
Caso de prueba: Verificar la modificación de un producto.

Entrada: • Se introduce la descripción del producto, en un JTextArea, en caso de que se vaya a modificar este campo. • Se selecciona la unidad de medida base, en caso de que se vaya a modificar este campo. • Se presiona el botón UM Equivalentes, para desplegar el panel donde se pueden asignar unidades de medidas equivalentes a la unidad de medida base del producto. Se selecciona la UM equivalente y se introduce la equivalencia, en caso de que se vaya a modificar este campo. • Se selecciona el tipo de producto, en caso de que se vaya a modificar este campo. • Se seleccionan si está activo o no, en un JCheckBox en caso de que se vaya a modificar este campo. • Se seleccionan si se controla la existencia o no, en un JCheckBox, en caso de que se vaya a modificar este campo.
Resultado: • Se presiona el botón Guardar y se limpia el formulario para dar la posibilidad de crear un nuevo producto, además se actualiza la tabla con todos los productos existentes en la BD. • En caso de que se hayan modificado las unidades de medidas equivalentes, se actualiza la lista de estas unidades de medida, en el panel correspondiente.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar un producto de la tabla de los productos registrados.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Producto
Caso de prueba: Verificar la eliminación de un producto.
Entrada:
Resultado: • Se presiona el botón Eliminar y se limpia el formulario para dar la posibilidad de crear

un nuevo cliente, además se actualiza la tabla con todos los clientes existentes en la BD. • En caso de que el producto esté activo, se muestra un mensaje de error indicando que no se puede eliminar por esta razón.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar un producto de la tabla de los productos registrados.
Elaborado por: Carlos Felipe Pérez León

CU_Proveedor:

Caso de uso: CU_Proveedor
Caso de prueba: Verificar la creación de un proveedor.
Entrada: • Se introduce el código del proveedor en un JFormattedTextField. • Se introduce el nombre del proveedor en un JTextField. • Se introduce la nacionalidad del proveedor en un JTextField. • Se introduce la dirección del proveedor en un JTextArea. • Se introduce la dirección del correo electrónico del proveedor en un JTextField. • Se introduce número de teléfono del proveedor en un JTextField. • Se seleccionan si está activo o no, en un JCheckBox. • Se selecciona(n) la(s) especialidad(es) de proveedor para asociar al proveedor.
Resultado: • Se presiona el botón Crear y se limpia el formulario para dar la posibilidad de crear un nuevo proveedor, además se actualiza la tabla con todos los proveedores existentes en la BD. • En caso de que el código del proveedor coincida con algún código de proveedor existente en la Base de Datos, se muestra un mensaje de error indicando que el proveedor con ese código ya existe. • En caso de que los campos del código o el nombre estén vacios, se muestra un mensaje indicando que son valores requeridos. • En caso de que el campo del nombre contenga caracteres que no estén en

correspondencia con el formato, se muestra un mensaje de error indicando que no contiene caracteres válidos. • En caso de que el campo del número telefónico contenga caracteres no numéricos, se muestra un mensaje de error indicando que no contiene caracteres numéricos válidos. • En caso de que el campo de la dirección de correo electrónico no esté en el formato correcto, se muestra un mensaje de error indicando los elementos incorrectos. • En caso de que no se haya asociado al menos una especialidad de proveedor, se muestra un mensaje de error indicando que debe asociar la(s) especialidad(es) de proveedor.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe existir al menos una especialidad de proveedor registrada, para asociarla al proveedor.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Proveedor
Caso de prueba: Verificar la modificación de un proveedor.
Entrada: • Se introduce la nacionalidad del proveedor en un JTextField en caso de que se vaya a modificar este campo. • Se introduce la dirección del proveedor en un JTextArea en caso de que se vaya a modificar este campo. • Se introduce la dirección del correo electrónico del proveedor en un JTextField en caso de que se vaya a modificar este campo. • Se introduce número de teléfono del proveedor en un JTextField en caso de que se vaya a modificar este campo. • Se seleccionan si está activo o no, en un JCheckBox, en caso de que se vaya a modificar este campo. • Se selecciona(n) la(s) especialidad(es) de proveedor para asociar al proveedor en caso de que se vaya a modificar este campo.
Resultado:

