Universidad de las Ciencias Informáticas

Facultad 3

Título: “Diseño de la solución informática para el Control de Bienes en los Registros Públicos y Notarías de la República Bolivariana de Venezuela”.

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autores: Julio Cardoso Ventura

Johnny Pérez Acosta

Tutor: Ing. Henrik Pestano Pino

Consultor: Ing. Yosvany Márquez Ruíz

La Habana, Junio del 2008

DECLARACIÓN DE AUTORÍA

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

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

Julio Cardoso Ventura **Johnny Pérez Acosta**

_____ _____

Firma del Autor Firma del Autor

Ing. Henrik Pestano Pino

_____

Firma del Tutor

Opinión del Tutor sobre el trabajo de Diploma

Sobre el Trabajo de Diploma Presentado para Optar por el Título de Ingeniero en Ciencias Informáticas

Título: Diseño de la solución informática para el control de Bienes en los Registros Públicos y Notarías de la República Bolivariana de Venezuela

Autores: Julio Cardoso Ventura y Johnny Pérez Acosta

El tutor del presente Trabajo de Diploma considera que durante su ejecución los diplomantes han mostrado total independencia, creatividad y responsabilidad en la solución presentada. Basándose en conocimientos adquiridos a lo largo de su proceso de formación y aplicando las experiencias adquiridas durante su trabajo en el proyecto Registros y Notarías, han demostrando dominio en el manejo de técnicas de investigación y determinación al enfrentar diversas problemáticas que surgen en el desarrollo de Software.

El documento presenta calidad y ha sido desarrollado en menos tiempo del planificado cumpliendo con las normas de redacción y ortografía exigidas a este nivel. Además vale destacar como han realizado este trabajo de diploma a la par de las tareas de desarrollo de nuevos módulos así como las actividades docentes propias de la Facultad y Universidad.

El trabajo desarrollado cumple con los requisitos necesarios y objetivos trazados. El impacto que tendrá el producto del que forma parte este módulo sobrepasa el marco económico, pues representa un beneficio social digno de la hermana Revolución Bolivariana y de su pueblo.

Por todo lo anteriormente expresado se puede plantear que los diplomantes han cumplido los objetivos propuestos, por lo que le propongo el título de *Ingeniero en Ciencias Informáticas*, con una calificación máxima de 5 puntos.

_____ _____

Ing. Henrik Pestano Pino Fecha

Agradecimientos:

A todas las personas involucradas en la materialización de esta gran idea de nuestro líder Fidel. La creación de este gran centro de altos estudios (UCI).

A nuestro tutor por dedicar la mayor parte de su tiempo a atender nuestras dudas y malcriadeces.

A los profesores que de una forma u otra sirvieron de ejemplo e influyeron en nuestra formación.

A la dirección y el equipo de desarrollo de RN (Lourdes, Yosvany, Henrik), mil gracias por todo.

A los compañeros de grupo que siempre estuvieron en los momentos que hacían falta.

A todos los que nos han ayudado en cualquier circunstancia.

A aquellos que nos han puesto obstáculos y pensaron que este día nunca llegaría, también por ustedes estamos aquí.

Nosotros.

Dedicatoria:

Jajaja…esto es increíble, ahora resulta que está es la parte más difícil de toda la tesis. No es que no tenga a quien dedicársela, más bien creo que es que tengo muchas personas a las que les debo la formidable conformidad que siento en estos momentos. Por favor si existe alguna persona que se pueda sentir herido por no verse en estas líneas que me disculpe… recuerden que no tengo muy buena memoria y este es un momento difícil….Pero bueno hay quien prácticamente ha hecho más que yo por obtener este título. Aquí se incluyen mis abuelos, mis formidables abuelos que aún después de 24 años siguen pendientes de cada detalle de mi vida. Mi mama, mi amiga, mi…..no sé ni que decir. Tal vez una frase que ella siempre utilizo. “Vistes, juntos lo logramos”. Mi hermanito hectiquin: ojalá algún día pueda sentirse como me siento yo en estos momentos…jajaj. Quien lo diría; comencé a escribir riéndome y ahora estoy llorando como un niño. No importa, también le dedico esto a mi papa que creo no sea necesario decirlo pero ni el mismo sabe cuánto lo quiero. A mi hermanito Carlitín.” pipo estudia mucho que no hay mejor sensación que sentirse realizado”. Mis tíos Omáida y Roberto por el apoyo que siempre me han dado. A mis primos Lismey, Tonito, Deborah. A Jesús (El gato), Mari; a los dos: “muchas gracias por tratarme como un hijo”. A mis amigos, los viejos y los nuevos; Daybert, Asmel, Dayant, Omel, Yandry, Samuell, Shalimar, Carlito, el migue…jejje Julio (mi compañero de tesis): “que faena juli!!!!”. En fin a todos los que han estado conmigo en las buenas y las malas.

Esta persona que viene ahora no sé ni donde incluirla, Rudel (muy buen amigo!!!) pero muy buen profesor tu también tienes que estar aquí... “Gracias por todo mi hermano”. Libertad, no tienes la menor idea de cuánto te agradezco todo lo que hiciste por mí en el IPVCE: “Ojala algún día tener la posibilidad de retribuírtelo”. Henrik (mi tutor): “no te sientas mal por estar al final, solo me fui organizando cronológicamente para no olvidarme de nadie. Un Millón de gracias por todo y bueno; tú también formas parte de este equipo”. Yadi:”muchas gracias por tu apoyo y cariño, de verdad que me ha hecho mucho bien “.

Pero bueno, terminemos ya que si sigo no tengo para cuando parar y no me van a imprimir la tesis jajaja. Realmente no sé si he redactado unos agradecimientos o una dedicatoria pero bueno. Al final creo haber cumplido mi objetivo. Y como esta es la única sección en la que puedo poner lo que desee aprovecho para agradecer y dedicar este trabajo de 24 años (que soy yo) a todos ustedes. A todos un millón de gracias……

Ing. Johnny Pérez Acosta

Jajaja…que bien se ve la firma he?

Dedicatoria:

Por el inmenso amor, cariño y respeto que le tengo, dedico este trabajo de diploma a mi mamá linda…coño negra eres lo mejor para mí…nunca me has fallado, sé que te sientes orgullosa por lo que has logrado y hoy me satisface saber que sigo dándote alegrías…tremendos esfuerzos mami…pero al fin graduados!!!
También a mi abuelita del alma… que ha estado junto a mí…venciendo cada momento difícil…Mi vieja linda todavía le falta mucho por andar!!!

A mi guía, mi padre, mi amigo…Sauce… negro agradezco la confianza, educación y el cariño que me ha brindado, quiero que sepa que en cada logro obtenido existe una enseñanza suya.

A mi hermanito Carli…asere… para mí es un orgullo saber que también tú nos darás muchas alegrías… quizás aún no entiendas esto…pero deseo de ti…el respeto, el orgullo, la confianza que en cada momento exigieron de mí.

A mi primo Yurito…nos criamos y crecimos juntos…te agradezco cada momento en los que me has ayudado…mi hermanito espero algún día hacer lo mismo por ti…

Yami también tú eres parte de esto…te doy mil gracias por todo el cariño, el amor que siempre me has dado.

A mis primitos Javie, Dayi, la nueva primita super linda Mari Carla…también deseo mucho de ustedes…

A toda mi familia que tan felices se ponen cuando me ven…mi abuelo Alejo…mis tías…Merci, Tere, Eneida…mis tíos Luiso, Ali… a todos que con tanto esfuerzo han luchado en cada momento de la vida…esto también es de ustedes….

A mis viejos y nuevos amigos Ada, Ivanet, Ome, Yandrito, Samue……Johnito mi compañero de tesis y de pincha…asere la echastes buena!!!...ojalá se te den las cosas

A nuestro tutor Henrik…chama te agradecemos esa confianza y coraje que nos has demostrado…de corazón…tremendo ingeniero que eres…

A todos mis viejos y nuevos compañeros de grupo que también están felices por sus logros….

Para todos GRACIAS!!!

Tte.Cardoso…..ñoo

Resumen:

En el presente trabajo se elabora el modelo de diseño correspondiente a los procesos de negocio identificados para el módulo Control de Bienes en el subsistema Administración Financiera. Partiendo de la necesidad de culminar con buenos resultados el flujo de Análisis y Diseño, se realiza una explicación de los principales procesos de negocio que darán lugar al sistema a desarrollar, se describen las técnicas y herramientas disponibles, así como los artefactos que se generan producto de la actividad del diseñador de sistema dentro del equipo de desarrollo. La aplicación de patrones de diseño constituye un factor imprescindible para la obtención de un producto con calidad, motivo por el cual se lleva a cabo un estudio sobre algunos de los más conocidos. Además se realiza un análisis sobre la arquitectura base que soporta el sistema para poder comprender mejor su funcionamiento y lograr explotar al máximo sus potencialidades. Todo este conocimiento técnico se torna insuficiente si el equipo de trabajo no se encuentra perfectamente organizado y centrado en que tarea debe realizar en un momento indicado, por lo cual se sigue la metodología desarrollo de software Rational Unified Process (RUP). Por último y con el objetivo fundamental de evaluar la calidad del producto obtenido se realiza un estudio sobre técnicas de validación de diseño, a partir del cual se obtiene el conocimiento necesario para la selección y aplicación de algunas de ellas.

Palabras Claves

Diseño, Bienes, Inventario, Patrones, Arquitectura, Métricas de calidad, Modelo de diseño, Sistema, Módulo, Diagrama, Caso de Uso, Herencia, Agregación, Composición, Procesos, Negocio.

Abstract:

In this work it’s developed a design for business processes identified for Release Assets Control in the Financial Management subsystem. Based on the need to complete successfully the flow of Analysis and Design, is an explanation of key business processes that give rise to any system to develop. In this work will be described the techniques and tools available, as well as artifacts that are generated product designer of the activity within the development team. The application of design patterns is an essential factor for obtaining a quality product, why is carried out a study on some of the best known. Besides an analysis based on the architecture that supports the system in order to better understand its operations and exploit to achieve their maximum potential. All this technical knowledge becomes insufficient if the task force is not perfectly organized and focused on that task which must be performed at a certain time, thus follows the methodology software development Rational Unified Process (RUP). Finally, with the primary objective of evaluating the quality of the product, a study is carried out about the technical validation of design, from which they obtained the necessary knowledge for the selection and implementation of some of them.

ÍNDICE

Índice de Imágenes

Índice de Tablas

Introducción:

La práctica del inventario data de años inmemorables; desde las primeras civilizaciones debido a la necesidad de almacenar alimentos para períodos de escasez, se fue haciendo imprescindible mantener una relación de los recursos almacenados y así controlar la existencia exacta.

El objetivo principal de inventariar es distribuir los recursos necesarios para la entidad, colocándolos a su disposición en el momento indicado y así minimizar las pérdidas económicas por aumento de precio o extravió de los bienes inventariados. Permitiendo satisfacer las necesidades de la entidad a la cual debe estar constantemente adaptado.

La siguiente definición de inventario deja plasmada la importancia de la que es objeto este proceso dentro de cualquier entidad, motivo por el cual se busca sin descanso una vía para minimizar sus costos en esfuerzos y recursos, así como maximizar su fiabilidad y disponibilidad.

Inventario: …" Relación ordenada de bienes y existencias de una entidad o empresa, a una fecha determinada. Contablemente es una cuenta de activo circulante que representa el valor de las mercancías existentes en un almacén. En términos generales, es la relación o lista de los bienes materiales y derechos pertenecientes a una persona o comunidad, hecha con orden y claridad. En contabilidad, el inventario es una relación detallada de las existencias materiales comprendidas en el activo, la cual debe mostrar el número de unidades en existencia, la descripción de los artículos, los precios unitarios, el importe de cada renglón, las sumas parciales por grupos y clasificaciones y el total del inventario*."* (1)*.*

Problemática existente:

En el Ministerio del Poder Popular para las Relaciones Interiores y Justicia de Venezuela existe una dirección dedicada al control de los Registros y Notarías. El objetivo fundamental de estos registros es prestar servicios de procesos legales a la población venezolana. Para el correcto desempeño de estas funciones el estado asigna a cada uno de estos registros una serie de bienes, sobre los cuales es necesario mantener un control centralizado para evitar el desvío o malversación de los mismos. Actualmente una de las acciones que se está llevando a cabo en el marco del convenio Cubano Venezolano, es el proyecto de Servicios Autónomos de Registros y Notarías (SAREN), el cual está conformado por varios subsistemas, siendo uno de ellos Administración Financiera. Este subsistema garantizará la gestión de procesos funcionales en los Registros y Notarías existentes en el Ministerio del Poder Popular para las Relaciones Interiores y Justicia de Venezuela con el desarrollo de varios módulos entre los que se encuentra Control de Bienes; el cual culminó la fase de inicio y está centrado

en la fase de elaboración. Para el cumplimiento exitoso del flujo de trabajo Análisis y Diseño correspondiente a la fase antes mencionada, es imprescindible la obtención del Modelo de Diseño, que deberá proporcionar una imagen completa del software, enfrentándose a los dominios de comportamiento y de datos desde una perspectiva de implementación.

Formulación del Problema:

¿Cómo completar el proceso de Análisis y Diseño de la solución informática para el módulo Control de Bienes en el subsistema Administración Financiera del proyecto SAREN a partir de la identificación y caracterización de los procesos de Negocio?

Objeto de Estudio:

Proceso de Desarrollo de Software.

Campo de Acción:

El diseño de software en el módulo Control de Bienes perteneciente al subsistema Administración Financiera del proyecto SAREN

Hipótesis:

Si se obtiene el Modelo de Diseño de la solución informática para el módulo Control de Bienes en el subsistema Administración Financiera del proyecto SAREN, entonces se completará el proceso de Análisis y Diseño de la solución informática.

Objetivo General:

Diseñar una aplicación informática para el Control de los Bienes en los Registros y Notarías de la República Bolivariana de Venezuela a partir del análisis realizado por el equipo de analistas del módulo de Administración Financiera.

Partiendo de este objetivo se obtienen los siguientes **objetivos específicos**:

- Caracterizar el funcionamiento de los procesos de Inventario de Bienes de los Registros y Notarías de Venezuela.
- Aplicar los patrones de diseño definidos en la línea base de la arquitectura del sistema.
- Obtener el diseño de la solución informática para el Control de Bienes en los Registros y Notarías de la República Bolivariana de Venezuela.

- Evaluar a través de la aplicación de métricas la calidad del diseño obtenido.

Tareas:

- Realizar la búsqueda y clasificación de la bibliografía.
- Llevar a cabo un estudio sobre patrones de Diseño y Arquitectura que resulten factibles para su uso en sistemas distribuidos.
- Realizar un estudio de las metodologías de desarrollo de Software y seleccionar una que se utilizará para el diseño en cuestión, atendiendo a las características específicas del problema a resolver.
- Desarrollar un estudio sobre las herramientas y tecnologías actuales para el desarrollo de software.
- Explicar brevemente los principales procesos del negocio.
- Realizar el diseño de la aplicación para el Control de Bienes acorde con las necesidades de los Registros y Notarías de la República Bolivariana de Venezuela.
- Aplicar sobre el diseño propuesto Métricas para determinar la calidad del mismo.

Metodología de la Investigación

Para el desarrollo del presente trabajo de diploma fue necesario utilizar los métodos científicos Teórico y Empírico, dentro de los cuales se pueden identificar los siguientes:

Teóricos:

El método Analítico-Sintético fue utilizado para analizar la documentación obtenida de los flujos de trabajo anteriores y definir las características fundamentales que deben guiar el desarrollo del diseño.

El método Histórico-Lógico permitió analizar la trayectoria completa de los procesos de Inventario de Bienes desde su origen hasta la actualidad y revelar sus etapas principales. El método de la Modelación, posibilitó la creación de modelos que representan la solución de cada uno de los procesos identificados en el módulo Control de Bienes:

Empíricos:

La utilización del método de la entrevista fue necesaria para lograr un mayor entendimiento de los procesos de negocio identificados en el módulo Control de Bienes

Resultados Esperados:

Con la realización del presente trabajo se espera obtener los siguientes resultados:

- Modelo de diseño correspondiente al módulo Control de Bienes.
- Artefactos correspondientes al rol de diseñador de sistemas, establecidos por el proyecto siguiendo la metodología de desarrollo de software Rational Unified Process (RUP).
- Concluir el Flujo de trabajo de Análisis y Diseño para el desarrollo del módulo Control de Bienes.

Estructura del Documento:

El presente documento ha sido estructurado de la siguiente forma:

Capítulo 1: En este capítulo se realiza un estudio sobre los antecedentes y situación actual que muestran los sistemas ERP (Enterprise Resource Planning)[1] en cuba y el mundo, también se realiza una breve comparación del diseño orientado a objetos y el resto de los enfoques existentes. Además se estudian distintos patrones de diseño y arquitectura. La función del diseño en las metodologías de desarrollo de software más utilizadas así como las herramientas y tecnologías actuales para el desarrollo del software. Para finalizar con el contenido del capítulo se muestran técnicas que determinan la calidad de los diseños de software haciendo énfasis en las métricas orientadas a objetos.

Capítulo 2: En este capítulo reside la propuesta del diseño, se comienza con una caracterización de los procesos de negocio que tienen lugar en los Registros y Notarías de Venezuela y que están relacionados de una forma u otra con el control de los bienes. Luego se muestran las principales características del diseño del módulo Control de Bienes, haciendo mención a los principales patrones utilizados. Para entrar un poco más en materia en cuanto a la actividad del diseño se explican brevemente cada uno de los artefactos involucrados en esta actividad del proceso de desarrollo de software. Luego se procede con la realización de los casos de uso más relevantes, para culminar se muestra y explica la estructura del Módulo.

Capítulo 3: En este capítulo se aplican sobre el diseño obtenido en el capítulo anterior algunas métricas para la evaluación del diseño orientado a objetos, realizándose además un análisis de los resultados obtenidos.

[1] Sistemas de Planificación de Recursos Empresariales.

Capítulo 1: Fundamentación Teórica.

Introducción

En el presente capítulo se realiza una síntesis acerca de los sistemas ERP, su evolución a lo largo de la historia así como las principales empresas que se dedican a su creación o explotación. También se abordan los estilos arquitectónicos y patrones de diseño haciendo énfasis en los que se utilizarán en el desarrollo del módulo Control de Bienes. Se relacionan algunas de las metodologías de desarrollo de software, herramientas y tecnologías actuales con el diseño del software. Además de destacar la forma de evaluar dicho diseño con la aplicación de métricas y así determinar la calidad que poseerá el futuro producto de software.

1.1 Antecedentes y Estado del Arte.

El surgimiento de los sistemas ERP como herramientas estratégicas de trabajo se registra aproximadamente hace cuatro décadas atrás. La evolución de estos sistemas ha estado muy relacionada con el desarrollo que actualmente manifiestan las tecnologías de la información y las técnicas de gestión.

Los sistemas ERP que existen en la actualidad tuvieron sus antecedentes, los cuales a pesar de ser sistemas mejorables, fueron de gran utilidad en la gestión de inventarios. Se tiene conocimiento de sistemas como el EQQ (Orden Económica de Cantidades) utilizado un poco antes de la década del 60, el MRP (Planificación de Pedidos de Material) que ya surge en los años 60 y que aunque tuvo sus deficiencias, demostró ser una herramienta potente para la gestión de inventarios, el CL MRP (Planificación de Pedidos de Material de Ciclo Cerrado) que fue creado en la década del 70 y era un sistema semejante al MRP. En la década del 80 surge una versión del MRP conocida como MRP-II, la cual constituyó un efectivo método de gestión de todos los recursos de una empresa debido a que transformaba la planificación operacional en unidades y la planificación financiera en dinero.

A partir de este momento es que surge como tal el concepto de ERP asociado a términos como la planificación de negocios, la planificación de producción, tablas de tiempos de producción, la planificación de material y requisitos, planificación de capacidades y el funcionamiento del sistema para capacidades y prioridades. Pero el MRP-II sufrió también ciertos contratiempos producto de asumir elementos con carácter erróneo, como son los tiempos de producción fijos y las capacidades infinitas.

Alrededor de los años 90 y ya con todas las innovaciones tecnológicas de la época y su necesidad de expansión hacia otras áreas tan distinguidas como la Ingeniería, Finanzas, Recursos Humanos, Gestión de Proyectos, Servicios y Banca, surge el Enterprise Resource Planning (Software de Gestión Empresarial).

Los ERP evolucionaron y continúan evolucionando hasta la actualidad, pues intentan acompañar el desarrollo de las propias tecnologías computacionales. Todo esto provocó que las empresas comenzaran a ver a los ERP como sistemas capaces de hacerlas mejorar. Los sistemas de planificación empresarial, son sistemas estructurados que buscan satisfacer las demandas de soluciones de gestión empresarial, basados en el concepto de una solución completa que permita a las empresas unificar las diferentes áreas de productividad de la misma.

En la actualidad, la implantación de sistemas de gestión, sirven de soporte para la realización de una administración eficiente, brindando soluciones prácticas e integrales a problemas reales.

Los sistemas ERP se consideran herramientas potentes de trabajo a partir de que facilitan la existencia de un sistema de información integrado de todas las áreas funcionales de una empresa, ejecutan las tareas críticas de una empresa, aumentan la calidad de los servicios a los clientes mejorando la imagen de la empresa, posibilitan el cambio de información en ambientes distribuidos, posibilitan la integración de información de los distintos departamentos, oficinas, fábricas de una empresa así como de las varias empresas pertenecientes a un grupo financiero, resuelven muchos de los problemas comunes en una empresa: gestión de inventarios, servicios a clientes, gestión financiera, ayudan a la empresa a interrelacionar todos sus recursos, gestionándolos de la mejor forma posible en tiempo real, constituyen la mejor solución para una eficiente gestión de proyectos.

Muchos de los problemas que tienen las compañías con el ERP son debido a la inversión inadecuada para la educación continua del personal relevante, incluyendo los cambios de implementación y de prueba, y una falta de políticas corporativas que afectan como se obtienen los datos del ERP y como se mantienen actualizados.

Existen proveedores a nivel mundial que marcan la pauta en el mercado ERP, algunas de estas grandes empresas proveedoras de sistemas ERP a nivel mundial son:

- SAP: Fue fundada en 1972 en Alemania por cinco ingenieros de la IBM, siendo hoy la mayor empresa de su rama. Su sistema R/3 fue optimizado para gestionar los procesos de producción y gestión, logística y recursos humanos. Hoy día, pasados más de 30 años, cuenta ya con más

- de 12 millones de usuarios, 64.500 instalaciones, 1.500 socios y 23 soluciones informáticas. Es considerada la mayor empresa proveedora de ERP a nivel mundial, contribuyendo para eso, haber sido una de las pioneras.
- PeopleSoft: Es el segundo mayor proveedor mundial, siendo su arma más fuerte los módulos de gestión de recursos humanos. La compañía Peoplesoft está actualmente por direccionar sus productos para las áreas de los servicios, con productos de control de costos. SAP y la Peoplesoft han mantenido un éxito continuo debido a la oferta de nuevas potencialidades a sus clientes, así como el constante aumento de clientes que son empresas conocidas mundialmente.
- Oracle: Produce y vende aplicaciones ERP desde 1987, siendo la mayoría de sus clientes empresas relacionadas a la producción y consumo de productos, siendo así un adversario directo de SAP. Curiosamente en cerca de un 80% de los casos, el software de SAP opera sobre una base de datos Oracle.
- Baan: Es una empresa holandesa y una fuerte competidora de SAP. Recientemente, tal como otros proveedores, ha dedicado especial atención al mercado de pequeñas y medianas empresas, hecho que tiene resultado en una enorme variedad de productos que ofrece así como un rápido retorno financiero.
- JDEdwards: A pesar de vender software ya hace muchos años, sólo se hicieron conocidos mundialmente hace muy poco. Desde que lanzaron el OneWorld como software ERP, consiguieron una importante cuota dentro del mercado mundial de ERP.

Empresas venezolanas como Petróleos de Venezuela (PDVSA) utilizan soluciones brindadas por SAP, así como también PeopleSoft ha desarrollado soluciones para compañías de Telefonía Móvil (Movilnet).

En Cuba también existen compañías como la Empresa de Telecomunicaciones de Cuba, Sociedad Anónima (ETECSA) que usan soluciones implementadas por SAP y algunos efectivos sistemas cubanos de gestión Contable-Financiero como el Versat Sarasola utilizado por el Ministerio de la Azúcar (MINAZ), la Oficina Nacional de Administración Tributaria (ONAT), entre otras.

En los sistemas ERP, la gestión de Inventarios se convierte en una gran herramienta para la planificación de todos los bienes que una empresa posee, permitiendo llevar una gestión integral y mejorar la eficiencia en sus procesos. Brindan en todo momento información sobre la situación actual de los bienes, así como los diversos movimientos que han sufrido.

1.2 El Diseño en el Paradigma Orientado a Objetos POO

Con el reciente auge de la computación en el mundo entero ha aumentado de forma exponencial la demanda de software, evidenciando esto la necesidad de encontrar técnicas y tecnologías de la Ingeniería de Software que sean cada vez más eficientes. En esa búsqueda de la solución apareció el Paradigma de la Programación Orientada a Objetos, el mismo basa su filosofía en interacciones entre objetos, los cuales poseen características y funcionalidades propias que pueden ser utilizadas por otros objetos.

Dentro de los procedimientos a seguir en un proyecto de ingeniería de software se encuentra el Diseño de Software, el cual es definido por el máster en ciencias Mario Rossainz como la acción de construir soluciones que satisfagan los requerimientos del cliente (2) . Cualquier diseño debe ser modelado como una gráfica dirigida, compuesta por entidades con atributos que participan en relaciones.

A continuación se listan los criterios que sugiere Bertrand Meyer (3) que juzgan la capacidad que posee un método de diseño para lograr ciertos elementos importantes tales como la modularidad:

- Descomponibilidad.- Facilidad con la cual un método de diseño ayuda al diseñador a descomponer un gran problema en sub problemas más sencillos de resolver.
- Componibilidad.- Grado con el cual un método de diseño asegura que los componentes de un programa (módulos), una vez diseñados y construidos, pueden reusarse para crear otros sistemas.
- Comprensibilidad.- Facilidad de comprensión de un componente de programa sin referencia a otra información o módulos.
- Continuidad.- Facilidad de hacer pequeños cambios en un programa y hacer que estos se manifiesten por sí mismos en cambios correspondientes solamente en uno o unos pocos módulos más.
- Protección.- Característica arquitectónica que reducirá la propagación de efectos colaterales si ocurre un error en un módulo dado.

Estos criterios pueden ser aplicados a cualquier método de diseño, incluso al diseño estructurado, no obstante existe un método que cumple con todos estos criterios de manera más eficiente, dando como resultado una arquitectura que cumpla con todos los principios de modularidad de una manera más eficaz. En la POO el diseño es más complejo que en el resto de los enfoques, pues se persigue obtener un diseño mucho más abierto y genérico.

Características principales del Diseño Orientado a Objetos:

- Los objetos son abstracciones del mundo real o entidades del sistema que se administran entre ellas mismas
- Los objetos son independientes y encapsulan el estado y la representación de información
- La funcionalidad del sistema se expresa en términos de servicios de los objetos
- Las áreas de datos compartidas son eliminadas.
- Los objetos se comunican mediante paso de parámetros
- Los objetos pueden estar distribuidos y pueden ejecutarse en forma secuencial o en paralelo

Ventajas del Diseño Orientado a Objetos:

- Fácil de mantener, los objetos representan entidades auto-contenidas
- Los objetos son componentes reutilizables
- Para algunos sistemas, puede haber un mapeo obvio entre las entidades del mundo real y los objetos del sistema

1.3 Arquitectura y Diseño.

En la actualidad existe un amplio debate acerca de la relación que guardan los términos Arquitectura y Diseño dentro del proceso de desarrollo de software. A pesar que la comunidad de Arquitectura de Software (AS) expresa que esta difiere grandemente de la mera función del diseño, existen fuentes que hacen un análisis un poco más profundo y coinciden en conclusiones mucho más aceptadas. Según estas fuentes en alguna medida, la arquitectura y el diseño sirven al mismo propósito. Solo que se encuentran a diferentes niveles y por lo tanto poseen un grado de especificidad desigual.

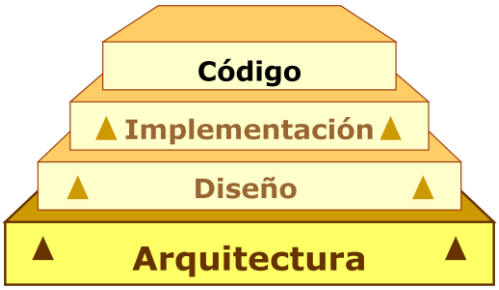


Figura 1 Relación entre Arquitectura y Diseño

Se expresa que a medida que la arquitectura se refina y se hace referencia a los elementos arquitectónicos de más bajo nivel se está realizando una transformación de la arquitectura en diseño. En su reciente libro sobre el arte de la AS, Stephen plantea que la AS es una metáfora relativamente

nueva en materia de diseño de software y que en realidad abarca también las metodologías de diseño, así como metodologías de análisis. Según Stephen el arquitecto de software es como una combinación de los roles analista y diseñador del sistema así como el ingeniero de software. Plantea además que el concepto de arquitectura intenta incluir las actividades de Análisis y Diseño en un framework de diseño más amplio y más coherente (4).

Como se evidencia en las dos concepciones lo que está bien claro es la gran importancia que recae sobre estas actividades definidas para el desarrollo de un software, motivo por el cual se hace imprescindible en este trabajo de diploma la realización de un breve resumen sobre algunos de los estilos arquitectónicos y patrones de diseño más utilizados.

1.3.1 La Arquitectura

La Arquitectura de Software (AS) es una disciplina reciente en el mundo del desarrollo de software pero se ha demostrado que representa un elemento de vital importancia dentro del ciclo de desarrollo del mismo. Como se expresa en la definición oficial de la AS según la IEEE:" La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución" (5).

Si una Arquitectura de Software presenta dificultades en su diseño o peor aún no se ha definido, existen grandes posibilidades de que el sistema que se está desarrollando no cumpla con la totalidad de los requerimientos definidos. Esto provocará la necesidad de un esfuerzo adicional con el objetivo de redefinir el sistema o en el peor de los casos provocará el fracaso del sistema cuando se encuentre en explotación.

Con el objetivo de no incurrir en estos fallos y construir los cimientos de un sistema robusto, seguro, reutilizable, buscando siempre un bajo acoplamiento y alta cohesión entre los componentes del mismo; se han definido una serie de patrones arquitectónicos que atendiendo a disímiles características del software a desarrollar podrían ser o no aplicables a un proyecto determinado.

A continuación se realiza un resumen de algunos de los estilos arquitectónicos más utilizados.

1.3.1.1 Modelo Vista Controlador (MVC)

Un objetivo muy generalizado en la mayoría de los sistemas no es más que mostrar al usuario la información almacenada y almacenar dicha información luego de ser modificada. Debido a que el flujo de información es entre la interfaz y el almacenamiento una de las primeras ideas o tentaciones es la

de unir ambas piezas para reducir el código y optimizar el rendimiento. Esta idea es totalmente errada, pues la interfaz suele depender de distintas clases de dispositivos como pueden ser los buscadores, además la programación de la interfaz de usuario requiere de técnicas y habilidades muy distintas a las requeridas en la programación de la lógica del negocio. Otra de las razones es que en la mayoría de los casos las aplicaciones incorporan lógica de negocio que va mucho más allá de la transmisión de datos.

El patrón conocido como Modelo Vista Controlador separa el modelado del dominio, la presentación y las acciones a realizar sobre la información ingresada por el usuario en tres clases diferentes:

- Modelo. El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).
- Vista. Maneja la visualización de la información.
- Controlador. Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

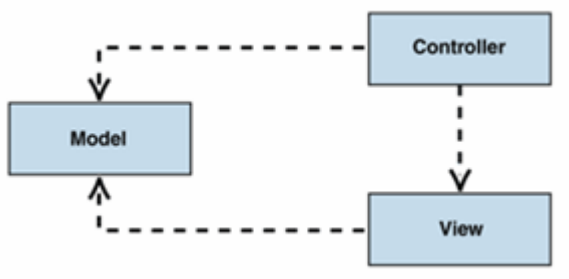


Figura 2. Modelo Vista Controlador

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo independientemente de la representación visual.

La principal ventaja que ofrece este patrón según (6) en su documentación de Patterns & Practices es el soporte de vistas múltiples. Dado que la vista se encuentra separada del modelo y no existe una dependencia directa entre estos; el usuario puede mostrar múltiples vistas de los mismos datos de forma simultánea. Como inconveniente se tiene que la complejidad de la solución aumenta ligeramente, producto a la introducción de nuevos niveles de indirección. Además que la orientación a eventos del código de la interfaz de usuario puede llegar a ser difícil de depurar.

1.3.1.2 Arquitectura en Capas (layering)

Otro estilo Arquitectónico de gran utilización y en ocasiones utilizado junto con el Modelo Vista Controlador es la arquitectura en capas. Esta es una de las técnicas más utilizadas por los diseñadores de software a la hora de concebir un sistema. Este estilo arquitectónico facilita el entendimiento y la construcción del software al crear una capa que utiliza los servicios brindados por una capa inferior y así sucesivamente. Un ejemplo típico de esto es la llamada Arquitectura en tres capas, en la cual la capa de ***presentación*** solo conoce los servicios brindados por la capa de ***negocios*** y esta a su vez conoce lo que sucede en la capa de ***datos,*** es importante aclarar que una capa inferior no tiene conocimiento de lo que ocurre en su capa superior, es decir; en este esquema la capa superior utiliza los servicios brindados por su capa inferior.

Este estilo facilita la comprensión de una capa de forma coherente sin tener conocimiento de cómo están definidas el resto de las capas, con esto permite sustituir por capas alternativas las implementaciones de los mismos servicios básicos sin tener que cambiar lo definido en las capas restantes. También se minimizan las dependencias entre las capas evitando así tener que realizar grandes cambios debido a una modificación realizada en la forma de cometer los servicios por parte de las capas inferiores. Y como una de las ventajas más valiosas brinda una gran posibilidad de reutilización al poder utilizar en una capa específica todos los servicios definidos en las capas inferiores.

La Arquitectura en capas posee un gran número de ventajas pero también tiene algunos inconvenientes, se ha comprobado que logra encapsular muchas funcionalidades del sistema, pero no todas. Esta cuestión provoca que en ocasiones se tenga que realizar cambios en cascada; ejemplo de esto es que si se adiciona un campo en una tabla de la base de datos el cual se desea mostrar en un formulario de la capa de presentación, se tendrá que ir realizando el cambio en la capa inferior, la que le sigue y así sucesivamente hasta llegar a la presentación. Un exceso de capas también podría ser fatal, pues perjudica el rendimiento debido a la necesidad de transformar la información una y otra vez al pasar de una capa a otra.

A la hora de establecer una arquitectura en capas es necesario comprender perfectamente que se desea, siendo una de las cosas más difíciles la definición de las capas a utilizar y la responsabilidad se le asignará a cada una. Un ejemplo de esto es la Figura 3 que se muestra a continuación.

Esta estructura posee tres capas horizontales. La capa de interfaz en la cual se gestionan las interfaces de usuario, una capa que incluye toda la lógica del negocio así como una capa para gestionar el acceso a los datos.

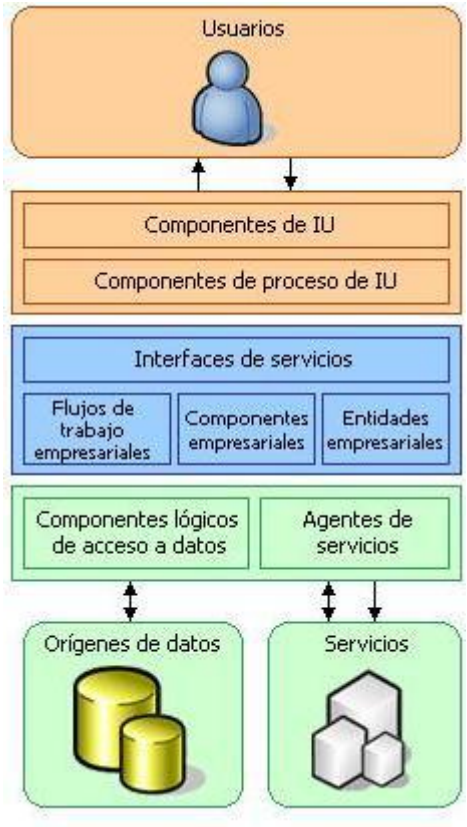


Figura 3 Diagrama de la Arquitectura en Capas

1.3.2 Patrones de Diseño de Software

Un patrón de diseño, obviamente, debe encajar por un lado con otros tipos de patrones imaginables, y por el otro con la teoría, la práctica y los marcos que en general rigen el diseño. Los patrones de diseño de software buscan codificar y hacer reutilizables un conjunto de principios a fin de diseñar aplicaciones de alta calidad. Se aplican en principio sólo en la fase de diseño, aunque en los últimos tiempos se ha comenzado a definir y aplicar patrones en las otras etapas del proceso de desarrollo, desde la concepción arquitectónica inicial hasta la implementación del código. La mayoría de los textos de la disciplina vinculan los estilos con los patrones de diseño según (7) los estilos se pueden comprender como grupos o lenguajes de patrones. Un estilo proporciona un lenguaje de diseño con un vocabulario y un framework a partir de los cuales los arquitectos pueden construir patrones de diseño para resolver problemas específicos. Los estilos se vinculan además con conjuntos de usos

idiomáticos o patrones de arquitectura que ofician como micro arquitecturas; a partir de las cuales es fácil derivar patrones de diseño.

A continuación se realiza un resumen en el que se muestran las situaciones en las que se recomienda la utilización de algunos de los patrones que se utilizarán en el diseño del módulo en cuestión, así como sus ventajas y desventajas más influyentes.

1.3.2.1 Fábrica Abstracta (Af).

El propósito de este patrón es crear una interfaz para la creación de familias de objetos u objetos independientes, sin tener la necesidad de especificar sus clases correspondientes. Este patrón puede ser utilizado por diferentes motivos, entre ellos se pueden encontrar los siguientes:

- Cuando se necesita que el sistema sea independiente de la creación, composición y representación de sus objetos.
- Cuando el sistema puede ser configurado con una familia de productos dentro de múltiples disponibles.
- Cuando se quiere proporcionar una librería de clases de objetos de la que se desea revelar solo sus interfaces y no la implementación.

Af potencia el encapsulamiento aislando a los clientes de las implementaciones, aumenta la flexibilidad del diseño permitiendo sustituir fácilmente una familia de objetos. Y refuerza la consistencia al obligar a la utilización de una familia de objetos a la vez.

Independientemente de que este patrón dificulta la extensibilidad al tornar en una tarea engorrosa la agregación de una nueva familia de objetos constituye uno de los patrones Orientados a Objetos más utilizados desde su creación.

1.3.2.2 Fachada

Tiene como propósito brindar una interfaz que engloba las funcionalidades de todo un sistema, la cual; no oculta las funcionalidades de menor nivel para que puedan ser utilizadas por los usuarios que lo requieran, y facilita el trabajo a la mayoría de los clientes. Es recomendable utilizar este patrón en los casos en que existe mucha dependencia entre los usuarios y las clases que implementan una abstracción para brindar independencia y portabilidad al sistema. También se utiliza en los sistemas estructurados en capas para englobar las funcionalidades de cada una de las capas, además de que puede ser utilizado para proporcionar una interfaz simple a un sistema complejo. A continuación se muestran las ventajas principales que proporciona la utilización de este patrón.

- Se facilita el uso del subsistema pues al separar al cliente de los componentes del subsistema se reduce la cantidad de objetos con los que el cliente interactúa dentro del mismo.
- Se eliminan las dependencias entre el subsistema y los clientes, logrando un bajo acoplamiento.
- No existen obstáculos para que las aplicaciones usen las clases del subsistema que necesiten. De esta forma se puede elegir entre facilidad de uso y generalidad.

La Fachada sabe que clase es responsable de cada petición que recibe, pero esto es invisible al cliente, siendo la propia fachada la encargada de recibir la petición y de traducir el resultado de la misma.

1.3.2.3 Singleton

El objetivo de este patrón es garantizar que una clase sólo tenga una única instancia, proporcionando un punto de acceso global a la misma. Este patrón se utiliza cuando debe haber únicamente una instancia de una clase y debe ser claro su acceso para los clientes. Aunque también es utilizado en ocasiones en que la "Instancia Única" debe ser especializada mediante herencia y los clientes deben poder usar la instancia extendida sin modificar su código. Como es de esperar esto aporta muchas ventajas como son:

- El acceso a la "Instancia Única" está más controlado.
- Se reduce el espacio de nombres (frente al uso de variables globales).
- Permite refinamientos en las operaciones y en la representación, mediante la especialización por herencia de "Solitario".
- Es fácilmente modificable para permitir más de una instancia y, en general, para controlar el número de las mismas (incluso si es variable).

El método "Instancia" es el punto de acceso a la "Instancia Única" del "Solitario" para los clientes. Lo que ocurre en realidad con este método es que, si "Instancia Única" no ha sido aún creada, la crea (inicialización perezosa) llamando al constructor "Solitario", el cual solo puede ser accedido por el mismo Singleton debido a su nivel de visibilidad privado.

1.3.3 Frameworks

Una tendencia bastante reciente en el mundo de la programación es la utilización de frameworks como base para el desarrollo de las aplicaciones, en ellos se definen reglas y funcionalidades que serán

utilizadas con gran frecuencia dentro de la aplicación. A continuación se muestran algunas de las ventajas que provee la utilización de framework.

- El programador no necesita plantearse una estructura global de la aplicación, sino que el framework le proporciona un esqueleto que hay que “rellenar”.
- Facilita el desarrollo, como estandariza muchas cosas dentro del sistema se hace más ágil el desarrollo y la comunicación entre los miembros del equipo de trabajo
- Es más fácil encontrar herramientas (utilidades, librerías) adaptadas al framework concreto para facilitar el desarrollo.

1.4 El Diseño y las Metodologías de Desarrollo de Software (MDS).

Con el objetivo de facilitar la comprensión del presente epígrafe, se hace una compilación de las definiciones brindadas por (1) del término método, llegando a los siguientes resultados:

- Proceso o camino sistemático establecido para realizar una tarea o trabajo con el fin de alcanzar un objetivo predeterminado.
- Modo de decir o hacer algo en orden.
- Procedimiento científico seguido en la ciencia para hallar la verdad.

De forma análoga se puede llegar a una definición para metodología de desarrollo de software:

- Proceso específico que define quién, cómo y cuándo debe realizar un conjunto de actividades necesarias con el fin de lograr un producto de software.

En la actualidad existen una gran variedad de metodologías de desarrollo de software las cuales priorizan distintos objetivos, por lo cual son utilizadas generalmente para desarrollar productos de software con características particulares. A continuación se realiza una pequeña descripción de tres de las metodologías más utilizadas en la actualidad, así como la relevancia del diseño en cada una de ellas.

1.4.1 Rational Unified Process (RUP)

La metodología RUP, llamada así por sus siglas en inglés Rational Unified Process, divide en 4 fases el desarrollo del software:

- **Inicio**, El objetivo en esta etapa es determinar la visión del proyecto.
- **Elaboración**, En esta etapa el objetivo es determinar la arquitectura óptima.

- **Construcción**, En esta etapa el objetivo es llegar a obtener la capacidad operacional inicial.
- **Transición**, El objetivo es llegar a obtener la solución del proyecto.

Cada una de estas fases se desarrolla de forma iterativa, característica que no es más que reproducir el ciclo de vida en cascada a menor escala. Los Objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes.