• Se presiona el botón Guardar y se limpia el formulario para dar la posibilidad de crear un nuevo cliente, además se actualiza la tabla con todos los clientes existentes en la BD. • En caso de que se haya modificado el número de teléfono y el campo del número telefónico contenga caracteres no numéricos, se muestra un mensaje de error indicando que no contiene caracteres numéricos válidos. • En caso de que se haya modificado el correo electrónico y el campo de la dirección de correo electrónico no esté en el formato correcto, se muestra un mensaje de error indicando los elementos incorrectos. • En caso de que no se haya asociado al menos una especialidad de proveedor, se muestra un mensaje de error indicando que debe asociar la(s) especialidad(es) de proveedor.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar un proveedor de la tabla de los clientes registrados. • Debe existir al menos una especialidad de proveedor registrada, para asociarla al proveedor.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Proveedor
Caso de prueba: Verificar la eliminación de un proveedor.
Entrada:
Resultado: • Se presiona el botón Eliminar y se limpia el formulario para dar la posibilidad de crear un nuevo proveedor, además se actualiza la tabla con todos los proveedores existentes en la BD. • En caso de que el proveedor esté activo, se muestra un mensaje de error indicando que no se puede eliminar por esta razón.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta

acción. • Debe seleccionar un proveedor de la tabla de los proveedores registrados.
Elaborado por: Carlos Felipe Pérez León

CU_Sección_de_Inventario:

Caso de uso: CU_Sección_de_Inventario
Caso de prueba: Verificar la creación de una sección de inventario.
Entrada: • Se introduce el nombre de la sección, en un JTextField. • Se introduce la descripción de la sección de inventario, en un JTextArea. • Se introduce el nombre del responsable de la sección de inventario, en un JTextField. • Se seleccionan la clasificación de la sección de inventario. • Se introduce la dirección de la sección de inventario en un JTextArea. • Se seleccionan la referencia del documento de la sección de inventario. • Se seleccionan el método de valoración de inventario (MVI) de la sección de inventario. • Se seleccionan la empresa para la sección de inventario. • Se seleccionan el centro de costo para la sección de inventario. • Se seleccionan si está activa o no, en un JCheckBox. • Se seleccionan si se valida la existencia o no, en un JCheckBox.
Resultado: • Se presiona el botón Crear y se limpia el formulario para dar la posibilidad de crear una nueva sección de inventario, además se actualiza la tabla con todas las secciones de inventarios existentes en la BD. • En caso de que el campo del nombre de la sección de inventario esté vacio, se muestra un mensaje de error indicando que es un valor requerido. • En caso de que el nombre coincida con el valor de alguna sección de inventario registrada, se muestra un mensaje indicando que la sección de inventario ya existe. • En caso de que el nombre del responsable de la sección de inventario esté vacio o contenga un valor que no esté acorde con el formato requerido, se muestra un mensaje de error indicando que es un valor requerido o que no contiene una cadena

con caracteres válidos.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe existir al menos una clasificación de secciones registrada.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Sección_de_Inventario
Caso de prueba: Verificar la modificación de una sección de inventario.
Entrada: • Se introduce la descripción de la sección de inventario, en caso de que se vaya a modificar este campo. • Se seleccionan la clasificación de la sección de inventario, en caso de que se vaya a modificar este campo. • Se introduce la dirección de la sección de inventario en un JTextArea, en caso de que se vaya a modificar este campo. • Se seleccionan la referencia del documento de la sección de inventario, en caso de que se vaya a modificar este campo. • Se seleccionan el método de valoración de inventario (MVI) de la sección de inventario en caso de que se vaya a modificar este campo. • Se seleccionan la empresa para la sección de inventario en caso de que se vaya a modificar este campo. • Se seleccionan el centro de costo para la sección de inventario en caso de que se vaya a modificar este campo. • Se seleccionan si está activa o no, en un JCheckBox en caso de que se vaya a modificar este campo. • Se seleccionan si se valida la existencia o no, en un JCheckBox en caso de que se vaya a modificar este campo.
Resultado: • Se presiona el botón Guardar y se limpia el formulario para dar la posibilidad de crear una nueva sección de inventario y se actualiza la tabla de las secciones de inventarios registradas.

Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar una sección de inventario de la tabla de las secciones de inventarios registradas. • Debe existir al menos una clasificación de secciones registrada.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Sección_de_Inventario
Caso de prueba: Verificar la eliminación de una sección de inventario.
Entrada:
Resultado: • Se presiona el botón Eliminar y se actualiza la tabla con de las secciones de inventarios registradas. • En caso de que la sección de inventario esté activa, se muestra un mensaje de error indicando que no se puede eliminar.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar una sección de inventario de la tabla de las secciones de inventarios registradas.
Elaborado por: Carlos Felipe Pérez León

CU_Tipo_de_Producto:

Caso de uso: CU_Tipo_de_Producto
Caso de prueba: Verificar la creación de un tipo de producto.
Entrada: • Se introduce el nombre del tipo de producto en un JTextArea.
Resultado: • Se presiona el botón Crear y se limpia el formulario para dar la posibilidad de crear un

nuevo tipo de producto, además se actualiza la tabla con todos los tipos de productos existentes en la BD. • En caso de que el campo del nombre esté vacio, se muestra un mensaje indicando que es un valor requerido. • En caso de que el nombre del tipo de producto coincida con algún tipo de producto registrado en la BD, se muestra un mensaje indicando que el tipo de producto ya existe.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Tipo_de_Producto
Caso de prueba: Verificar la modificación de un tipo de producto.
Entrada: • Se introduce el nombre del tipo de producto en un JTextArea en caso de que se vaya a modificar este campo.
Resultado: • Se presiona el botón Guardar y se limpia el formulario para dar la posibilidad de crear un nuevo tipo de producto, además se actualiza la tabla con todos los tipos de productos existentes en la BD. • En caso de que el tipo de producto esté asociado a un producto, se muestra un mensaje de error indicando que no se puede modificar. • En caso de que el campo del nombre esté vacio, se muestra un mensaje indicando que es un valor requerido.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar un tipo de producto de la tabla de los tipos de productos registrados.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Tipo_de_Producto
Caso de prueba: Verificar la eliminación de un tipo de producto.
Entrada:
Resultado: • Se presiona el botón Eliminar y se limpia el formulario para dar la posibilidad de crear un nuevo tipo de producto, además se actualiza la tabla con todos los tipos de productos existentes en la BD. • En caso de que el tipo de producto esté asociado a un producto, se muestra un mensaje de error indicando que no se puede eliminar.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar un tipo de producto de la tabla de los tipos de productos registrados.
Elaborado por: Carlos Felipe Pérez León

CU_Unidad_de_Medida:

Caso de uso: CU_Unidad_de_Medida
Caso de prueba: Verificar la creación de una unidad de medida.
Entrada: • Se introduce el nombre de la unidad de medida, en un JTextField. • Se selecciona la métrica de la unidad de medida. • Se introduce la descripción de la unidad de medida, en un JTextArea. • Se seleccionan si está activa o no, en un JCheckBox.
Resultado: • Se presiona el botón Crear y se limpia el formulario para dar la posibilidad de crear una nueva unidad de medida, además se actualiza la tabla con todas las unidades de medida existentes en la BD. • En caso de que el campo del nombre de la unidad de medida esté vacio, se muestra un mensaje de error indicando que es un valor requerido. • En caso de que el nombre coincida con el de alguna unidad de medida registrada, se muestra un mensaje indicando que la unidad de medida ya existe. • En caso de que el nombre contenga más de 10 caracteres, se muestra un mensaje de