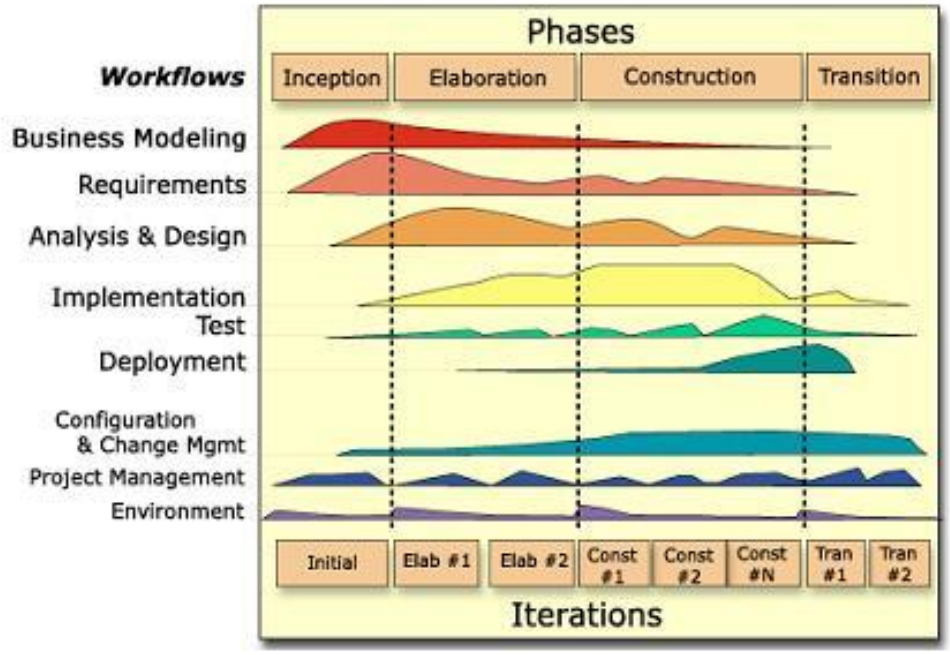


Figura 4 Fases e Iteraciones de la Metodología RUP

El Lenguaje Unificado de Modelado (UML) es una parte esencial de RUP. Este es utilizado para modelar todos los esquemas de un sistema de software. Durante el flujo de trabajo de Diseño se modela el sistema de forma tal que responda a los requerimientos funcionales y no funcionales que se le suponen. Según (8) los propósitos específicos del diseño son los siguientes:

- Adquirir una compresión en profundidad de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnología de destrucción y concurrencia, tecnología de interfaz.
- Crear una entrada apropiada y un punto de partida para actividades de implementación subsiguiente, capturando los requisitos y subsistemas individuales, interfaces y clases.

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software. Entre los artefactos generados durante el flujo de trabajo de diseño se encuentran los siguientes:

- Modelo de diseño.
    - Clases del diseño.
    - Realización de caso de uso del diseño.
- Subsistema de diseño.
- Interfaz.
- Descripción de la arquitectura.
- Modelo de despliegue.

Siendo el modelo del diseño el de mayor importancia por su utilización en el flujo de trabajo de implementación.

1.4.2 Programación Extrema (XP)

Esta metodología es una de las más exitosas en la actualidad. Es utilizada para proyectos pequeños en los cuales el plazo de entrega es corto y el equipo de desarrollo cuenta con poco personal. Esta metodología consiste en integrar al cliente o usuario final al equipo de trabajo, propiciando así una programación rápida o extrema.

Esta metodología posee varias características con el fin de lograr el éxito final del proyecto:

- **Pruebas Unitarias:** se basa en realizarle pruebas a los principales procesos, de tal manera que se puedan predecir los posibles problemas del futuro.
- **Refabricación:** se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- **Programación en pares:** una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo.

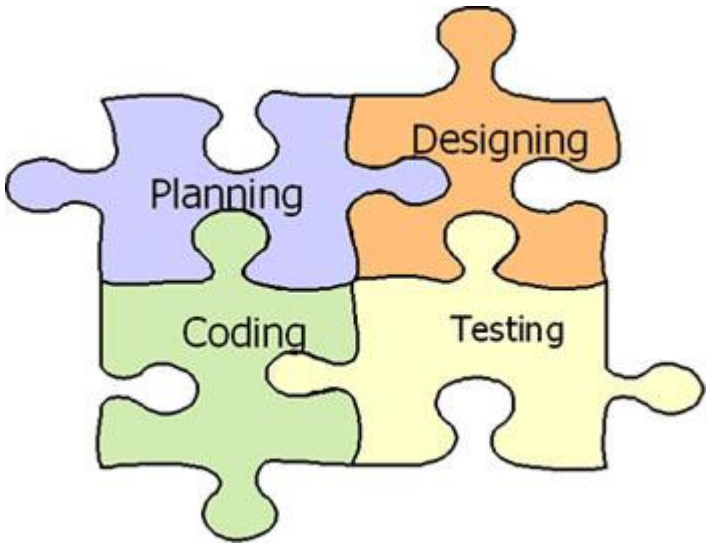


Figura 5 Metodología Programación Extrema (Extreme Programing)

El Diseño en esta metodología debe ser muy simple. Para lograr esto se establece que siempre se realicen las tareas de la forma más simple posible y que no se incluyan en el diseño características funcionales que no fueron analizadas durante la planificación. En la Programación Extrema un buen diseño es imprescindible para el éxito.

1.4.3 Microsoft Solution Framework (MSF)

Esta es una metodología flexible que controla la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando a un segundo plano las elecciones tecnológicas

Figura 6 Metodología MSF

Entre las principales características de MSF se encuentran las siguientes:

- **Adaptable:** es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- **Escalable:** puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas o más.
- **Flexible:** es utilizada en el ambiente de desarrollo de cualquier cliente.
- **Tecnología Agnóstica:** porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y finalmente el modelo de Aplicación. Aconseja a los equipos que esperen cambios originados por los participantes e incluso por el propio equipo. Por ejemplo, admite que los requerimientos de un proyecto pueden ser difíciles de articular de antemano y que a menudo sufren modificaciones significativas, a medida que los participantes van viendo sus posibilidades con mayor claridad. MSF ha diseñado tanto su Modelo de Equipo como su Modelo de Proceso para anticiparse al cambio y controlarlo.

Los principios de MSF evidencian su integración con las metodologías ágiles:

- Alentar comunicaciones abiertas.
- Trabajar hacia una visión compartida.
- Otorgar poder a los miembros del equipo.
- Establecer responsabilidad clara y compartida.
- Concentrarse en la entrega de valor de negocios.
- Permanecer ágil, esperar el cambio.
- Invertir en calidad.
- Aprender de todas las experiencias.

Con esta metodología el equipo obtiene mediante el diseño lógico los modelos para objetos, servicios, atributos y relaciones de la solución. Generalmente el equipo utiliza los artefactos que mejor representen las partes más complejas del proyecto. Aquí se encuentran los siguientes:

- Inventarios de Objetos y Servicios.
- Diagramas de Clases.

- Diagramas de Secuencia.
- Diagramas de Actividades.
- Diagramas de Componentes.

Durante el diseño físico, el equipo refina estos modelos generados en el diseño lógico del sistema.

1.5 Herramientas y tecnologías actuales.

En el presente epígrafe se aborda el tema referente a herramientas y tecnologías, que en la actualidad constituyen la base principal para el eficiente diseño de un Software. Por tal motivo se realiza una breve descripción de las principales características y beneficios que aporta cada una de ellas con su aplicación.

1.5.1 Rational Rose.

Rational Rose se considera una de las herramientas CASE para modelado más potente existente en el mercado mundial. Esto se basa principalmente en el nivel de integración que tiene esta con el resto de las herramientas que lo acompañan en la suite, donde aparece el Rational Clear CASE para el control de versiones, el Rational Clear Quest para el control de cambios, el Rational Model Integrator, entre otros. La posibilidad de generar y realizar ingeniería inversa en una buena cantidad de lenguajes de programación es otro aspecto a destacar así como también el número de framework que vienen predefinidos, entre los cuales se pueden citar j2ee, .net, Visual Basic 6, C++ entre otros.

1.5.2 Enterprise Architect.

Enterprise Architect 7.0 es una herramienta de construcción y modelado de software de alto rendimiento basado en el estándar de UML 2.1, diseñada para ayudar a construir software robusto y fácil de mantener. Brinda una solución de modelado verdaderamente ágil, fácil de usar, rápido, flexible y con una interfaz gráfica amigable. Soporta generación e ingeniería inversa de código fuente para muchos lenguajes populares, incluyendo C++, C#, Java, Delphi, VB.Net, Visual Basic y PHP. Es una herramienta multi-usuario, con seguridad y administración de permisos incorporada. Soporta diferentes repositorios basados en DBMS (Sistemas Manejadores de BD), incluyendo Oracle, SQL Server, My SQL, PostgreSQL. Brinda soporte para control de versiones y posee bajos costos de licencias.

1.5.3 Plataforma J2EE.

J2EE (Java 2 Enterprise Edition) plataforma que define como estándar un grupo de especificaciones, que deben ser seguidas para desarrollar el producto, para ello es necesario adquirir una serie de recursos como el JRE (Java Runtime Environment), JDK (librerías), servidores web, entornos de

programación, entre otros, que en su conjunto permitirán desarrollar las aplicaciones. Soporta un único lenguaje que es Java, utilizado para el desarrollo de todos los componentes y compilado por un código intermedio denominado ByteCodes que se ejecuta en un entorno de ejecución llamado JRE para transformar el lenguaje intermedio a código propio de la máquina en la que se corre la aplicación. Brinda soporte para múltiples sistemas operativos, debido a la portabilidad, o posibilidad de ejecutar las aplicaciones desarrolladas en cualquier sistema operativo y/o máquina del mercado.

Existen sólo dos formas oficiales para acceder a la plataforma J2EE con otros lenguajes, la primera es a través de JNI (Java Native Interface) y la segunda es a través de la interoperabilidad que ofrece CORBA. Utiliza múltiples productos lanzados al mercado que ofrecen entornos de desarrollo adecuados, tales como NetBeans de Sun, Visual Café de WebGain, Eclipse, entre otros.

Muchas empresas crean soluciones basadas en J2EE que ofrecen características tales como rendimiento y precio muy diferentes. De este modo, se ha desarrollado a un nivel exponencial la plataforma y los clientes tienen la posibilidad de escoger entre una gran cantidad de opciones.

1.5.4 Plataforma .NET.

.NET es una plataforma de desarrollo de Software que brinda la posibilidad de conectar una gran variedad de tecnologías de uso personal y de negocios. Teniendo como base, estándares de Servicios Web XML. Esta plataforma también permite, independientemente del sistema operativo, tipo de computadora o del lenguaje de programación empleado para crearlo, que las aplicaciones sin importar su origen, conecten sus datos y transacciones.

.NET está presente en toda la línea de productos Microsoft, ofreciendo la capacidad de desarrollar, implementar, administrar y utilizar soluciones conectadas a través de servicios, de manera rápida, económica y segura. Simplificar el desarrollo de aplicaciones es uno de los objetivos que posee la plataforma .NET, así como proveer las herramientas y tecnologías para transformar la red en una plataforma de computación distribuida a gran escala.

Con una plataforma de software definida, resulta importante tener conocimiento de cómo desarrollar aplicaciones de la forma más sencilla posible sobre dicha plataforma y esta responsabilidad la posee .NET Framework que constituye un marco de trabajo, y este denota la infraestructura sobre la cual se reúnen un conjunto de lenguajes, herramientas y servicios que simplifican el desarrollo de aplicaciones en entorno de ejecución distribuido. Este marco de trabajo soporta múltiples lenguajes de programación y aunque cada lenguaje tiene sus características propias, es posible desarrollar cualquier tipo de aplicación con cualquiera de estos lenguajes. Existen más de 30 lenguajes adaptados

a .Net, desde los más conocidos como C# (C Sharp), Visual Basic o C++ hasta otros lenguajes menos conocidos como Perl o Cobol.

Los principales componentes de este entorno son:

- El conjunto de lenguajes de compilación.
- El Entorno Común de Ejecución para Lenguajes o CLR por sus siglas en inglés.
- La Biblioteca de Clases Base o BCL

1.5.5 Framework Spring.

Spring es un conjunto de librerías o módulos, donde se escogen aquellos que faciliten la implementación de las aplicaciones. Es un framework libre y de código abierto. Entre sus posibilidades más potentes está su contenedor de Inversión del Control. Es uno de los proyectos más sorprendentes en el panorama actual de Java, por el grado en que ayuda a que los diferentes componentes que forman una aplicación trabajen entre sí, pero no establece apenas dependencias consigo mismo, siendo esta su primera característica. Sería posible retirarlo sin prácticamente cambiar líneas de código. Lo único que sería necesario es, lógicamente, añadir la funcionalidad que provee, ya sea con otro framework principal o con el código del propio desarrollador. A nivel de soporte de la comunidad, Spring es uno de los proyectos con mayor actividad, con desarrollos dentro y fuera del framework.

1.5.6 Framework Nhibernate.

NHibernate es un framework de Object-Relational-Mapping (ORM) open-source que resuelve en forma automática la persistencia de los objetos de dominio .NET, está basado en el framework Hibernate surgido en la comunidad Java en el año 2002. NHibernate es la conversión de Hibernate de lenguaje Java a C# para su integración en la plataforma .NET. Al igual que muchas otras herramientas libres para esta plataforma, también funciona en Mono. Permite el acceso a datos asegurándole al desarrollador de que su aplicación es excelente en cuanto al motor de base de datos a utilizar en producción, pues soporta los gestores habituales en el mercado: MySQL, PostgreSQL, Oracle, MS SQL Server, etc. Sólo se necesita cambiar una línea en el fichero de configuración para que se pueda utilizar una base de datos distinta.

1.6 Métricas para el Diseño.

El objetivo fundamental de la ingeniería de software es la obtención de un producto de calidad. Para lograr esto es necesaria la aplicación por parte del equipo de desarrollo de un conjunto de técnicas y herramientas tanto para el desarrollo como para la validación del producto obtenido. Entre estas

técnicas se evidencian las métricas para la evaluación del software. Estas métricas pueden ser aplicadas en cada una de las etapas del ciclo de vida existiendo variedades para el Modelo de Análisis, el Modelo de Diseño o el Código propiamente dicho, así como para evaluar la calidad de la Pruebas realizadas sobre el producto.

Un inconveniente de las métricas es que no proporcionan información por sí solas y a veces en vez de claridad aportan confusión. Esto se debe a que muchas métricas no guardan relación con los intereses de las partes, y el indicador de la calidad de un esquema se construye generalmente con todas ellas.

Para ilustrar lo antes mencionado se plantea la siguiente situación:

La propiedad de corrección constituiría una característica deseable para el diseñador, en tanto que la comprensibilidad lo sería para el "requirente" (cliente, usuario, todo participante que formula requerimientos) y la facilidad de implementación para el programador y a todos les interesaría que el diagrama sea legible. La presencia de criterios de calidad que para un participante del proceso resultan irrelevantes "oscurece" un indicador de la calidad para su punto de vista.

Por este mismo motivo a continuación se refieren los indicadores de calidad relacionados con el rol de diseñador.

Arquitecto/ Diseñador	Agregación, completitud, corrección sintáctica, economía, estética, expresividad, extensibilidad, factorización, ¨legibilidad ¨

Tabla 1 Criterios de Calidad para el diseñador

En este epígrafe se exhibe un compendio de aquellas métricas que son aplicadas sobre el Modelo de Diseño o en otras palabras miden la calidad de los criterios asociados al Rol de Diseñador.

Los objetivos de las métricas de diseño son:

- Llevar un control de la calidad del producto que se está desarrollando
- Estimar el impacto que ese producto tendrá en las fases posteriores del proceso de desarrollo

Hay muchas métricas disponibles en el modelo de diseño, por lo que se estructuran de la siguiente manera:

- Métricas de diseño de alto nivel
- Métricas de diseño de componentes
- Métricas de diseño de interfaz
- Métricas de diseño OO

1.6.1 Métricas de diseño de alto nivel

Las métricas de diseño de alto nivel se concentran en las características de la arquitectura del programa, dando mayor importancia a la estructura arquitectónica y a la eficiencia de los subsistemas. Estas métricas son de caja negra y por lo tanto no se toca el código bajo ningún concepto.

Algunos expertos definen tres medidas de la complejidad del diseño del software: **complejidad estructural, complejidad de datos y complejidad del sistema.**

***La complejidad estructural*,** $S(i)$ de un módulo i se define de la siguiente manera: $S(i) = f_{out}^2(i)$ donde $f_{out}^2(i)$ es la expansión del módulo i.

La complejidad de datos, $D(i)$ proporciona una indicación de la complejidad de la interfaz interna de un módulo i y se define como: $D(i) = \frac{V(i)}{|f_{out}(i)+1|}$ donde $V(i)$ es el número de variables de entrada y de salida del módulo i.

La complejidad del sistema, $C(i)$ se define como las sumas de las complejidades estructural y de datos, se define como: $C(i) = S(i) + D(i)$

Según aumentan los valores de complejidad aumentará la complejidad arquitectónica del sistema, lo que propiciará un aumento en el esfuerzo necesario para la integración y las pruebas.

1.6.2 Métricas de diseño de componentes

Estas métricas están orientadas a medir el acoplamiento, complejidad y la cohesión del subsistema. Se denominan técnicas de caja blanca porque se requiere del conocimiento del funcionamiento interno del módulo. Se realizan luego de haber realizado un diseño procedimental aunque se pueden retrasar hasta tener el código fuente.

1.6.3 Métricas de diseño Orientado a Objeto

Estas métricas son de gran importancia para evaluar la calidad del diseño de un producto de software. Están orientadas a una clase en específica, a la jerarquía de clases así como a las colaboraciones que puedan existir entre un conjunto de ellas.

Cualquier característica de las clases puede ser utilizada como base para la medición, ya sean las operaciones y atributos encapsulados por las clases o las relaciones de herencia o colaboración que puedan existir entre las mismas.

Algunas de las métricas que se han definido a nivel de clases por algunos expertos son:

1.6.3.1 La familia de métricas CK (propuestas por Chidamber y Kemerer)

- **Métodos ponderados por clase (MPC).**

Se normalizan las complejidades (ciclomática o cualquier otra elegida) de todos los métodos pertenecientes a una clase determinada de manera que la complejidad nominal para un método tome valor de 10 y luego se realiza una sumatoria de estas complejidades:

$c_1, c_2, \ldots . c_n$ Pertenecientes a la clase C

$\mathrm{MPC} = \sum c_i$ Para cada i = 1 hasta n

El número de métodos y su complejidad constituyen indicadores suficientes para determinar el esfuerzo necesario para implementar y verificar una clase. Además según aumenta el número de métodos de una clase aumenta el árbol de herencia, pues todas las subclases heredan los métodos de sus padres y mientras más métodos pertenezcan a una clase más posibilidad existe que se tornen más específicos de la aplicación, disminuyendo así la posibilidad de reutilización.

- **Árbol de profundidad de herencia (APH)**

Esta métrica se define como la longitud máxima de los nodos a la raíz del árbol. Mientras mayor sea el APH las clases de los más bajos niveles heredarán muchos métodos. Esto dificulta la tarea de predecir el comportamiento de una clase además de introducir una dificultad mayor en el diseño.

- **Número de descendiente (NDD)**

Según esta métrica el aumento de los descendientes[2] incrementa la reutilización pero introduce una posible disolución de la abstracción representada por la clase predecesora. El hecho de tener muchas clases hijas aumenta la posibilidad de la existencia de un miembro no apropiado. Además de incrementar las pruebas necesarias para ejercitar cada descendiente en su contexto operativo.

- **Acoplamiento entre clases objeto (ACO)**

ACO es el número de colaboraciones definidas para una clase. A medida que estas aumentan es más probable que el grado de reutilización de una clase decrezca. Además mientras mayor sea el ACO

[2] descendiente según Pressman: Las subclases inmediatamente subordinadas a una clase en la jerarquía

más complicadas se tornarán las modificaciones y las pruebas a realizar sobre las mismas. Por esto es aconsejable que los valores de ACO se mantengan lo más bajo posible para cada una de las clases.

- **Respuesta para una clase (RPC)**

El RPC se define como el conjunto de métodos en el conjunto de respuesta [3]de una clase. A medida que este se incrementa, el esfuerzo requerido para la comprobación también aumenta, incrementándose también la secuencia de comprobación. Además con el aumento de la RPC, se eleva la complejidad del diseño global de la clase.

- **Carencia de cohesión en los métodos (CCM)**

El CCM no es más que la cantidad de métodos que acceden a un mismo atributo dentro de la clase. Cuando el CCM es alto los métodos deben acoplarse a otro por medio de los atributos lo que incrementa la complejidad del diseño de clases. En general, los valores de CCM altos implican que la clase debe ser rediseñada descomponiéndola en dos o más clases distintas.

1.6.3.2 La serie de métricas LK (propuestas por Lorenz y Kidd)

- **Tamaño de Clase (TC).**

Consiste en medir el tamaño general de una clase tomando el valor de la cantidad de operaciones y el número de atributos que están encapsulados dentro de dicha clase. Si el resultado obtenido indica valores grandes, significa que la clase posee un alto grado de responsabilidad, debido a esto se reducirá la reutilización, se hará mucho más difícil la implementación y la realización de pruebas de dicha clase. Por tanto mientras menor sea el valor para el TC se hará mucho más fácil la reutilización de dicha clase dentro del sistema.

- **Número de Operaciones redefinidas por una subclase (NOR).**

Se basa en la suma de las operaciones heredas que se redefinen, donde los valores grandes obtenidos indican generalmente problemas con el diseño debido a la dificultad que va existir al modificar y probar las clases heredas en dicho diseño.

[3] conjunto de respuesta según Pressman: Una serie de métodos que pueden potencialmente ser ejecutados, en respuesta a un mensaje recibido por un objeto en la clase.

- **Índice de Especialización (IES).**

Proporciona una indicación aproximada del grado de especialización, para cada una de las clases hijas dentro de determinado diseño. Es importante destacar que la especialización se puede alcanzar redefiniendo, agregando o eliminando operaciones.

A continuación se muestra la manera en que se obtiene el IES:

$$\mathrm{IES} = [\mathrm{NOR} \times \mathrm{nivel}] / \mathrm{Mtotal}$$

Donde $nivel$ corresponde al nivel en la jerarquía de clases donde se encuentra y $Mtotal$ es el número total de métodos que posee la clase.

Mientras más elevado sea el valor de *IES*, más probable será que la jerarquía contengan clases que no se ajusten a la abstracción de las clases heredadas.

Conclusiones del Capítulo:

Partiendo de lo mencionado durante el presente capítulo se define la siguiente propuesta de solución técnica, la cual regirá el desarrollo de este trabajo de diploma en sus fases posteriores.

Se sigue la metodología de desarrollo de software **Rational Unified Process (RUP)** debido a que independientemente de ser una metodología tradicional puede hacerse tan ágil como se desee. Esta metodología permite evaluar constantemente la calidad del producto así como tener una idea bastante cercana sobre el estado actual del proyecto al poder contar con los artefactos generados en cada una de las iteraciones. Por último pero no por eso menos importante utiliza el UML como lenguaje de modelado. Coincidiendo con esta característica se encuentra la herramienta case seleccionada el **Enterprise Architect 7.0**. Esta herramienta está diseñada para ayudar a construir software robusto y fácil de mantener basada fundamentalmente en la utilización de UML 2.1 como lenguaje de modelado. Es una herramienta multiusuario con seguridad y administración de permisos incorporada. Fácilmente integrable con el Subversion y el VSS. Permite generación de código fuente e ingeniería inversa para el lenguaje a utilizar en las fases posteriores del ciclo de desarrollo.

Entre las plataformas se selecciona **Microsoft.NET,** por las facilidades que brinda para la comunicación entre las aplicaciones. Además con el framework.NET se hace mucho más sencillo el desarrollo de aplicaciones al encontrarse integrado en él un conjunto de lenguajes, herramientas y servicios que facilitan en gran medida el trabajo. Perfectamente compatible con la plataforma seleccionada, **Microsoft Visual Studio**; es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) pensado para hacer rápida y fácilmente las aplicaciones de la nueva generación. Este IDE puede utilizarse para construir aplicaciones dirigidas a **Windows** (utilizando Windows Forms), **Web** (usando ASP.NET y Servicios Web) y **dispositivos portátiles** (utilizando .NET Compact Framework). El estilo arquitectónico seleccionado es en capas, el cual se complementa con la aplicación de los patrones de diseño Singleton, Fábrica Abstracta y Fachada logrando de esta forma una arquitectura sólida y segura.

Capítulo 2: Propuesta del Diseño

Introducción

El presente capítulo comienza con una descripción de los procesos de negocio con el objetivo de acercar al lector a la realidad funcional del negocio y de esta manera propiciarle un conocimiento que sirva de base para la comprensión del resto de los epígrafes. Luego se realiza una descripción de la arquitectura definida para el subsistema, haciendo énfasis en los principales patrones de diseño y arquitectónicos que se adoptan. Y para culminar se presentan todos los artefactos generados que dan solución a uno de los principales casos de uso identificado en el módulo Control de Bienes.

2.1 El Negocio.

Como resultado de un levantamiento minucioso realizado en las oficinas pertenecientes al Ministerio del Poder Popular para las Relaciones Interiores y Justicia de Venezuela se ha obtenido la descripción de los principales procesos que se tendrán en cuenta a la hora de desarrollar el diseño para el módulo de "***Control de Bienes***" del proyecto ***Modernización de los Registros y Notarías de la República Bolivariana de Venezuela***. Estos procesos se explican a continuación para facilitar el entendimiento del resto de los contenidos expuestos en el presente trabajo de diploma. Es importante tener presente que ***Control de Bienes*** es solo un módulo dentro de un gran sistema que automatiza e integra todos los procesos legales de estos Registros y Notarías. Por lo que cada uno de estos procesos tiene efecto sobre el resto de los módulos del sistema.

De manera general el módulo se encarga de la automatización de algunos procesos que puedan incidir de una forma u otra sobre los bienes que forman parte del patrimonio de SAREN. Estos bienes se pueden encontrar en dos estados fundamentales, en uso o en almacén; y dependiendo de este estado serán los procesos que afectan a dicho bien o suministro. Los Bienes en Almacén son registrados una vez que se realiza la carga inicial y luego son afectados por una serie de movimientos clasificados como incorporaciones o desincorporaciones. Toda la información generada por motivo de la ejecución de alguno de estos movimientos sobre un bien específico es almacenada y procesada con vistas a mantener disponible en todo momento el historial y existencia actual de dicho bien. Los bienes en uso ya son sometidos a procesos un poco más complejos, tal es el caso de la reasignación de bienes entre las Unidades Ejecutoras o la depreciación de los bienes. Este último comienza justo cuando el bien alcanza el estado de bien en uso y no es más que la resta de un valor que se encuentra en correspondencia con la explotación del bien durante un período de tiempo determinado en la cuenta patrimonial que respalda al bien. El equivalente a este monto se añade a una cuenta con el objetivo de

tener el respaldo monetario para la reposición del bien una vez que haya expirado su vida útil o se haya depreciado totalmente. A continuación se describen de manera más detallada estos procesos con vistas a propiciar la comprensión de los mismos y así lograr un mejor entendimiento del ulterior diseño.

2.1.1 Descripción de los procesos elementales de Control de Bienes Muebles, Materiales y Suministros en el almacén.

2.1.1.1 Crear o actualizar la ficha de un bien.

Este proceso ocurre siempre que se desea realizar algún movimiento sobre un bien en específico, por lo que el Responsable del Almacén(RA) busca la ficha del bien y la actualiza en dependencia del movimiento a realizar, en caso de no encontrar dicha ficha procede a elaborarla registrando así el movimiento realizado. **Anexo 1**

2.1.1.2 Incorporación de Bienes

En este proceso el donante del bien emite un acta que será revisada y aprobada por la Unidad Administradora (UA), la cual emite un acta de incorporación. En caso de que este bien no venga con una factura original se hace necesaria la ayuda de un perito para evaluar el bien y determinar su valor. Luego la UA realiza el registro en el auxiliar de bienes y lo envía junto con el acta de incorporación hacia el almacén. Donde será recibido por el RA el cual se encarga de incorporar el bien y actualizar o crear la ficha correspondiente además de actualizar el Libro Auxiliar de Bienes Existentes. **Anexo 2**

2.1.1.3 Desincorporación de Bienes

La UA emite un Acta de Desincorporación por los diferentes conceptos que correspondan a tales efectos, realiza un Informe Técnico de acuerdo al tipo de bien a desincorporar y elabora un Acta de Desincorporación, de la cual envía una copia al almacén. El RA recibe Acta de Desincorporación, verifica su validez y procede a Desincorporar el Bien. Actualiza la Ficha del Bien y se actualiza el Libro auxiliar de Bienes existentes. Firma la desincorporación y la envía a la UA. Esta recibe Acta de Desincorporación firmada por el RA y emite los asientos contables que correspondan. **Anexo 3**

2.1.1.4 Inventario Físico de los Bienes

La UA realiza periódicamente un conteo de los Bienes en Almacén, en el caso de encontrar alguna deficiencia se emite un acta para proceder a incorporar o desincorporar el bien según sea el caso teniendo en cuenta los procesos administrativos para justificar la diferencia. En caso de la desincorporación se inicia un proceso para investigar las causas. **Anexo 4**

2.1.2 Descripción de los Procesos Elementales para el Control de Bienes Mueble e Inmuebles en uso.

2.1.2.1 Reasignación de Bienes

La UA solicitante envía una Solicitud de Reasignación a la UA Transferente, esta recibe la solicitud y la revisa, en caso de no estar de acuerdo envía sus observaciones a la UA solicitante y se anula el trámite .En caso de aceptar actualiza la ficha del bien y la envía junto con el bien en cuestión a la UA solicitante. Este proceso se le Notifica a la Unidad Administradora Central (UAC), la cual se encarga verificar la calidad de la transferencia y de que la ficha este correctamente actualizada.

2.1.2.2 Depreciación de Bienes

La UA realiza mensualmente una lista con todos los bienes en uso y el valor de su depreciación, esta relación debe estar correctamente firmada para enviársela a la UAC y que esta emita el asiento contable correspondiente.

Una vez analizados estos procesos se está en condiciones de continuar con el desarrollo de las actividades pertenecientes al rol de diseñador, que en cuestión es el rol sobre el que se desarrolla el presente trabajo de diploma. A continuación se hará una explicación general de las principales características del Diseño y la Arquitectura definida para el módulo Control de Bienes haciendo énfasis en los tipos de clases existentes en cada una de las capas que conforman el módulo así como en sus principales características y funcionalidades.

2.2 Características fundamentales del diseño para el módulo Control de Bienes.

En este epígrafe se realiza una descripción del diseño adoptado en el módulo de forma general, con el objetivo fundamental de facilitar el entendimiento de la estructura y las principales particularidades del mismo.

El módulo Control de Bienes está estructurado siguiendo el patrón Arquitectónico en Capas, este facilita tanto la comprensión como la construcción del subsistema, reduce las dependencias entre las capas evitando así tener que realizar grandes cambios como consecuencia de pequeñas modificaciones y propicia excelentes condiciones para la reutilización al poder utilizar en las capas superiores las funcionalidades implementadas en inferiores. El módulo tiene como base un framework común que regula entre otras cosas la gestión de la presentación adoptando para esto una variante del Modelo Vista Controlador (MVC). Además maneja los temas relacionados con la seguridad, ya sea control de acceso, manejo de excepciones y navegabilidad dentro del módulo.

A continuación se hace referencia a cada una de las capas existentes en el módulo definiendo los tipos de clases que se pueden encontrar y su relación con los recursos brindados por el framework común antes mencionado.

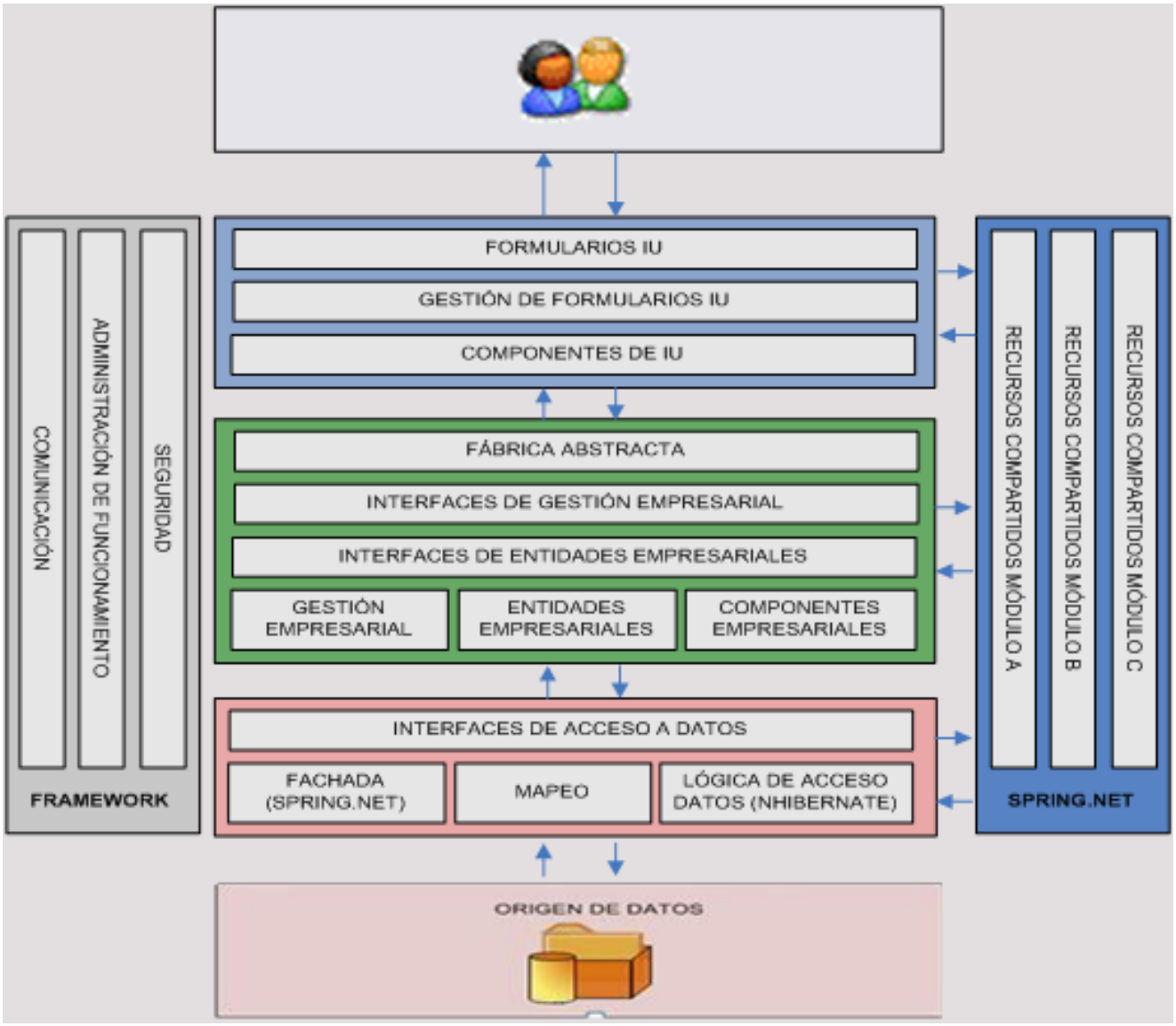


Figura 7 Vista de la arquitectura del módulo Control de Bienes

2.2.1 Capa de Presentación

En esta capa se encuentran todas las clases encargadas de una forma u otra de facilitar la interacción del usuario con el sistema. En ella se encuentran los formularios de interfaz de usuario los cuales responden a un diseño y comportamiento estándar en todos los módulos con vista a facilitar el trabajo con ellos y lograr la estandarización de la aplicación. Están basados fundamentalmente en dos tipos de formularios proveídos por el framework dependiendo de sus características. Frmmensaje (Figura 8) para las interfaces flotantes o FrmAreaDeTrabajo (Figura 9) para aquellas interfaces de tamaño grande que no son flotantes y se encuentran dentro del área de trabajo de la aplicación. El control de estos formularios esta dado por la utilización de gestores de interfaz de usuario, estos componentes son brindados por el framework, el cual facilita el desarrollo del sistema basándose en acciones. Estas acciones definen un bloque de código reusable por varias operaciones sobre el sistema. Pueden tener o no una forma visual asociada y en caso de tenerla se deben programar dentro de la acción todos los eventos que se deseen utilizar de dicho formulario. Las acciones deben heredar de las clases Accion o AccionSegura, que se encuentran en el framework dependiendo de la necesidad que exista de mantener un control del acceso sobre dicha acción.

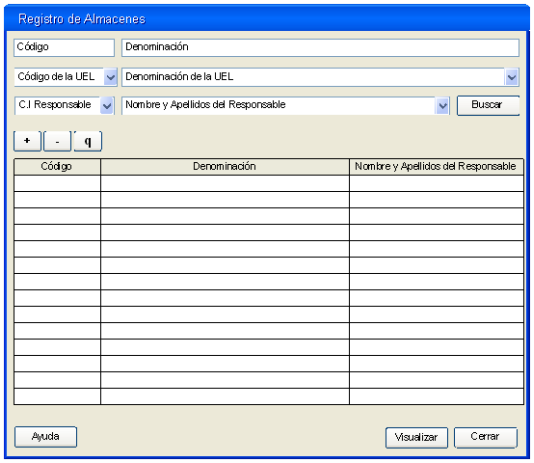


Figura 9. Formulario Estándar

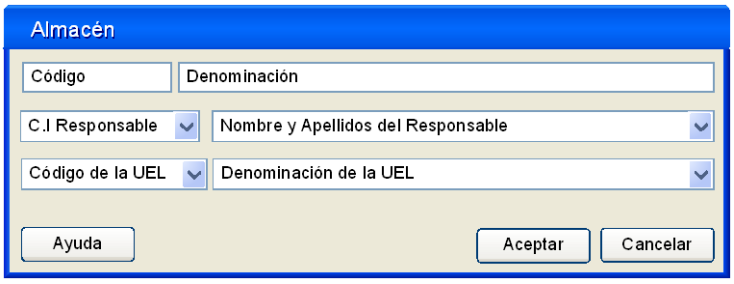


Figura 8. Formulario Flotante

En esta misma capa se pueden encontrar también los componentes de interfaz de usuario. Estos componentes tienen como objetivo fundamental encapsular una función determinada que será utilizada en más de una ocasión dentro del módulo. El hecho que se encuentren en la capa de presentación se debe precisamente a que trabajan directamente con los formularios, ya sea de forma visual o con los datos que se encuentran relacionados en el mismo a través de otros componentes.

2.2.2 Capa Lógica del Negocio

Esta capa está estrechamente relacionada con las características del negocio en cuestión, en ella se modela, gestiona y procesa la información obtenida de la presentación y en caso de necesitarse se interactúa con la Capa de Acceso a Datos, la cual se explicará más adelante en este mismo epígrafe.
Entre las clases que se encuentran en la capa lógica del negocio se pueden ver las Entidades del Negocio Figura 10, estas son clases objeto-valor que representan los datos con los que se va a trabajar en cada uno de los procesos que se están automatizando. Cada entidad es un elemento auto sustentado, es decir, se encarga de procesar sus propios datos o valores sin interactuar con los demás elementos del negocio, con esto se garantiza la independencia y el encapsulamiento de la información.

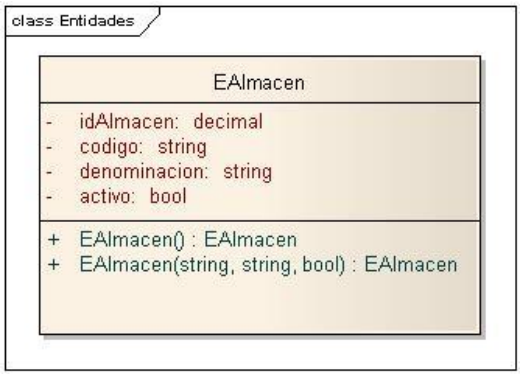


Figura 10 Clase entidad EAlmacen

También se encuentran las clases de Gestión del Negocio Figura 11, estas son las encargadas de proporcionar las funcionalidades que van a interactuar sobre las entidades. En estas clases se propone la utilización del patrón Singleton, debido a que en ellas solo se encuentran funcionalidades y con una única instancia se posibilita su ejecución efectiva. Al igual que en la capa de presentación existen componentes que agrupan una serie de funcionalidades que serán utilizadas en distintas ocasiones y para contribuir en disímiles funcionalidades del módulo, estos son los Componentes del Negocio y se encuentran solo en este nivel a pesar de que puedan tener en ocasiones relación con el Acceso a datos.

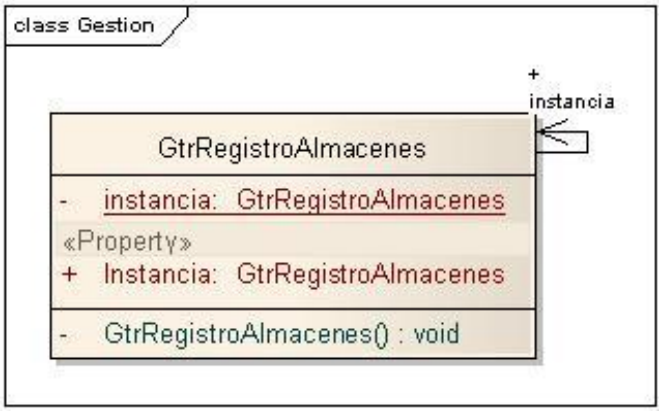


Figura 11 Clase Gestora GtrRegistroAlmacenes

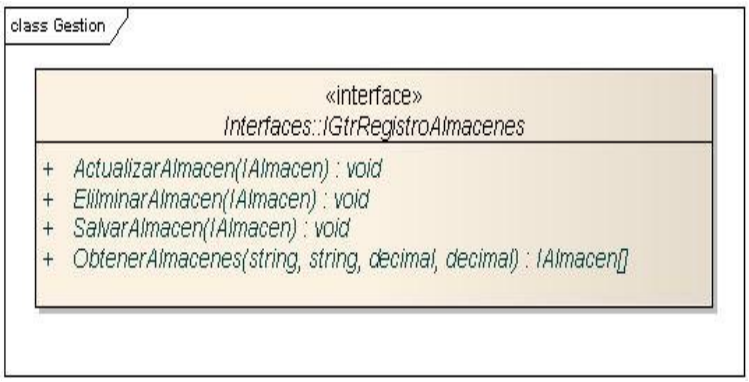


Figura 12 Interfaz IGtrregistroAlmacenes

Una premisa de la estructura definida para el módulo, es la utilización de Interfaces Figura 12. Estas son de gran importancia, pues se utilizan para brindar funcionalidades que más tarde serán implementadas. La mayoría de las interfaces de la aplicación van a estar precisamente en el Negocio, puesto que aquí se encuentran en su mayoría la implementación de estas funcionalidades, ya sea en las entidades, los gestores o los componentes de gestión. En esta capa se aplica el patrón de diseño *Fábrica Abstracta* definiéndose para eso una clase con la responsabilidad de instanciación ya sea de las entidades o los gestores de negocio. Otro de los patrones de diseño aplicados en esta capa es el patrón Estado, el cual se utiliza para permitir que el objeto cambie su comportamiento a medida que se modifica su estado.

2.2.3 Capa de Acceso a Datos

Definir la estrategia de persistencia de una aplicación es una de las decisiones de arquitectura más importantes. En una aplicación estándar más del 50% del código generado está relacionado con lógica de persistencia. Partiendo de esto, el Acceso a Datos se considera la capa más crítica y sensible de la arquitectura, pues controla todo lo concerniente a la información que se encuentra en la fuente de almacenamiento (Capa de Datos).

Al ser la capa inferior no conoce los niveles superiores, únicamente se limita al manejo de la información, ya sea para persistirla o proporcionarla para su procesamiento y propagación por la aplicación. Aquí se propone el uso de Nhibernate versión 1.1, framework orientado a objetos para la persistencia de datos.

En esta capa se implementa el patrón Fachada buscando abstracción, proporcionando una especie de interfaz con la cual interactuará la capa de lógica de negocio. Las clases que intervienen en la fachada son todas estáticas puesto que solo funcionan como puente de acceso a las funcionalidades y es necesario instanciarlas. Para la implementación de la fachada se decidió la utilización del framework Sprint.Net, el cual proporciona la implementación de una buena cantidad de patrones de diseño, facilitando la implementación por parte del equipo de desarrollo. En esta capa también se encuentran los ficheros de *mapeo*, estos son ficheros de uso exclusivo por el Nhibernate pero se encuentran separados en un único ensamblado producto a que son ficheros XML. Estos ficheros son los que

definen la relación existente entre las clases del negocio y las tablas que se encuentran en la capa de datos, logrando que la persistencia de las clases objeto-valor se realice de forma segura y rápida. Las responsabilidades de persistencia y obtención de objetos, ya sea a través de procedimientos de almacenado o la ejecución de consultas recae sobre el ensamblado de *Lógica de Acceso a Datos.* En este ensamblado se encuentran los Data Access Objects (DAO), en ellos se implementan la totalidad de las operaciones de interacción con la capa de datos explotando al máximo las potencialidades que brinda Nhibernate.Net.