error indicando que no debe exceder esta cantidad de caracteres.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Unidad_de_Medida
Caso de prueba: Verificar la modificación de una unidad de medida.
Entrada: • Se selecciona la métrica de la unidad de medida en caso de que se vaya a modificar este campo. • Se introduce la descripción de la unidad de medida en caso de que se vaya a modificar este campo. • Se seleccionan si está activa o no, en un JCheckBox en caso de que se vaya a modificar este campo.
Resultado: • Se presiona el botón Guardar y se limpia el formulario para dar la posibilidad de crear una nueva unidad de medida y se actualiza la tabla de las unidades de medida registradas.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar una unidad de medida de la tabla de las unidades de medida registradas.
Elaborado por: Carlos Felipe Pérez León

Caso de uso: CU_Unidad_de_Medida
Caso de prueba: Verificar la eliminación de una unidad de medida.
Entrada:
Resultado: • Se presiona el botón Eliminar y se actualiza la tabla con de las unidades de medida

registradas. • En caso de que la unidad de medida esté activa, se muestra un mensaje de error indicando que no se puede eliminar.
Condiciones: • El usuario debe haberse autenticado correctamente y tener permisos para realizar esta acción. • Debe seleccionar una unidad de medida de la tabla de las unidades de medida registradas.
Elaborado por: Carlos Felipe Pérez León

Es preciso destacar que para la realización de cada uno de los caso de prueba que tienen como objetivo eliminar o modificar algún objeto, se realiza una búsqueda por parámetros del mismo, teniendo en cuenta sus propiedades. Anexos XI y XII

3. 3 Conclusiones del capítulo.

En este capítulo se analizaron los resultados de las pruebas de validación de interfaz de usuario, realizadas a cada uno de los casos de uso que conforman los módulos Administración y Nomencladores. Mediante estas pruebas se pueden certificar cada una de estas interfaces que, de forma amigable, funcional y rápida, se integran con la capa de Negocio y culmina todo un proceso de desarrollo de estos dos módulos del SIGIA.

CONCLUSIONES GENERALES

- Se realizó un estudio acerca de las interfaces gráficas de usuario en los principales software de gestión de inventario.
- Se abordaron los principios básicos de la programación, además de las tareas fundamentales que debe desarrollar el programador, según la metodología seleccionada para el desarrollo del SIGIA.
- Se realizó un estudio sobre las herramientas definidas para el desarrollo del sistema.
- Teniendo en cuenta los patrones de arquitectura definidos para la capa de Presentación, se expone la estructura de esta capa para los módulos: “Administración y Nomencladores”.
- Se implementaron y validaron las interfaces de usuario, de los módulos: “Administración y Nomencladores”, integradas con la capa de Negocio desarrollada para el sistema.
- Se expone el resultado del trabajo del programador, acorde con la planificación, para el cumplimiento de la implementación y prueba de cada interfaz de usuario.

RECOMENDACIONES

- La implementación de la capa de Presentación, para los módulos restantes del SIGIA, teniendo en cuenta el estándar de codificación y la estructura de la capa, definida por el arquitecto.
- Evaluar alguna herramienta libre, que brinde iguales o mayores funcionalidades que el *WindowBuilder v5.0* y así culminar un producto desarrollado en su totalidad, con tecnologías no propietarias.

REFERENCIAS BIBLIOGRÁFICA

[IEEE, 1990] (IEEE), I. d. I. E. y. E. (1990) "A Compilation of IEEE Standard Computer Glossaries".

[Abelson - Sussman, 1996] Abelson, H. y. S., G.J. con Sussman, J. (1996). "Structure and Interpretation of Computer Programs". Url: http://mitpress.mit.edu/sicp/full-text/book/book.html.