2.2.4 La Fachada

El módulo Control de Bienes forma parte de un gran subsistema, motivo por el cual debe interactuar de forma estable con el resto de los módulos definidos. La integración de estos módulos tiene como característica fundamental la implementación de una fachada en la cual se encontrarán todas las funcionalidades que se desean compartir con el resto del sistema. Con la adopción de esta modalidad de comunicación entre los módulos se gana en abstracción y se reduce la dependencia entre los mismos; con la utilización del framework Sprint.Net para su implementación se mantiene la uniformidad y estabilidad del módulo.

2.3 Artefactos involucrados en el diseño.

A continuación se detallan algunos de los artefactos fundamentales generados en el diseño. Es válido resaltar que estos son los artefactos seleccionados por la dirección del proyecto como los fundamentales por lo cual son los que se generan durante el desarrollo del presente trabajo de diploma.

2.3.1 Modelo de diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de usos, centrándose en el impacto que puede tener los requisitos funcionales/no funcionales y otras restricciones relacionadas con el entorno de implementación. Las abstracciones del modelo de diseño tienen una correspondencia directa con los elementos físicos del ambiente de implementación.

2.3.1.1 Clase del diseño

Una clase de diseño es una abstracción de una clase de implementación, donde las operaciones, atributos, tipos, visibilidad, se pueden especificar con la sintaxis del lenguaje elegido.

2.3.1.2 Diagrama de Transición de Estados.

Algunos objetos del diseño poseen estados controlados que determinan su comportamiento cuando reciben un mensaje. En casos como este es viable la creación de un diagrama de transición de estados para describir las distintas transiciones a las que puede verse sujeto dicho objeto del diseño.

2.3.1.3 Realización de caso de uso-diseño

Un Caso de Uso (CU) es una secuencia de transacciones que son desarrolladas por un sistema en respuesta a un evento que inicia un actor sobre el propio sistema. El objetivo de la Realización de los Casos de Uso es conseguir separar la especificación del sistema (Modelo de Casos de Uso) del diseño del sistema, distribuyendo el comportamiento definido en los casos de uso entre las clases del modelo de diseño. La realización de un caso de uso ofrece una visión más cercana al diseño de la solución planteada para el escenario descrito por el caso de uso. Describe la manera en cada caso de uso es realizado y ejecutado en términos de clases y sus objetos. El documento generado en esta fase puede contener varios elementos para representar la realización de un caso de uso como son los diagramas de clases del diseño, los diagramas de interacción, haciendo uso de los diagramas de secuencia por la necesidad de detallar las secuencias de interacciones y ordenarlas en tiempo, y también se obtiene un flujo de sucesos-diseño que básicamente consiste en una descripción textual del flujo de eventos, descripción que explica y complementa los diagramas.

- **Diagramas de Clases**

Este diagrama muestra la especificación detallada de cada una de las clases modeladas, es decir sus atributos, operaciones que cubrirán las responsabilidades identificadas en el análisis, métodos que implementan dichas operaciones y el diseño preciso de las relaciones que se establecen entre las clases, bien sea de agregación, asociación o generalización. Las clases de diseño a menudo participan en varias realizaciones de casos de uso.

- **Diagramas de Interacción**

Los diagramas de interacción ilustran el flujo de interacciones entre las clases y subsistemas participantes. Estos diagramas de interacción son normalmente diagramas de secuencia y de colaboración, que permiten expresar fácilmente el comportamiento de los casos de uso en función de objetos que colaboran entre sí para realizar una operación. Estos diagramas contienen objetos, enlaces, mensajes y también pueden contener notas y restricciones.

- **Diagramas de Secuencia**

Los diagramas de secuencias muestran las interacciones entre los objetos o subsistemas mediante transferencia de mensajes. Suele ser útil crear un diagrama por cada escenario aportando claridad en la realización del caso de uso, en ellos se destaca la ordenación temporal de los mensajes. Para su creación se comienza por el principio del flujo del caso de uso, y después seguir ese flujo detalladamente, decidiendo qué objetos del diseño y qué interacciones son necesarias para realizar cada paso.

2.3.1.4 Subsistema y Paquetes de diseño

Los subsistemas de diseño son utilizados para organizar los artefactos del modelo de diseño en piezas más manejables, puede contener clases de diseño, realizaciones de casos de uso, interfaces y otros subsistemas. Los paquetes de diseño se utilizan para la recopilación de clases, relaciones, diagramas y otros paquetes. Paquetes que estructuran el modelo de diseño dividiéndolo en componentes más pequeños y organizados.

2.4 Realización de los Casos de Uso del módulo Control de Inventario

Ya conocidos los artefactos involucrados en el diseño a continuación se presenta la realización de algunos de los casos de uso más significativos para el módulo Control de Bienes.

2.4.1 Realización del CU: Inventario

Para la realización del caso de uso Inventario se hace una representación estática y dinámica de las clases, interfaces y objetos a través de los diagramas de clases de diseño y de secuencia. También se representa el conjunto de estados por los cuales pasa un objeto durante su vida y los eventos que provocan el cambio de su estado mediante un diagrama de transición de estados.

2.4.1.1 Diagrama de Clases CU: Inventario

En este diagrama se muestran las relaciones entre las clases de diseño identificadas para el CU: Inventario. Entre las clases más relevantes se encuentra el DaoComun y DaoInventario, ambas con la funcionalidad de realizar el acceso a datos desde el negocio, también se encuentra la FabricaInventario encargándose de instanciar las entidades y clases gestoras. La clase GtrInventario constituye un puente para el flujo de información entre las capas de Presentación y Datos. Otra de las clases de gran importancia es la AccInventarioAlmacenes y su funcionalidad es controlar los eventos que puedan ser ejecutados desde el formulario. Es importante resaltar la utilización de la clase RecursosCompartidos perteneciente a la Fachada para obtener instancias de clases perteneciente a otros módulos.

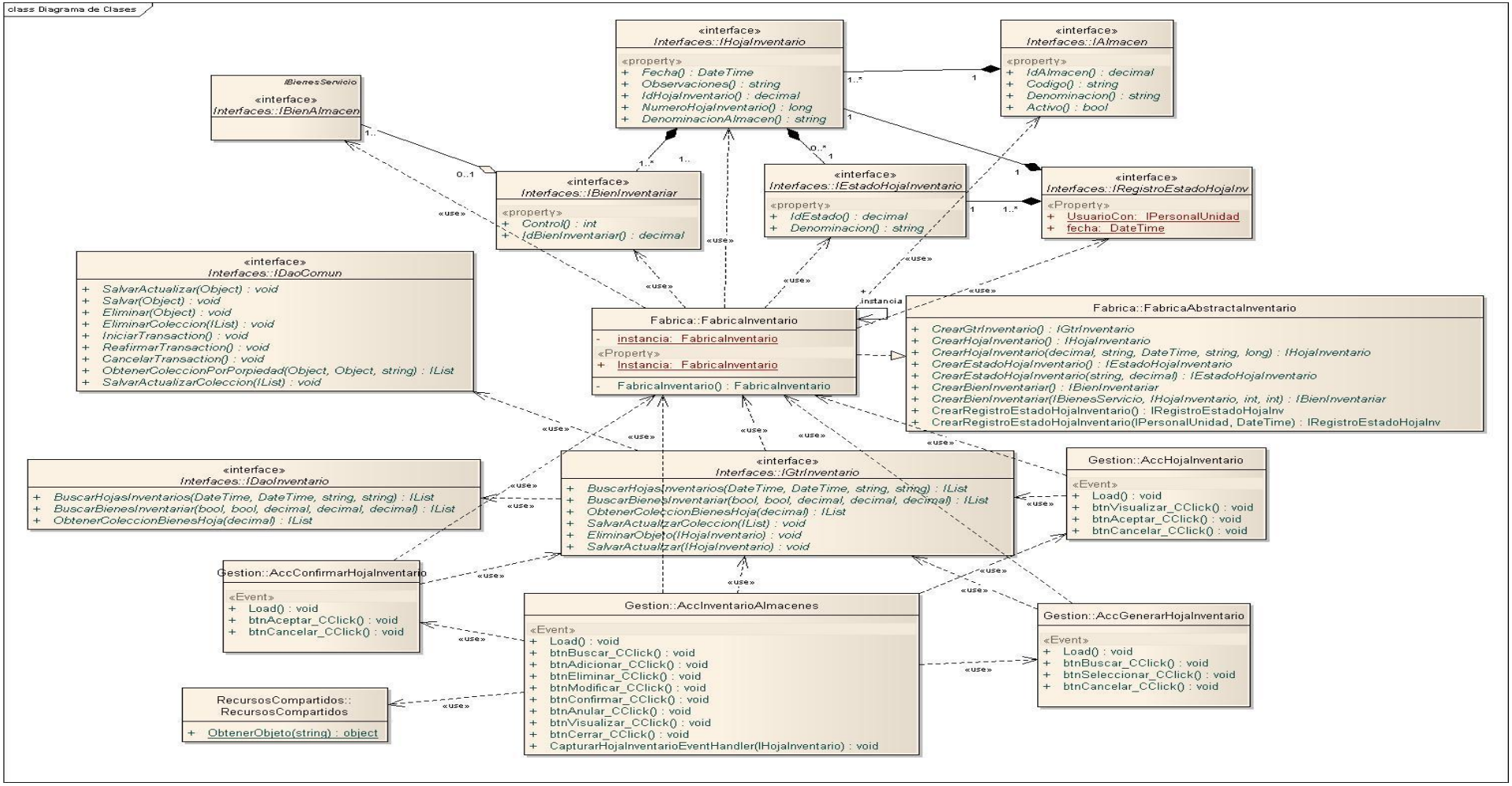


Figura 13 Diagrama de Clases del Diseño CU: Gestionar Hojas de Inventario

2.4.1.2 Diagrama de Transición de Estados

El siguiente diagrama de transición de estado muestra el comportamiento de la entidad Hoja Inventario y el conjunto de estados por el cual puede pasar dicha entidad desde su creación.

Entidad: Hoja Inventario

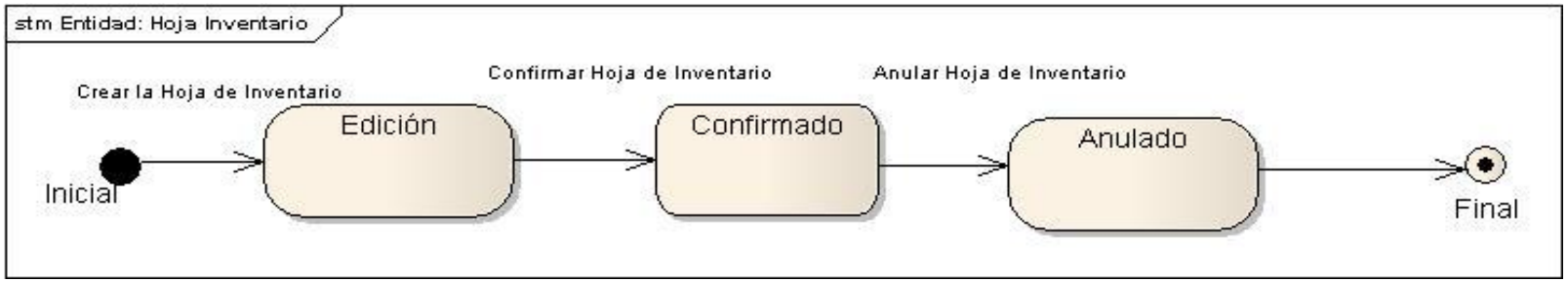


Figura 14 Diagrama de Transición de Estados Entidad Hoja de Inventario

2.4.1.3 Diagrama de Secuencia CU: Inventario

Con el objetivo de facilitar la comprensión del diagrama y así lograr un mayor entendimiento de la secuencia de acciones que se suceden se realiza un diagrama por cada escenario posible dentro del caso de uso:

- **Escenario: Buscar Hoja Inventario**

Este escenario representa una búsqueda de hojas de inventario y tiene como patrones de búsqueda el almacén, el estado y un rango de fechas.

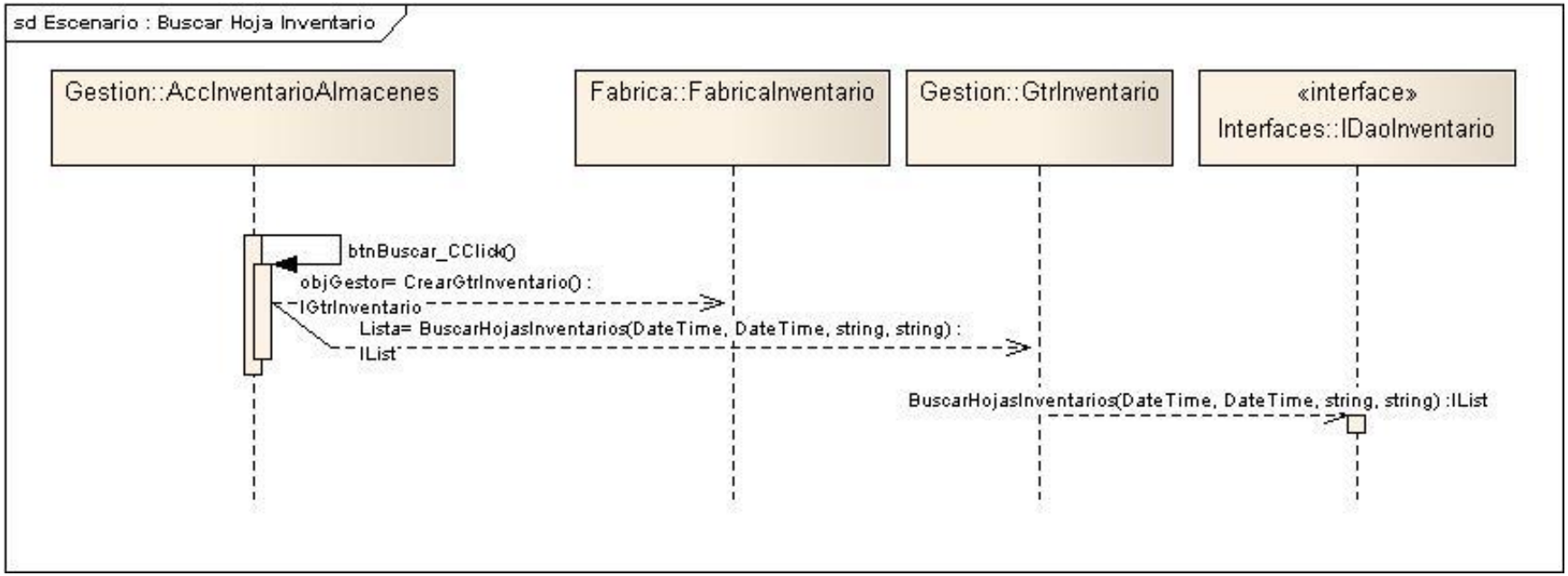


Figura 15 Escenario Buscar Hoja de Inventario

- **Escenario: Eliminar Hoja Inventario**

Este escenario permite eliminar una hoja de inventario seleccionada, es importante resaltar que esta acción solo es válida en caso que se refiera a una hoja en estado de Edición.

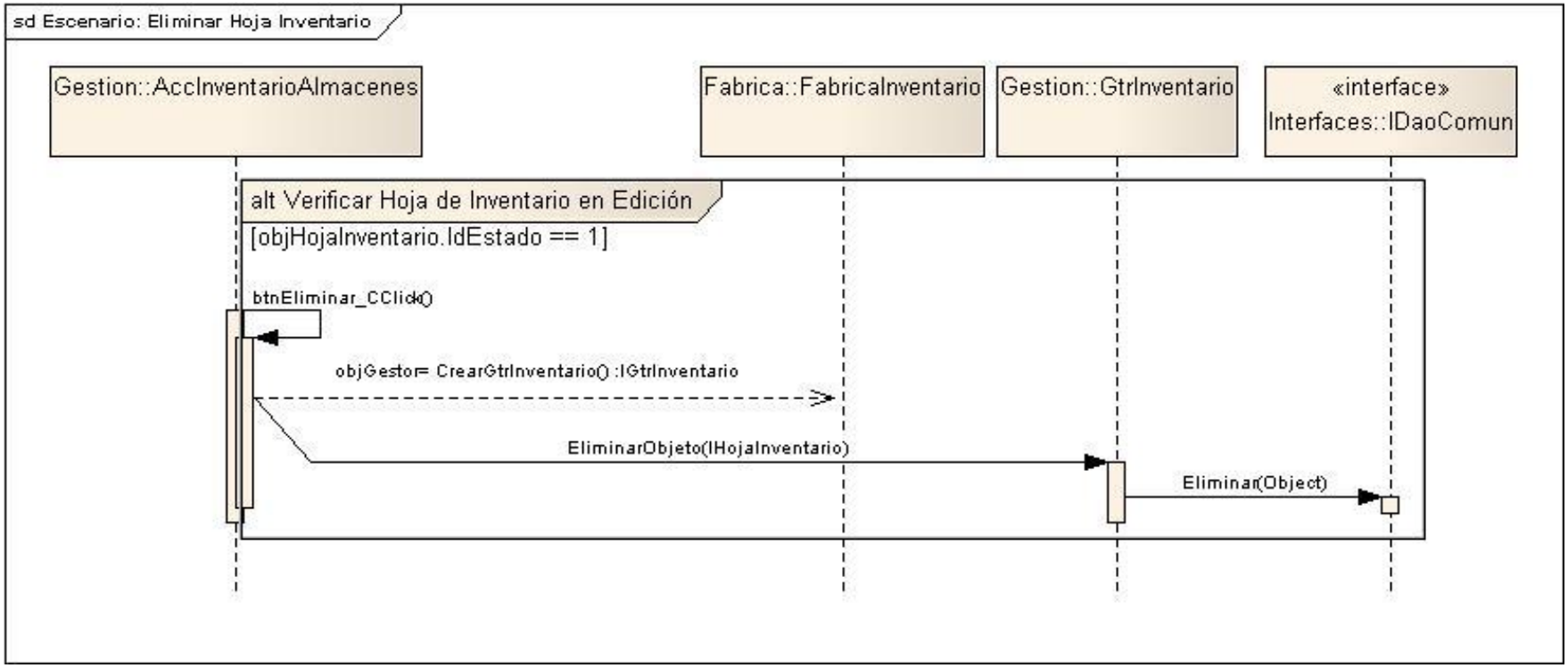


Figura 16 Escenario Eliminar Hoja de Inventario

- **Escenario: Adicionar Hoja Inventario**

Escenario que posibilita la creación de una nueva hoja de inventario.

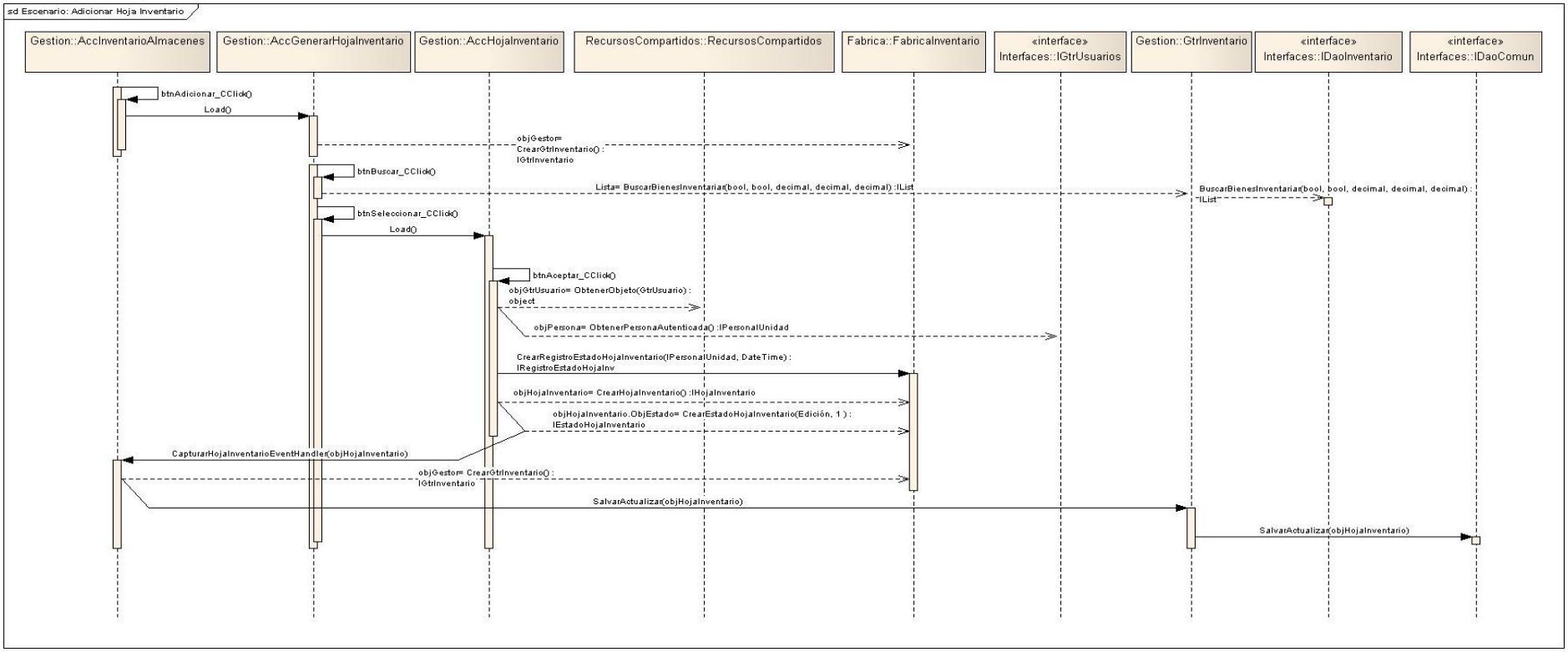


Figura 17 Escenario Adicionar Hoja de Inventario

- **Escenario: Modificar Hoja Inventario**

Este escenario permite modificar una hoja de inventario existente. Es importante señalar que solo las hojas de inventario que se encuentran en estado de Edición pueden ser modificadas.