[Bonanata, 2003] Bonanata, M. (2003). "Programación y Algoritmos".

[Booch, 1996] Booch, G. (1996). "Análisis y Diseño orientado a Objetos con Aplicaciones". Url: http://catalogo.info.unlp.edu.ar/cgi-bin/koha/opac-detail.pl?bib=1427.

[Bush, 1945] Bush, V. (1945). "As we may think", Url: http://web.mit.edu/STS.035/www/PDFs/think.pdf.

[Canchala, 2006] Canchala, A. (2006). "Fundamentos de la POO". Url: http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_2578.asp.

[Lindsey- Tolliver- Lindblad, 2005] Clark S. Lindsey, J. S. T., Thomas Lindblad (2005). "JavaTech, an Introduction to Scientific and Technical Computing with Java". Url: http://books.google.com/books?id=yL217mej8koC&pg=PA7&ots=7rzu3pazxL&dq=JAVA+2.+J2SE&sig=Q1V0rV932mOo2CuSADYEPWoGtmU.

[Clayberg, 2003] Clayberg, E. (2003). "Eclipse for Smalltalkers". Url: http://www.whysmalltalk.com/Smalltalk_Solutions/ss2003/pdf/clayberg.pdf.

[Collado, 2005] Collado, M. (2005). "Pruebas de Software". Url: http://lml.ls.fi.upm.es/~mcollado/ed2/pruebas.ppt.

[Calderón - Davis, 2000] Davis, M. C. y. E. (2000). "Swing, la solución actual de Java para crear GUIs". Url: http://www.dcc.uchile.cl/~lmateu/CC60H/Trabajos/edavis/swing.html.

[EclipseCorner, 2006] EclipseCorner. (2006). "WindowBuilder Pro 5.0 - "Better than Matisse" GroupLayout Support for SWT." Url: http://dev.eclipse.org/newslists/news.eclipse.commercial/msg00118.html.

[Fayad et al, 1999]Fayad, M., Schmidt, D., y Johnson, R. E. (1999). Application Frameworks., Wiley & Sons. 3.

[Figueroa, 2005] Figueroa, Z. J. H. (2005). "Fundamentos de estructuras de datos: soluciones en Ada, Java y C++". Url: http://books.google.com/books?id=KSE4HwF8BmcC&pg=PA458&ots=n49WibUYhd&dq=Fundamentos+de+la+programaci%C3%B3n.+Algoritmos+y+estructuras+de+datos&sig=vUhz093YzDxUti-2EWe42DInZEI.

[Floyd, 1979] Floyd. R. (1979). “The Paradigms of Programming”.

[Fuente, 2000] Fuente, J. S.-C. d. l. (2000). "Thomas Samuel Kuhn, Biografía". Url: http://www.webdianoia.com/contemporanea/kuhn.htm.

[Arnold - Gosling, 1997] Gosling., K. A. y. J. (1997). "El lenguaje de Programación Java".

[Ibañes – Gutiérrez, 1998] Gutiérrez, M. I. y. G. (1998). "Fundamentos de la Programación". Url: http://www.caos.inf.uc3m.es/asignaturas/pr_itig/apuntes/Prog_I_tema_2.pdf.

[Humphrey, 2001] Humphrey, W. S. (2001). "Introduction to the Personal Software Process". Url: http://bibliodoc.uci.cu/pdf/reg03058.pdf.

[Lago, 2006] Lago. (2006). "Eclipse Newcomers FAQ." Url: http://www.eclipse.org/home/newcomers.php.

[Pinson - Wiener, 2000] Lewis J. Pinson, R. S. W. (2000). "Fundamentals of OOP and Data Structures in Java", Url: http://books.google.com/books?id=m0vpxjdjUJYC&pg=PA3&ots=_n-NvsUMvS&dq=Fundamentals+of+Data+Structures&sig=2KdHbXgaDtN0sdPMtwB6kTfLcd4.