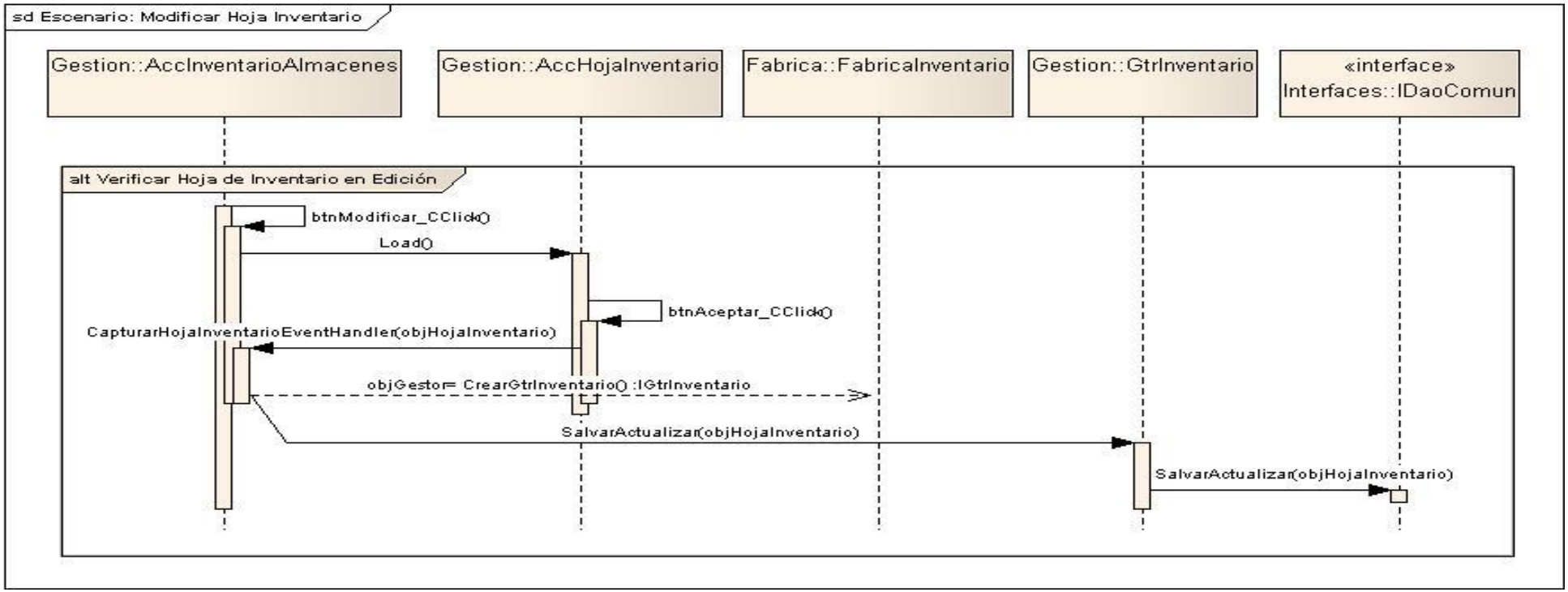


Figura 18 escenario Modificar Hoja de Inventario

- **Escenario: Confirmar Hoja Inventario**

Este escenario permite confirmar una hoja de inventario seleccionada, es importante resaltar que esta acción solo es válida en caso que se refiera a una hoja en estado de Edición.

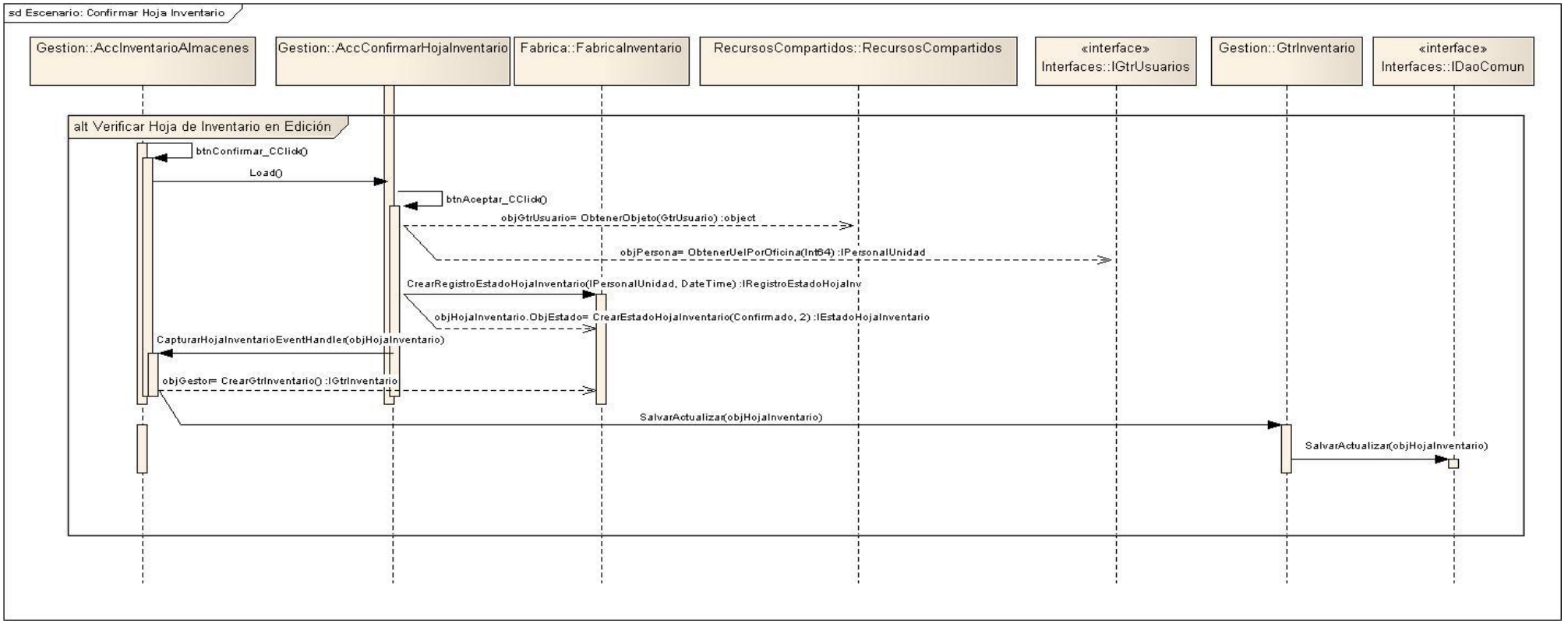


Figura 19 Escenario Confirmar Hoja de Inventario

- **Escenario: Anular Hoja Inventario**

Este escenario permite anular una hoja de inventario seleccionada, es importante resaltar que esta acción solo es válida en caso que se refiera a una hoja en estado de Confirmada.

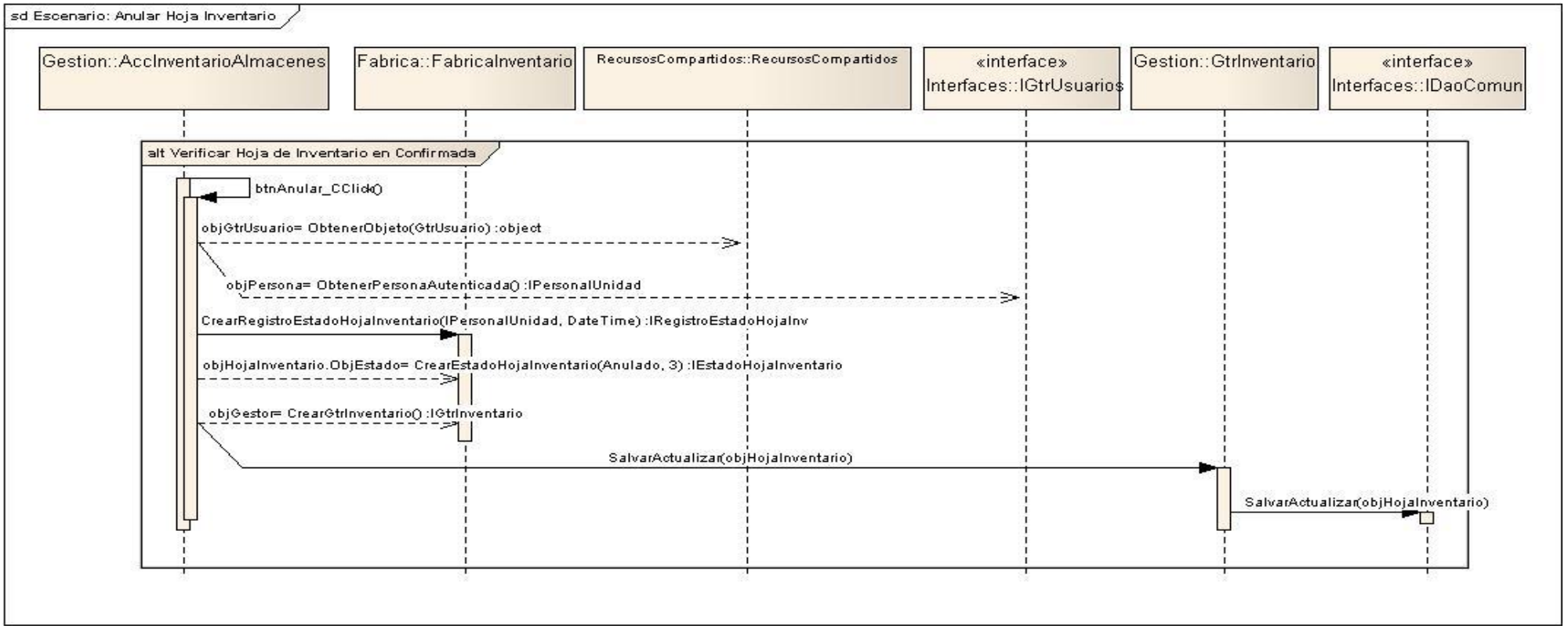


Figura 20 Escenario Anular Hoja de Inventario

2.4.2 Realización del CU: Gestionar Almacenes

El CU Gestionar Almacenes a pesar de no ser muy complejo en su desarrollo es considerado uno de los más significativos dentro del módulo Control de Bienes. Esta relevancia está dada por constituir el almacén un codificador de alta prioridad para el funcionamiento del sistema. Para ver los diagramas de su realización ver **Anexo 5**

2.4.3 Realización del CU: Gestionar Grupos y Subgrupos

Este CU tiene gran importancia para el sistema, ya que permite agrupar los bienes teniendo en cuenta sus características o finalidad, permitiendo que se le puedan dar un tratamiento identificado. Para ver los diagramas de su realización ver **Anexo 6**

2.4.4 Realización del CU: Gestionar Órdenes de Entrega

El caso de uso Gestionar Orden de Entrega reúne las características de ser relevante dentro del sistema y a su vez ser uno de los casos de uso de mayor complejidad. Permite mantener el control de los bienes que son entregados directamente a las UEL, generando movimientos de manera automática que mantienen actualizada la contabilidad del sistema, para lo cual se apoya en el CU: Generar Movimiento por Orden de Entrega. La entidad orden de entrega está expuesta a una serie de estados, a través de los cuales se determina el momento en el que se realiza un asiento contable u otro. Para ver los diagramas de su realización ver **Anexo 7**

2.4.5 Realización del CU: Generar Movimiento por Orden de Entrega

Este CU es un extend del caso de uso Gestionar Orden de Entrega. Posee una gran importancia, pues tiene la responsabilidad de crear los movimientos de los bienes correspondientes a las órdenes de entrega una vez que estas son confirmadas o anulada. Por tal motivo consta de dos escenarios, uno crea un movimiento de salida que respalda a la orden de entrega confirmada y el otro crea en movimiento de entrada invirtiendo todas las salidas de una orden ya confirmada que acaba de ser anulada. Para ver los diagramas de su realización consultar la documentación del proyecto.

2.4.6 Realización del CU: Gestionar Movimiento de Bienes

Este caso de uso es de vital importancia para la contabilidad del sistema, debido a que tiene la responsabilidad de crear asientos que modifican el estado de las cuentas contables. El caso de uso

gestionar movimiento permite que se cree, edite o elimine un movimiento siempre que su estado lo permita. La entidad movimiento solo cuenta con dos estados (Edición y Confirmado) debido al gran control que requiere este tipo de actividad. En caso que se confirme un movimiento erróneamente es necesario crear uno nuevo para revertir el anterior. Para ver los diagramas de su realización consultar la documentación del proyecto.

2.4.7 Realización del CU: Conocer Existencia de Bienes en Almacén

Este es el caso de uso menos crítico de todos los mencionados, y es precisamente por su función netamente informativa. Como su nombre lo indica la función principal de este CU es permitir al usuario conocer los almacenes que tienen un determinado bien, así como el historial generado por cada uno de los movimientos realizados sobre ese bien en el almacén. Para ver los diagramas de su realización consultar la documentación del proyecto.

2.5 Integración de los Subsistemas.

Para facilitar su desarrollo el módulo Control de Bienes ha sido concebido en dos fases, encontrándose actualmente en la primera fase, concerniente a los Bienes en Almacén. Motivo por el cual en el modelo de subsistemas que se presenta en este epígrafe aparecen procesos que no han sido modelados en el trabajo de diploma.

Como se observa en la Figura 21 los procesos del módulo Control de Bienes han sido agrupados en dos subsistemas de diseño teniendo en cuenta las distintas ubicaciones que puede tener el bien utilizado en cada uno de ellos. Por ejemplo el subsistema de Bienes en Almacén agrupa las funcionalidades Gestionar Orden de Entrega, Gestionar Hojas de Inventario y Gestionar Existencia de Bienes en Almacén, entre otras, mientras que el subsistema Bienes en Uso se encarga de Gestionar Bienes en Uso, Gestionar Depreciación de los bienes, y otros. Como bien indican los nombres de los subsistemas el de Bienes en Almacén controla los bienes que se encuentran en almacén y por lo tanto pueden ser sometidos a estos procesos mencionados anteriormente, mientras que bienes en uso controla el bien una vez que se encuentra en explotación. Estos subsistemas se encuentran relacionados entre sí por la necesidad de utilizar elementos comunes y a la vez se relacionan con los subsistemas Administración y Contabilidad. El subsistema Administración es el proveedor de configuraciones y algunos elementos que se encuentran precisamente allí por ser comunes para todo el sistema, mientras que Contabilidad brinda los recursos

necesarios para el trabajo con las cuentas contables, funcionalidad de gran importancia dentro de un sistema de gestión como este.

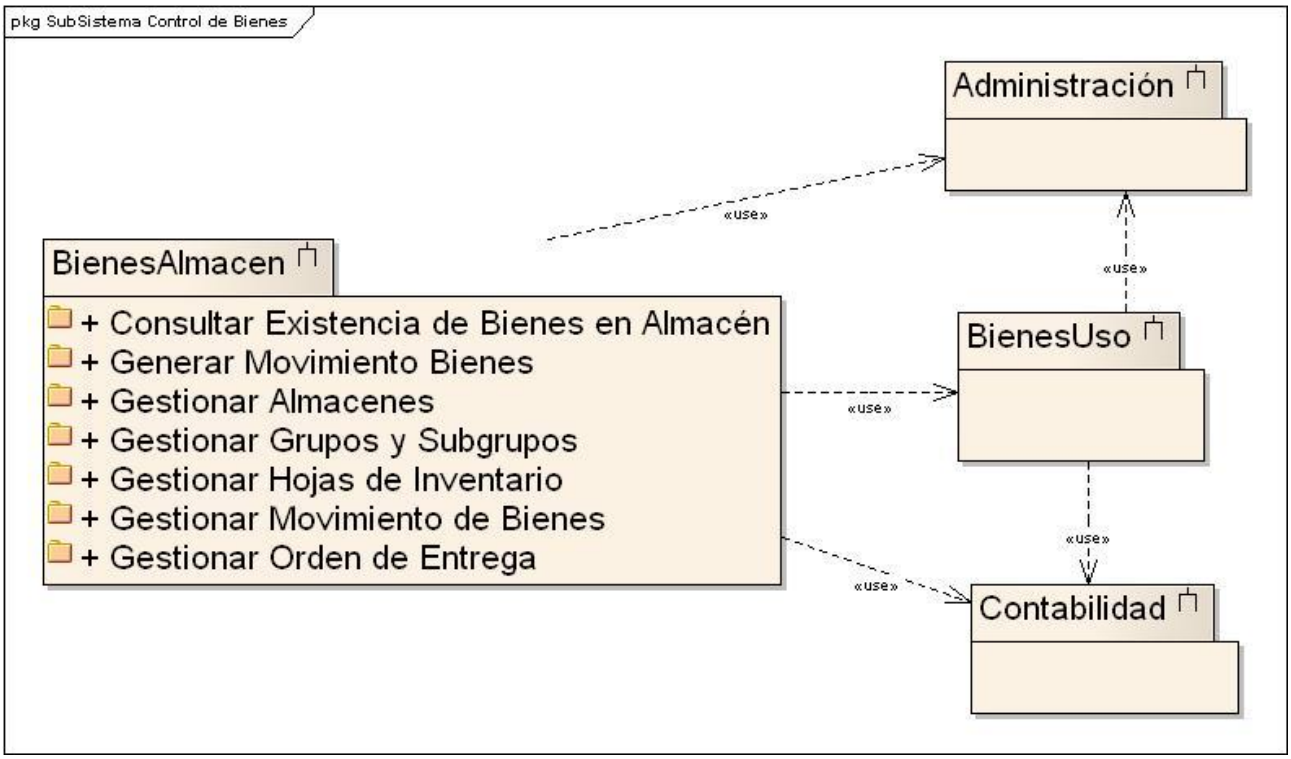


Figura 21 Modelo de subsistemas de diseño del módulo Control de Bienes

Conclusiones del Capítulo:

Durante el desarrollo de este capítulo se ha integrado y llevado a la práctica el que hacer correspondiente al rol de diseñador de sistema dentro de un equipo de desarrollo de software. Obteniendo como resultado el diseño del módulo Control de Bienes, satisfaciendo de esta manera el objetivo específico planteado al inicio de este trabajo de diploma.

Capítulo 3: Análisis de los resultados

Introducción

Es importante valorar si el diseño obtenido se ajusta a un nivel de calidad requerido para así poder conocer la efectividad de los procesos que han sido modelados y si se necesita de gran esfuerzo para su implementación. Por tal motivo en el presente capítulo se aplican un conjunto de métricas de diseño orientado a objetos y se analizan los resultados obtenidos. Los valores referentes a la cantidad de atributos y funciones por clases, así como los resultados de la aplicación de las métricas se encuentran en el **Anexo 8**. Es necesario plantear que dichos valores constituyen estimaciones conservadoras debido a que generalmente durante la fase de implementación se hace necesario incluir nuevas funcionalidades y atributos.

3.1 Resultado de las métricas orientadas a clases

A continuación se aplican un conjunto de métricas orientadas a clases con el objetivo de determinar el grado de calidad y fiabilidad del diseño propuesto en el capítulo anterior. Se seleccionaron las métricas a aplicar de dos grupos distintos para de esta manera aumentar el rigor de las pruebas realizadas. Estas métricas en la actualidad resultan ser unas de las más usadas.

3.1.1 Métricas propuestas por Lorenz y Kidd. Aplicación al modelo.

3.1.1.1 Tamaño de Clase (TC).

Primeramente para conocer el tamaño de una clase es necesario conocer la cantidad de operaciones y números de atributos que poseen, para así poder determinar el valor del tamaño de dicha clase. También se puede determinar el TC, calculando el promedio de los atributos y operaciones encapsulados en una clase. En la Tabla 2 se muestran las medidas o umbrales para los parámetros de calidad que fueron aplicados al diseño obtenido, dichos umbrales constituyen una polémica en el diseño de sistemas a nivel mundial por la variedad de medidas que brindan diferentes especialistas.

Nro. de operaciones y/o atributos	
TC	**Umbral**
Pequeño	**<= 20**
Medio	**>20 y <=30**
Grande	**>30**

Tabla 2 Umbrales para TC.

Los resultados obtenidos al aplicar la métrica TC al diseño propuesto fueron los siguientes:

Como se muestra en la Tabla 3 para un total de 71 clases existentes en el diseño se obtuvo un promedio de 9.9 operaciones y 3.8 atributos.

	Promedio de	
Total de Clases	**Operaciones**	**Atributos**
71	**9.9**	**3.8**

Tabla 3 Cantidad de clases de Diseño, operaciones y atributos promediados.

Según los umbrales propuestos se obtuvo que para un total de 71 clases presentes en el diseño, 66 son pequeñas, 3 de tamaño medio y solo 2 con un tamaño grande. Representándose en la Tabla 4.

Cantidad de clases	Umbral	Tamaño
66	**<=20**	**Pequeño**
3	**>20 y <=30**	**Medio**
2	**>30**	**Grande**

Tabla 4 Cantidad de clases por tamaño.

Como se puede observar en la Figura 22 solo el 3 y el 4 por ciento constituyen clases grandes y medianas respectivamente, quedando la mayor cantidad de clases dentro del rango correspondiente a la clasificación de pequeñas, esto resulta ser positivo debido a que al obtener valores bajos para el TC aumenta la reutilización, se hace más fácil su implementación y la realización de pruebas en fases posteriores.