[Mañas, 2007] Mañas, J. A. (1999). "Prueba de Programas". Url: http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/testing.htm.

[Juristo-Moreno-Vegas, 2006] Natalia Juristo, A. M. M., Sira Vegas. (2006). "TÉCNICAS DE EVALUACIÓN DE SOFTWARE". Url: http://is.ls.fi.upm.es/udis/docencia/erdsi/Documentacion-Evaluacion-7.pdf.

[Horta-Ostos-Gutiérrez, 1991] Patricia Horta, M. O. y. O. G. (1991). "Diseño estructurado de algoritmos". Url: http://www.itver.edu.mx/comunidad/material/algoritmos/.

[Perera, 2007] Perera, J. R. (2007). ""Arquitectura de software para Sistema de Gestión de Inventarios", UCI.

[Puebla, 2006] Puebla, I. G. (2006). "Eclipse: una herramienta profesional al alcance de todos". Url: http://www.gui.uva.es/~laertes/nuke/index.php?option=com_content&task=view&id=54&Itemid=41.

Rational Software Corporation. (2003) "Rational Unified Process Help"

[Reimer, 2005] Reimer, J. (2005). "A History of the GUI", Url: http://arstechnica.com/articles/paedia/gui.ars.

[Lewis-Rieman, 1993] Rieman, C. L. y. J. (1993). "Task-centered user interface design". Url: http://hcibib.org/tcuid/.

Science, E. o. C. (2007). "Lenguajes de programación". Url: http://www.dccia.ua.es/dccia/inf/asignaturas/LPP/2006-2007/tema-01.html.

[Bobrow-Stefik, 1986] Stefik, D. G. B. y. M. J. (1986). "Perspectives on Artificial Intelligence Programming", 23 Url: http://www2.parc.com/istl/groups/hdi/papers/stefik-perspectives-on-AI-programming.pdf.

[Stephanidis, 2001] Stephanidis, C. (2001). "User Interfaces for All". Url: http://books.google.com/books?id=kGslhSbZqzkC&pg=PA3&ots=5VCbLMr-5H&dq=Interfaces&sig=sjh97iBXzUoXmkTU2hJ0lPaSCQA.

[Cormen-Leiserson-Rivest-Stein, 2001] Thomas H. Cormen, C. E. L., Ronald L. Rivest, y Cliff Stein. (2001). "Introduction to Algorithms", Url: http://books.google.com/books?id=NLngYyWFl_YC&dq=Introduction+to+Algorithms&pg=PP1&ots=BvUvIE5nB8&sig=uW-7hPRKgvIpBLmpLxF2jaqy0XY&prev=http://www.google.com/search%3Fhl%3Den%26q%3DIntroduction%2Bto%2BAlgorithms%26btnG%3DSearch&sa=X&oi=print&ct=title.

[Zukowsky, 2003] Zukowsky, J. (2003). "Programación. Java 2 J2SE".

BIBLIOGRAFÍA

(IEEE), I. d. I. E. y. E. (1990) "A Compilation of IEEE Standard Computer Glossaries".

Abelson, H. y. S., G.J. con Sussman, J. (1996). "Structure and Interpretation of Computer Programs". Url: http://mitpress.mit.edu/sicp/full-text/book/book.html.

Alarcón, R. (2003). "DISEÑO ORIENTADO A OBJETOS CON UML". Url: www.grupoeidos.com/www.eidos.es.

Alexander, C. (1980). "A PATTERN LANGUAGE". Url: http://downlode.org/etext/patterns/.

Bonanata, M. (2003). "Programación y Algoritmos".

Booch, G. (1996). "Análisis y Diseño orientado a Objetos con Aplicaciones". Url: http://catalogo.info.unlp.edu.ar/cgi-bin/koha/opac-detail.pl?bib=1427.