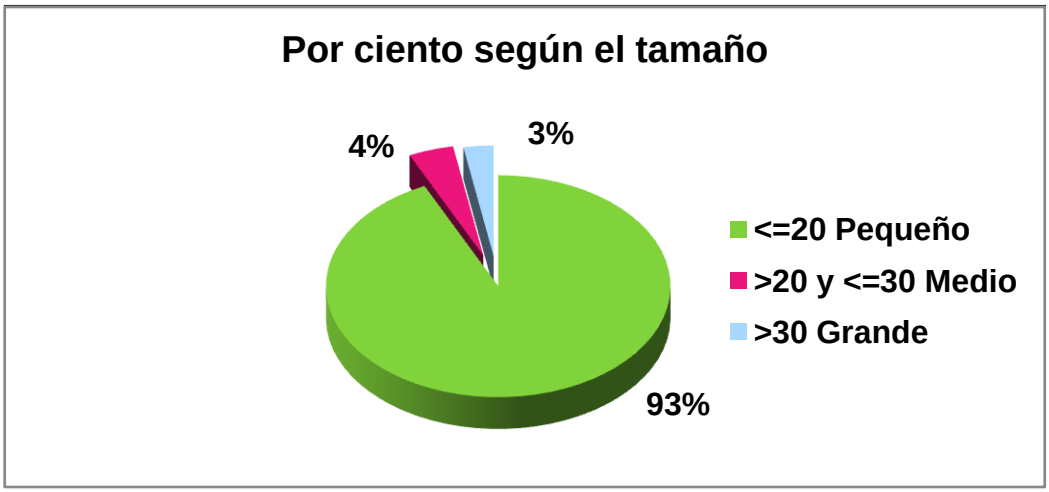


Figura 22 Representación de las clases según su TC

3.1.1.2 Número de Operaciones redefinidas por una subclase (NOR).

Para poder aplicar esta métrica es necesario conocer la cantidad de clases que redefinen métodos heredados de otras clases. La Tabla 5 muestra los valores obtenidos.

Cantidad de clases de diseño	Total de subclases que redefinen operaciones
71	12

Tabla 5 Total de subclases que redefinen operaciones.

Otra forma representativa es el por ciento de subclases que redefinen operaciones donde queda representado que el 17% de las clases existentes remplazan operaciones.

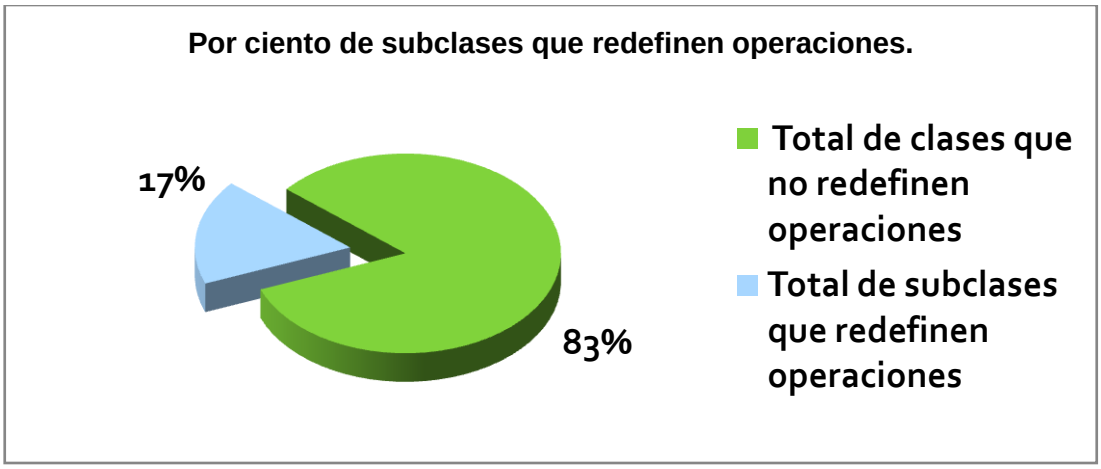


Figura 23 Por ciento de subclases que redefinen operaciones.

Como se puede observar el valor obtenido para la aplicación de la métrica NOR es relativamente bajo por lo que no se afecta la calidad del diseño propuesto, que puede ser modificado o llevado a pruebas sin dificultad.

3.1.2 Familia de métricas propuestas por Chidamber & Kemerer. Aplicación al modelo.

3.1.2.1 Árbol de profundidad de herencia (APH).

Si se desea conocer el APH del diseño obtenido primeramente se debe tener conocimiento del nivel de jerarquía de clases que presenta dicho diseño.

A continuación en la Tabla 6 se muestran los umbrales o medidas recomendadas por la documentación de Visual Studio .Net, estos umbrales brindan un conocimiento sobre la complejidad que puede alcanzar el diseño.

Nivel de jerarquía de clases	
APH	**Umbral**
Sencilla	**<= 5**
Compleja	**>5**

Tabla 6 Umbral para la Métrica APH

Apoyándose en los umbrales definidos se obtiene el siguiente resultado:

Total de Clases	APH
71	2

Tabla 7 Resultados aplicando la métrica APH

Como se puede observar el APH para la jerarquía de clases del diseño posee valor 2, al tener un indicador de nivel sencillo evita que exista problema si se desea predecir el comportamiento de una clase y también hace que el diseño no presente complejidad alguna.

3.1.2.2 Número de descendiente (NDD).

Para obtener el NDD es necesario conocer el número de subclases inmediatamente subordinadas a una clase en la jerarquía existente del diseño propuesto. A continuación en la Tabla 8 se muestra el número de clases presentes en el diseño que poseen descendientes.

Nro. de clases	Nro. de descendientes
7	1
2	2
1	3

Tabla 8 Cantidad de clases por cantidad de descendientes

Como se puede observar para un total de 71 clases contenidas en el diseño propuesto, existen siete clases con NDD igual a 1, otras dos con valores iguales a 2 y solo una con 3 descendientes.

Los resultados alcanzados para el NDD del diseño propuesto son relativamente pequeños asegurándose con esto de que cada descendiente es realmente miembro de la clase predecesora y también al haber obtenido un valor pequeño del NDD se reduce la cantidad de pruebas necesarias a para ejercitar cada uno de los miembros de la jerarquía.

3.1.2.3 Acoplamiento entre clases objeto (ACO).

Si se desea conocer el valor de ACO para el diseño propuesto primeramente se debe listar para una clase el número de colaboraciones que posee. Para aplicar esta métrica solo se medirán los valores de las colaboraciones existentes en las entidades del negocio. A continuación en la Tabla 9 se muestra la cantidad de clases que poseen dichas colaboraciones.

Nro. de clases	Nro. de Colaboraciones
9	1
11	2
5	3
4	4
1	5
2	6
1	7

Tabla 9 Colaboraciones por clases

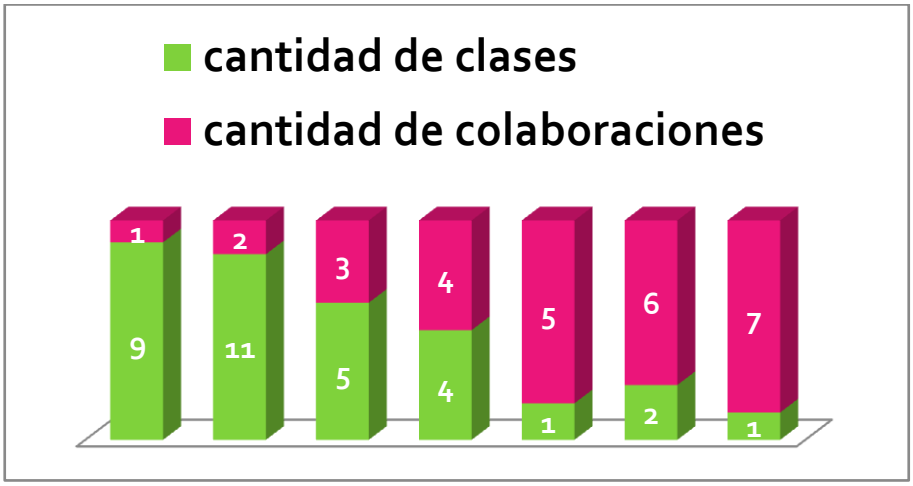


Figura 24 Colaboraciones por clases.

Conocido los valores para las colaboraciones existentes en las entidades del negocio se puede dar el siguiente resultado:

El diseño propuesto no presenta complejidad alguna, como se puede observar en la Figura 24 el valor que indica el ACO según los datos que se muestran son relativamente bajos, es decir existe un bajo acoplamiento permitiendo que aumente el grado de reutilización de las clases existentes, también este resultado ayuda para que las pruebas o modificaciones necesarias a realizar sobre el diseño resulten fáciles de ejecutar.

Conclusiones del Capítulo:

Con el desarrollo del capítulo se logra dar solución al objetivo planteado, aplicando las métricas propuestas para la evaluación del diseño obtenido. El diseño no posee alta complejidad estructural, evidenciándose en los valores obtenidos como resultado de las métricas aplicadas. Se puede concluir que el diseño obtenido posee una calidad aceptable, facilitando la continuación eficiente del desarrollo en etapas posteriores.

Conclusiones Generales:

Luego de haber finalizado el presente trabajo, se considera haber cumplido con todos los objetivos propuestos:

- Se caracterizaron los procesos que forman parte del Control de Bienes en los Registros y Notarías de Venezuela, permitiendo su comprensión con vistas al desarrollo del diseño del subsistema.
- Se aplicaron los patrones de diseño definidos en la línea base de la arquitectura y otros, garantizando un diseño óptimo, reutilizable y escalable.
- Se obtuvo el modelo de diseño de la solución informática para el Control de Bienes en los Registros y Notarias de Venezuela. Obteniendo una representación significativa del producto final a construir.
- Se evaluó el diseño aplicando una serie de métricas que determinaron la calidad del resultado obtenido. Certificando un nivel adecuado de utilización de los principios del Diseño Orientado a Objetos.

Recomendaciones:

- Refinar el diseño propuesto para el módulo Control de Bienes perteneciente al subsistema Administración Financiera.
- Realizar el diseño de la segunda fase del módulo Control de Bienes correspondiente al subsistema Bienes en Uso.
- Realizar la Implementación del módulo Control de Bienes perteneciente al subsistema Administración Financiera a partir del diseño obtenido durante el desarrollo del presente trabajo de diploma.
- Valorar la integración de la Capa de Datos con la Capa de Negocio a través de las potencialidades que ofrece Spring para ello.

Bibliografía

1. definicion.org. [En línea] [Citado el: 10 de 11 de 2008.] www.definicion.org.

2. **Rossainz López, Mario.** *Diseño Orientado a Objetos.*

3. **Bertrand, Meyer.** *Construccion de Software Orientado a Objetos. 2da Edicion.* s.l. : Prentice Hall, 1999.

4. **Stephen, Albin.** *The Art of Software Architecture: Design methods and techniques.* Nueva York : s.n., 2003.

5. IEEE Sitio Oficial. [En línea] [Citado el: 12 de 2 de 2008.] http://standards.ieee.org.

6. microsoft.com. [En línea] [Citado el: 13 de 2 de 2008.] http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.mspx#EMJAC.

7. **Reynoso, Carlos y Kicillof, Nicolás.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires : s.n., 2004.

8. **Pestano Pino, Henrik y Montane, Juan Carlos.** *Diseño del modulo para la Administracion Contable y Financiera de los Registros y Notarias de la Republica Bolivariana de Benezuela.* Ciudad de la Habana : s.n., 2007.

9. *Descripcion de Procesos de Negocio del Proyecto de Modernización de los Registros y Notarías.* **Equipo SAREN, Equipo ALBET.** 2007.

10. **Clements, Paul.** *"A Survey of Architecture Description Languages". Proceedings of the International Workshop on Software Specification and Design.* Alemania : s.n., 1996.

11. **Shaw, Mary y Garlan, David.** *Software Architecture: Perspectives on an emerging discipline. Upper Saddle River.* Prentice Hall : s.n., 1996.

12. **Fowler, Martin, y otros.** *Patterns Of Enterprise Application Architecture.* 2002.

13. **Mendoza Sanchez, María A.** informatizate.net. *informatizate.net.* [En línea] [Citado el: 13 de 2 de 2008.] http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html.

14. informatica-hoy.com. *informatica-hoy.com.* [En línea] [Citado el: 10 de 2 de 2008.] http://www.informatica-hoy.com.ar/software-erp/Evolucion-Historica-del-Software-ERP.php.

15. BetSime. [En línea] [Citado el: 9 de 2 de 2008.] http://www.betsime.disaic.cu/secciones/eco_enemar_07.htm#1.

16. AngelFire. [En línea] [Citado el: 13 de 2 de 2008.] http://www.angelfire.com/home/baan/compare.htm.

17. **Krauss, Roberto.** Revista Ingeniería Informática. [En línea] 2004. [Citado el: 12 de 2 de 2008.] http://www.inf.udec.cl/revista.

18. **Salvador Carrasco, Luis.** Luis de Salvador (pagina personal). [En línea] [Citado el: 13 de 2 de 2008.] http://www.luisdesalvador.com/Oposicion/T033_036_Metricas.pdf.

19. programacion.net. *programacion.net.* [En línea] [Citado el: 25 de 2 de 2008.] http://www.programacion.net/java/tutorial/ipintro/4/.

20. JavaHispano.com. *JavaHispano.com.* [En línea] [Citado el: 24 de 2 de 2008.] http://www.javahispano.org/contenidos/es/comparativa_j2ee___net/;jsessionid=C8A5FBCC416CBD77E3D27414B87D6363.

21. Sparxsystems.com. *Sparxsystems.com.* [En línea] [Citado el: 25 de 2 de 2008.] http://www.sparxsystems.com.ar/products/ea.html.

22. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El Proceso Unificado de Software.* La Habana : Félix Varela, 2004. Volumen 1.

23. **S.Pressman, Roger.** *Ingenieria del Software, Un Enfoque Práctico.* La Habana : Félix Varela, 2005. Parte 2.

24. **Ramírez, José.** Maracaibo ,Estado Zulia : Instituto Universitario de Tecnología “READIC”, Enero de 2007.

25. Definicion.de. [En línea] [Citado el: 24 de 5 de 2008.] http://definicion.de/paradigma/.

Anexos:

Anexo 1. Descripción del proceso de negocio para Crear o Actualizar la ficha de un Bien en el almacén

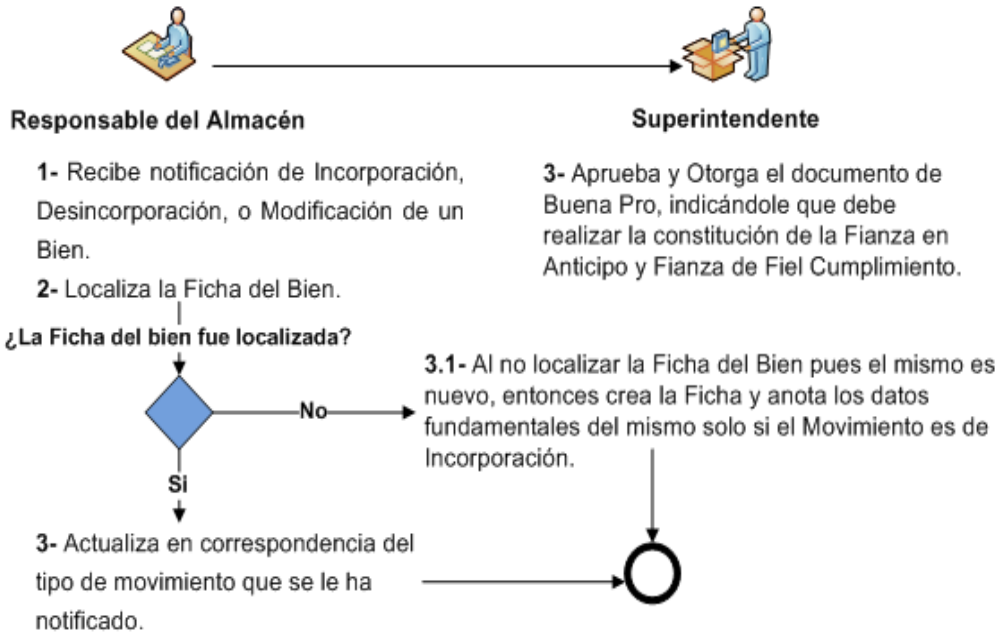


Figura 25 Crear/Actualizar la ficha de un Bien en el almacén

Anexo 2. Descripción del proceso de negocio para la Incorporación de Bienes en el almacén.

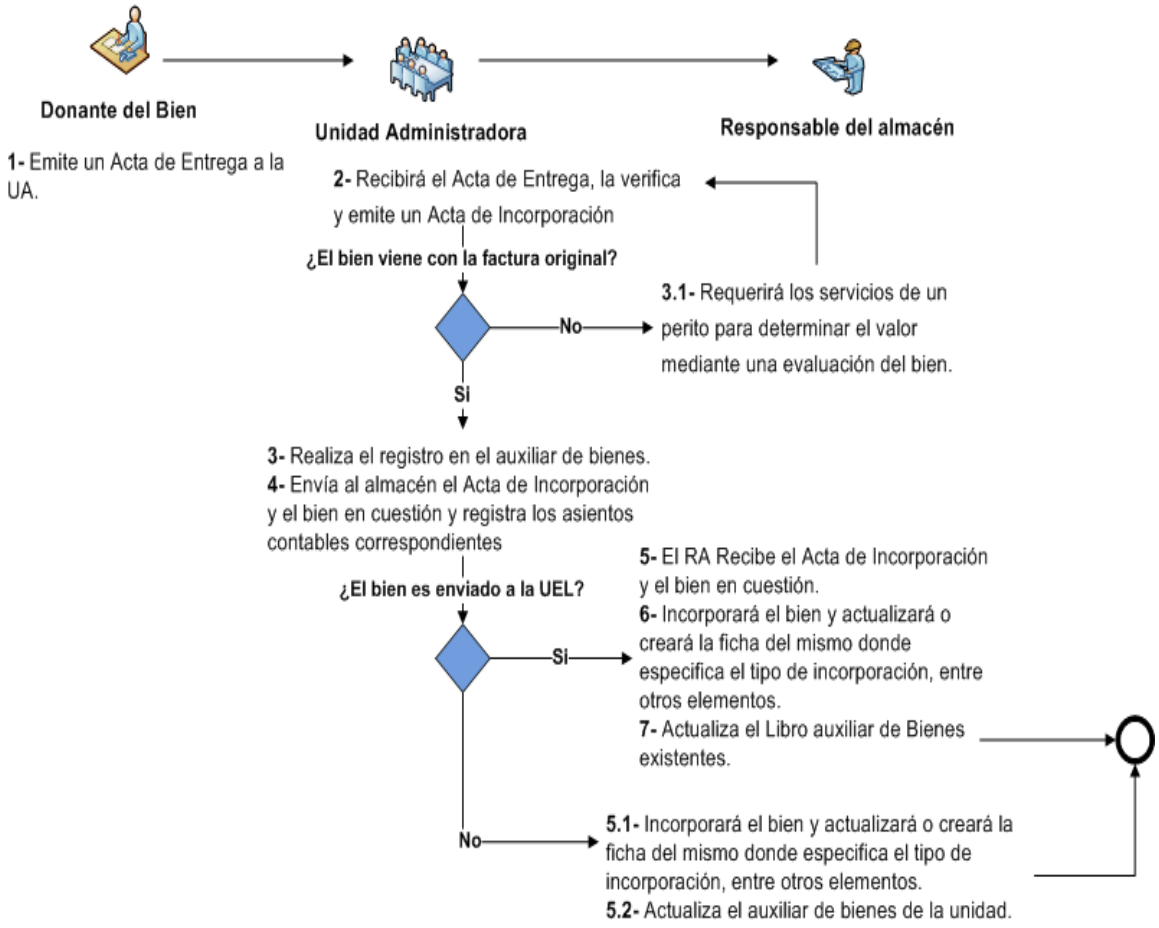


Figura 26 Incorporación de Bienes en el almacén

Anexo 3. Descripción del proceso de negocio para la Desincorporación de Bienes en el almacén.