Bush, V. (1945). "As we may think", Url: http://web.mit.edu/STS.035/www/PDFs/think.pdf.

Canchala, A. (2006). "Fundamentos de la POO". Url: http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_2578.asp.

Clark S. Lindsey, J. S. T., Thomas Lindblad (2005). "JavaTech, an Introduction to Scientific and Technical Computing with Java". Url: http://books.google.com/books?id=yL217mej8koC&pg=PA7&ots=7rzu3pazxL&dq=JAVA+2.+J2SE&sig=Q1V0rV932mOo2CuSADYEPWoGtmU.

Clayberg, E. (2003). "Eclipse for Smalltalkers". Url: http://www.whysmalltalk.com/Smalltalk_Solutions/ss2003/pdf/clayberg.pdf.

Collado, M. (2005). "Pruebas de Software". Url: http://lml.ls.fi.upm.es/~mcollado/ed2/pruebas.ppt.

Davis, M. C. y. E. (2000). "Swing, la solución actual de Java para crear GUIs". Url: http://www.dcc.uchile.cl/~lmateu/CC60H/Trabajos/edavis/swing.html.

EclipseCorner. (2006). "WindowBuilder Pro 5.0 - "Better than Matisse" GroupLayout Support for SWT." Url: http://dev.eclipse.org/newslists/news.eclipse.commercial/msg00118.html.

Elizondo, P. I. V. (2001). "PRUEBA DE COMPONENTES DE SOFTWARE BASADAS EN EL MODELO DE JAVABEANS". Url: http://www.cs.man.ac.uk/~velascop/publ/Tesis.pdf.

Figueroa, Z. J. H. (2005). "Fundamentos de estructuras de datos: soluciones en Ada, Java y C++". Url: http://books.google.com/books?id=KSE4HwF8BmcC&pg=PA458&ots=n49WibUYhd&dq=Fundamentos+de+la+programaci%C3%B3n.+Algoritmos+y+estructuras+de+datos&sig=vUhz093YzDxUti-2EWe42DInZEI.

Fayad, M., Schmidt, D, y Johnson, R. E. (1999). Application Frameworks, Wiley & Sons. 3.

Floyd. R. (1979). “The Paradigms of Programming”.

Froufe, A. (1999). "Sugerencias de Codificación". Url: http://www.itapizaco.edu.mx/paginas/JavaTut/froufe/parteD/capd-1.html.

Fuente, J. S.-C. d. l. (2000). "Thomas Samuel Kuhn, Biografía". Url: http://www.webdianoia.com/contemporanea/kuhn.htm.

García, O. M. (2004). "Gestión Empresarial y TIC."

Gasarch, I. P. y. W. (2002). "Problems on Algorithms". Url: http://hercule.csci.unt.edu/ian/books/free/license.html.

Gosling. K. A. y. J. (1997). "El lenguaje de Programación Java".

Gutiérrez, M. I. y. G. (1998). "Fundamentos de la Programación". Url: http://www.caos.inf.uc3m.es/asignaturas/pr_itig/apuntes/Prog_I_tema_2.pdf.

Humphrey, W. S. (1995). "A Discipline for Software Engineering".

Humphrey, W. S. (2001). "Introduction to the Personal Software Process". Url: http://bibliodoc.uci.cu/pdf/reg03058.pdf.

Kuchana, P. (2004). "Patrones de Diseño en Java". Url: http://books.google.com/books?id=FqfKFl4Bm7UC&pg=PA3&ots=7MZsioU6DA&dq=Design+Patterns&sig=JdqEZsHb6Hy8_DX6tU9V5q15XyA.

Lago. (2006). "Eclipse Newcomers FAQ." Url: http://www.eclipse.org/home/newcomers.php.

León, R. A. H. and S. C. González (2002). EL PARADIGMA CUANTITATIVO DE LA INVESTIGACIÓN CIENTIFICA.

[Pinson - Wiener, 2000] Lewis J. Pinson, R. S. W. (2000). "Fundamentals of OOP and Data Structures in Java”, Url: http://books.google.com/books?id=m0vpxjdjUJYC&pg=PA3&ots=_n-NvsUMvS&dq=Fundamentals+of+Data+Structures&sig=2KdHbXgaDtN0sdPMtwB6kTfLcd4.

Lozada, H. R. G. B. y. Y. L. (2005). Sistema de Inventario Participativo de la UCI.

Mañas, J. A. (1999). "Prueba de Programas". Url: http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/testing.htm.

Morero, F. (2000). "Introducción a la OOP".

Natalia Juristo, A. M. M., Sira Vegas. (2006). "Técnicas de evaluación de Software”. Url: http://is.ls.fi.upm.es/udis/docencia/erdsi/Documentacion-Evaluacion-7.pdf.

Patricia Horta, M. O. y. O. G. (1991). "Diseño estructurado de algoritmos". Url: http://www.itver.edu.mx/comunidad/material/algoritmos/.

Perera, J. R. (2007). "Arquitectura de software para Sistema de Gestión de inventarios", UCI.

Puebla, I. G. (2006). "Eclipse: una herramienta profesional al alcance de todos". Url: http://www.gui.uva.es/~laertes/nuke/index.php?option=com_content&task=view&id=54&Itemid=41.

Rational Software Corporation. (2003) "Rational Unified Process Help"

Reimer, J. (2005). "A History of the GUI", Url: http://arstechnica.com/articles/paedia/gui.ars.

Rieman, C. L. y. J. (1993). "Task-centered user interface design". Url: http://hcibib.org/tcuid/.

Science, E. o. C. (2007). "Lenguajes de programación". Url: http://www.dccia.ua.es/dccia/inf/asignaturas/LPP/2006-2007/tema-01.html.

Sedgewick, R. (2002). "Algorithms in Java". Url: http://books.google.com/books?id=hyvdUQUmf2UC&pg=PP1&ots=kyXM63Ljx9&dq=ALGORITHMS%2BSedgewick&sig=cXP674f9YlD7y-y5W0PaUEJE3iY.

Stefik, D. G. B. y. M. J. (1986). "Perspectives on Artificial Intelligence Programming", 23 Url: http://www2.parc.com/istl/groups/hdi/papers/stefik-perspectives-on-AI-programming.pdf.

Stelting, S. (2003). "Applied Java patterns". Url: http://books.google.com/books?id=3kCw05_yNTIC&pg=PP1&ots=5L6Tte7COv&dq=Applied+Java+patterns&sig=Ga2F3tcPVahjJrb_VD4WF8JU2LU.

Stephanidis, C. (2001). "User Interfaces for All". Url: http://books.google.com/books?id=kGslhSbZqzkC&pg=PA3&ots=5VCbLMr-5H&dq=Interfaces&sig=sjh97iBXzUoXmkTU2hJ0lPaSCQA.

Thomas H. Cormen, C. E. L., Ronald L. Rivest, y Cliff Stein. (2001). "Introduction to Algorithms", Url: http://books.google.com/books?id=NLngYyWFl_YC&dq=Introduction+to+Algorithms&pg=PP1&ots=BvUvIE5nB8&sig=uW-7hPRKgvIpBLmpLxF2jaqy0XY&prev=http://www.google.com/search%3Fhl%3Den%26q%3DIntroduction%2Bto%2BAlgorithms%26btnG%3DSearch&sa=X&oi=print&ct=title.

Torres, D. M. H. y. D. F. (2005). "Sistema informático para problemas de distribución".

UCI (2007). "Conferencia de Diseño Metodológico de la Investigación".

UCI (2007). "Conferencia de Diseño Teórico de la Investigación".

Zukowsky, J. (2003). "Programación. Java 2 J2SE".