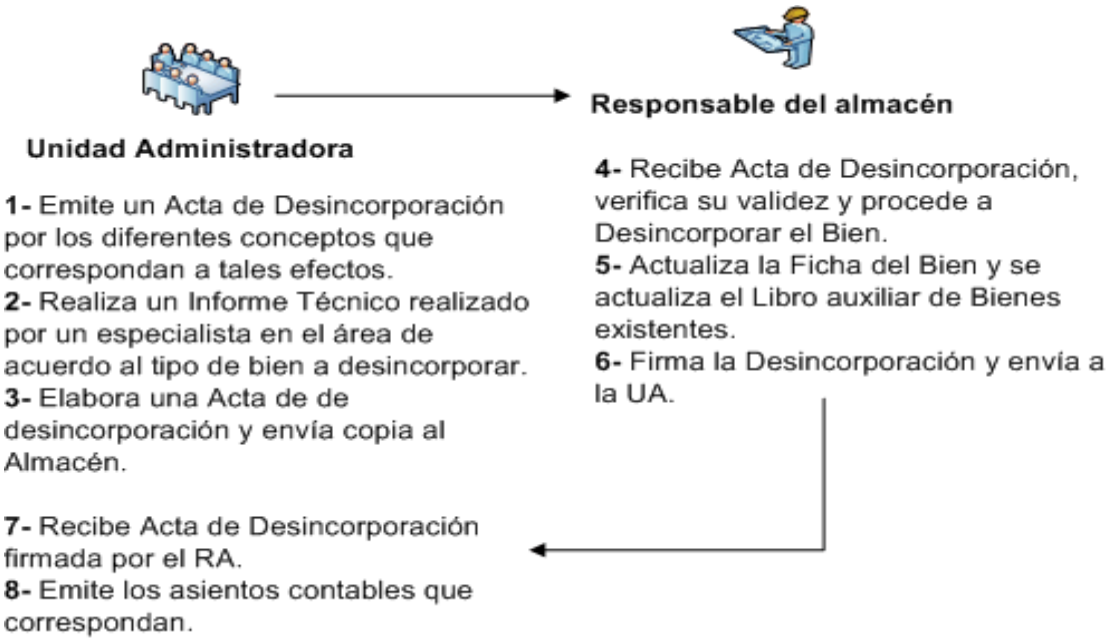


Figura 27 Desincorporación de Bienes en el almacén

Anexo 4. Descripción del proceso de negocio el Inventario Físico de los Bienes en el almacén

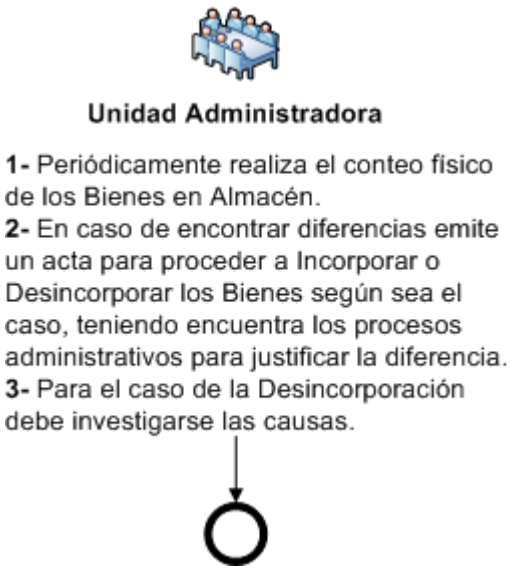
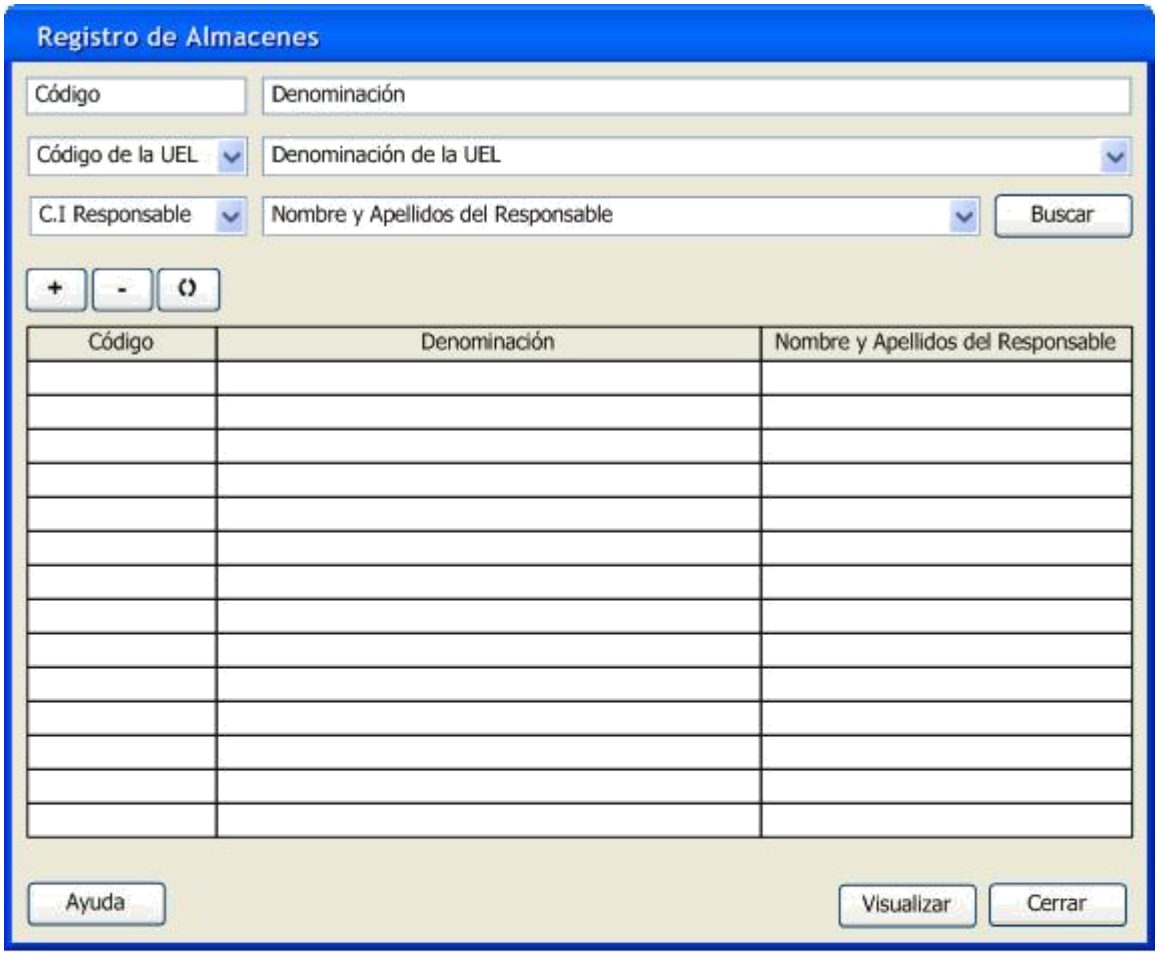
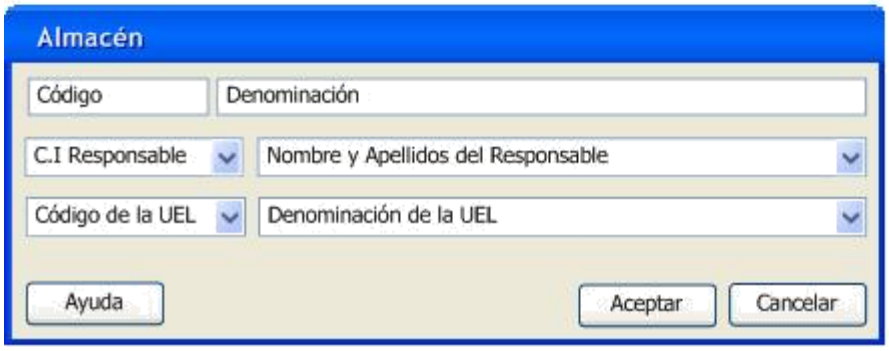


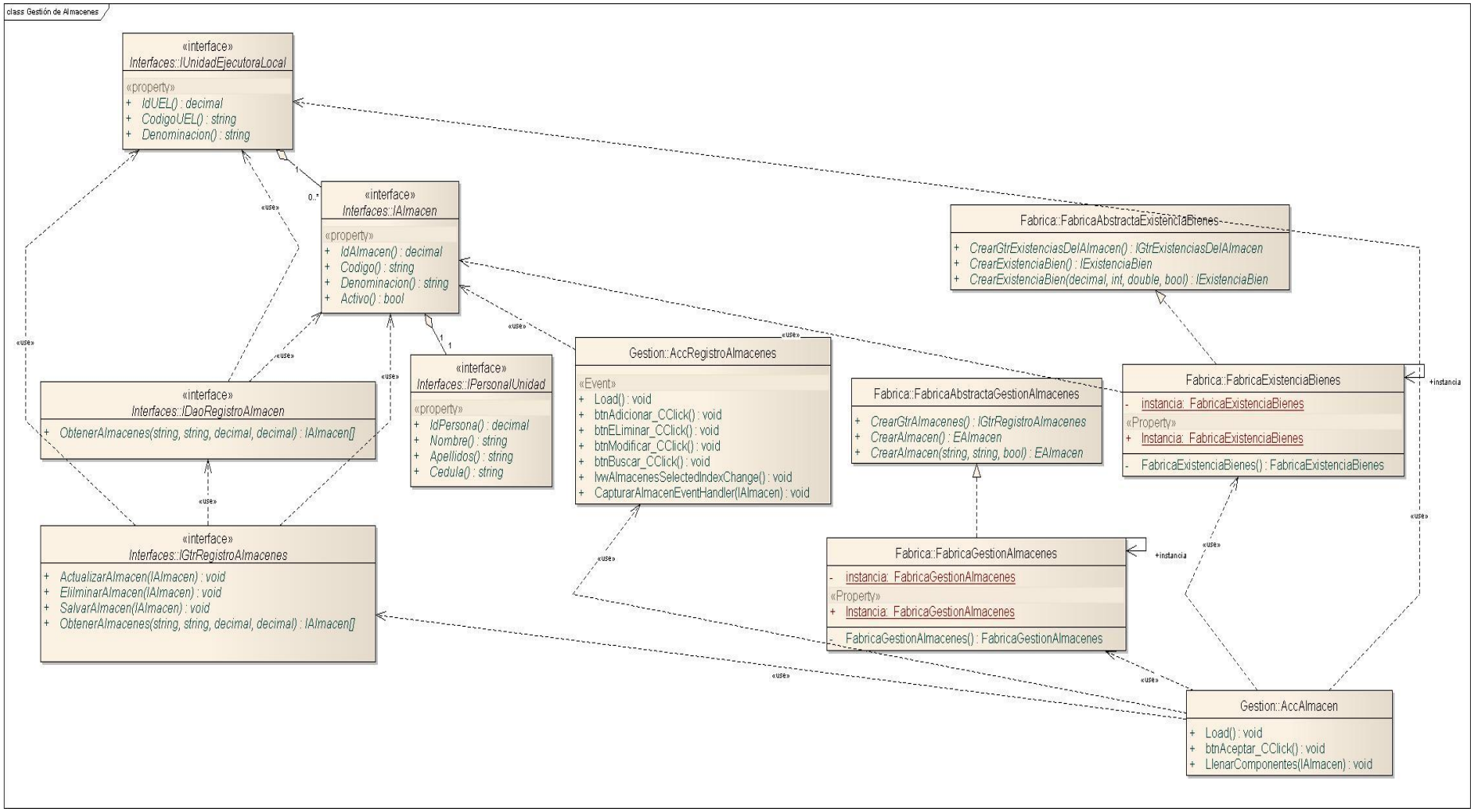
Figura 28 Inventario Físico de los Bienes en el almacén.

Anexo 5. RCU: Gestionar Almacenes.

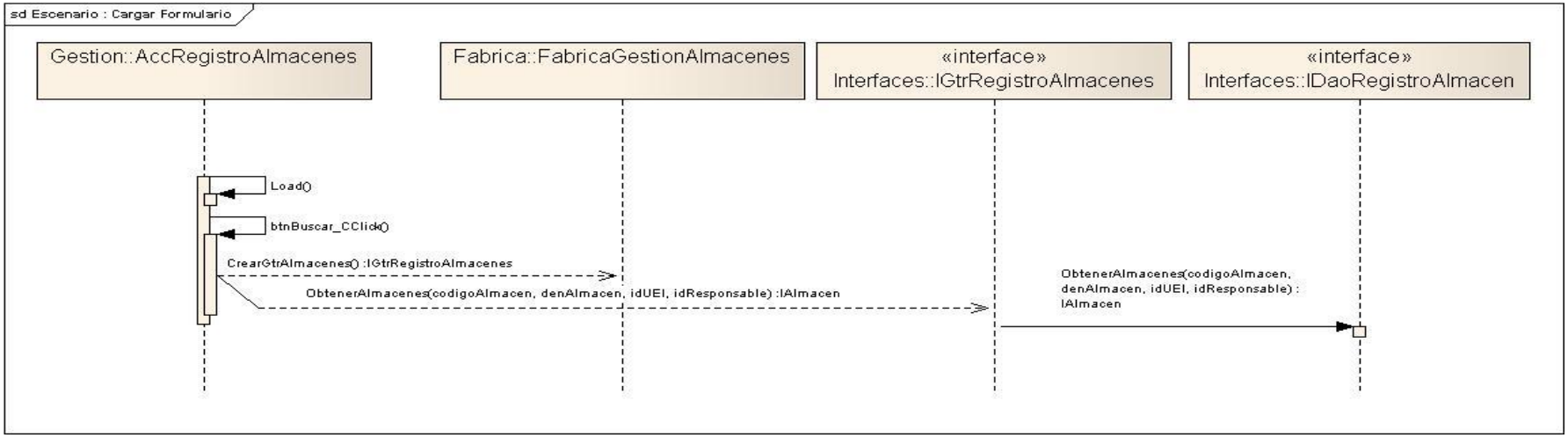
Prototipos de Interfaz de usuario, diagrama de Clases del diseño y diagrama de Interacción para el CU Registro de Almacenes. Este CU posibilita incorporar, modificar y eliminar almacenes al sistema.

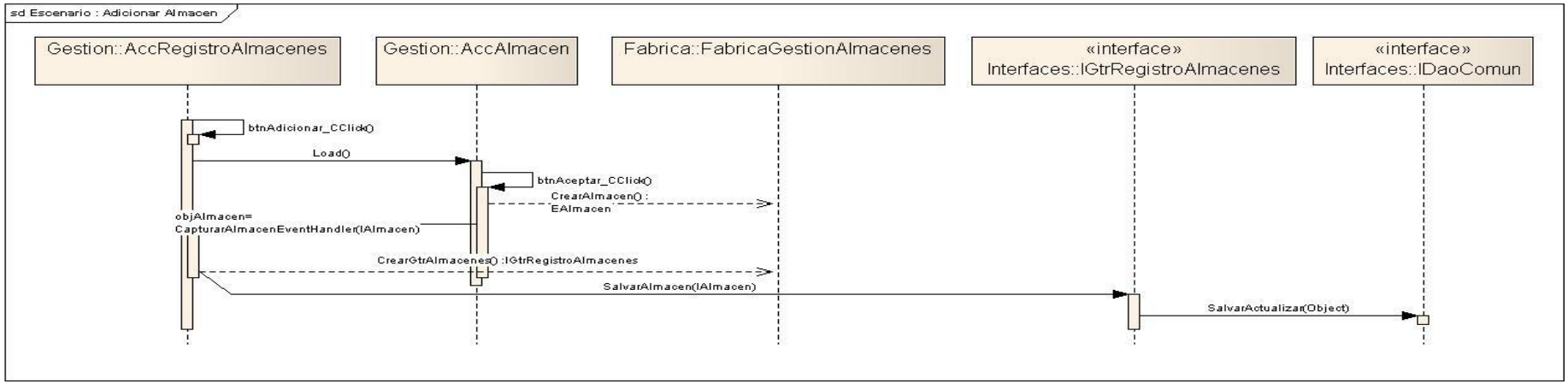


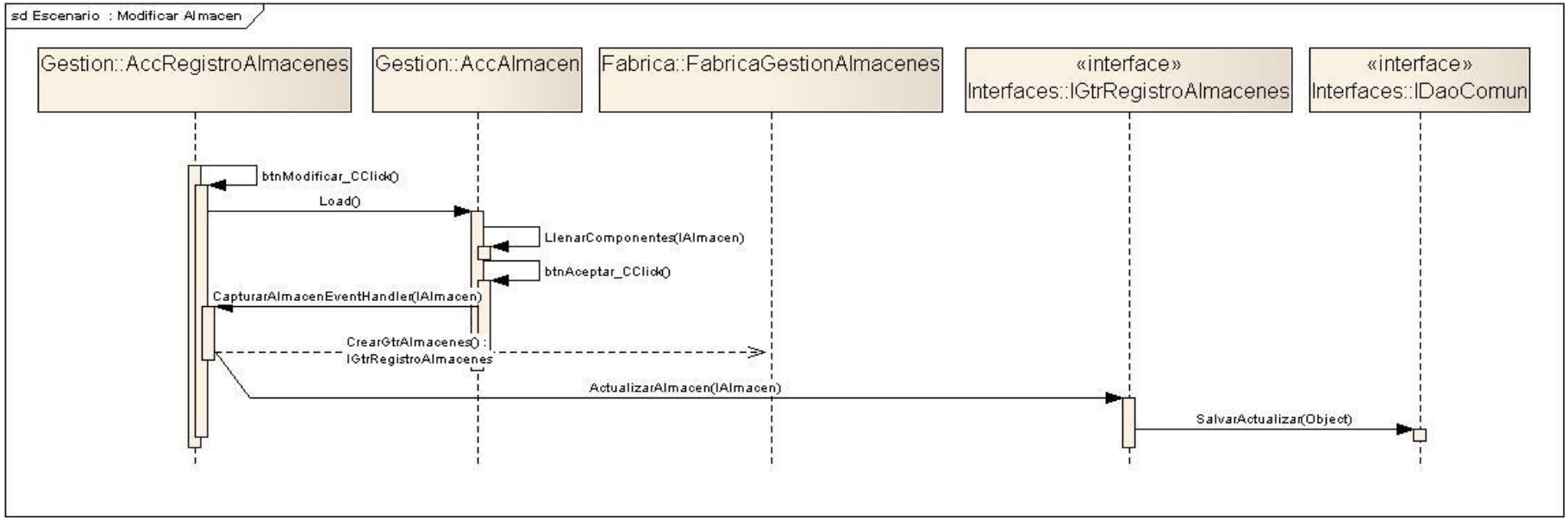


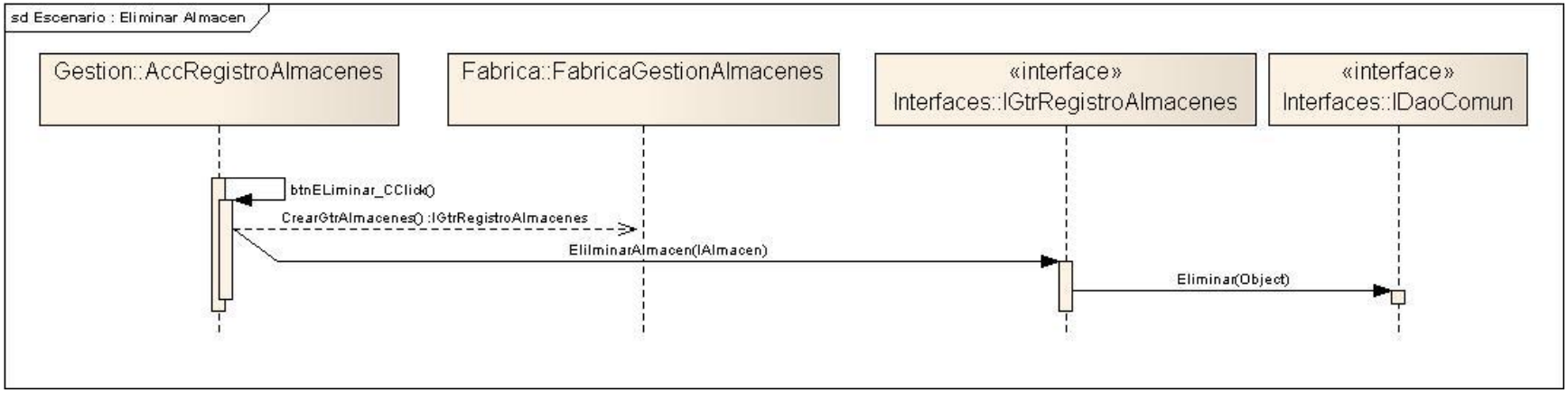

class Gestión de Almacenes
«interface»
Interfaces::IUnidadEjecutoraLocal
«property»
+ IdUEL() : decimal
+ CodigoUEL() : string
+ Denominacion() : string
«interface»
Interfaces::IAlmacen
«property»
+ IdAlmacen() : decimal
+ Codigo() : string
+ Denominacion() : string
+ Activo() : bool
Fabrica::FabricaAbstractaExistenciaBienes
+ CrearGtrExistenciasDelAlmacen() : IGtrExistenciasDelAlmacen
+ CrearExistenciaBien() : IExistenciaBien
+ CrearExistenciaBien(decimal, int, double, bool) : IExistenciaBien
«interface»
Interfaces::IDaoRegistroAlmacen
+ ObtenerAlmacenes(string, string, decimal, decimal) : IAlmacen[]
«interface»
Interfaces::IPersonalUnidad
«property»
+ IdPersona() : decimal
+ Nombre() : string
+ Apellidos() : string
+ Cedula() : string
Gestion::AccRegistroAlmacenes
«Event»
+ Load() : void
+ btnAdicionar_CClick() : void
+ btnELiminar_CClick() : void
+ btnModificar_CClick() : void
+ btnBuscar_CClick() : void
+ lvwAlmacenesSelectedIndexChange() : void
+ CapturarAlmacenEventHandler(IAlmacen) : void
Fabrica::FabricaAbstractaGestionAlmacenes
+ CrearGtrAlmacenes() : IGtrRegistroAlmacenes
+ CrearAlmacen() : EAlmacen
+ CrearAlmacen(string, string, bool) : EAlmacen
Fabrica::FabricaExistenciaBienes
- instancia: FabricaExistenciaBienes
«Property»
+ Instancia: FabricaExistenciaBienes
- FabricaExistenciaBienes() : FabricaExistenciaBienes
«interface»
Interfaces::IGtrRegistroAlmacenes
+ ActualizarAlmacen(IAlmacen) : void
+ EliminarAlmacen(IAlmacen) : void
+ SalvarAlmacen(IAlmacen) : void
+ ObtenerAlmacenes(string, string, decimal, decimal) : IAlmacen[]
Fabrica::FabricaGestionAlmacenes
- instancia: FabricaGestionAlmacenes
«Property»
+ Instancia: FabricaGestionAlmacenes
- FabricaGestionAlmacenes() : FabricaGestionAlmacenes
Gestion::AccAlmacen
+ Load() : void
+ btnAceptar_CClick() : void
+ LlenarComponentes(IAlmacen) : void
«use»
+instancia
1
0..*

A continuación se presentan los diagramas de secuencia correspondientes al CU: Gestionar Almacenes. Estos se muestran divididos por escenarios para facilitar su comprensión.



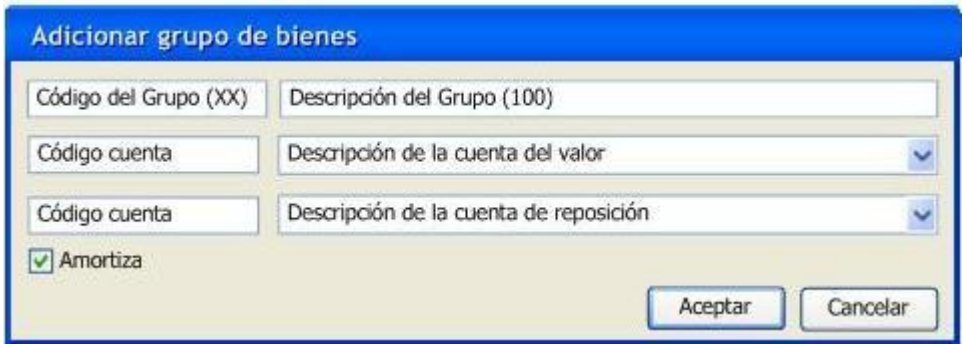

sd Escenario : Adicionar Almacen
Gestion::AccRegistroAlmacenes
Gestion::AccAlmacen
Fabrica::FabricaGestionAlmacenes
«interface»
Interfaces::IGtrRegistroAlmacenes
«interface»
Interfaces::IDaoComun
btnAdicionar_CClick()
Load()
btnAceptar_CClick()
CrearAlmacen() :
EAlmacen
objAlmacen=
CapturarAlmacenEventHandler(IAlmacen)
CrearGtrAlmacenes() :IGtrRegistroAlmacenes
SalvarAlmacen(IAlmacen)
SalvarActualizar(Object)

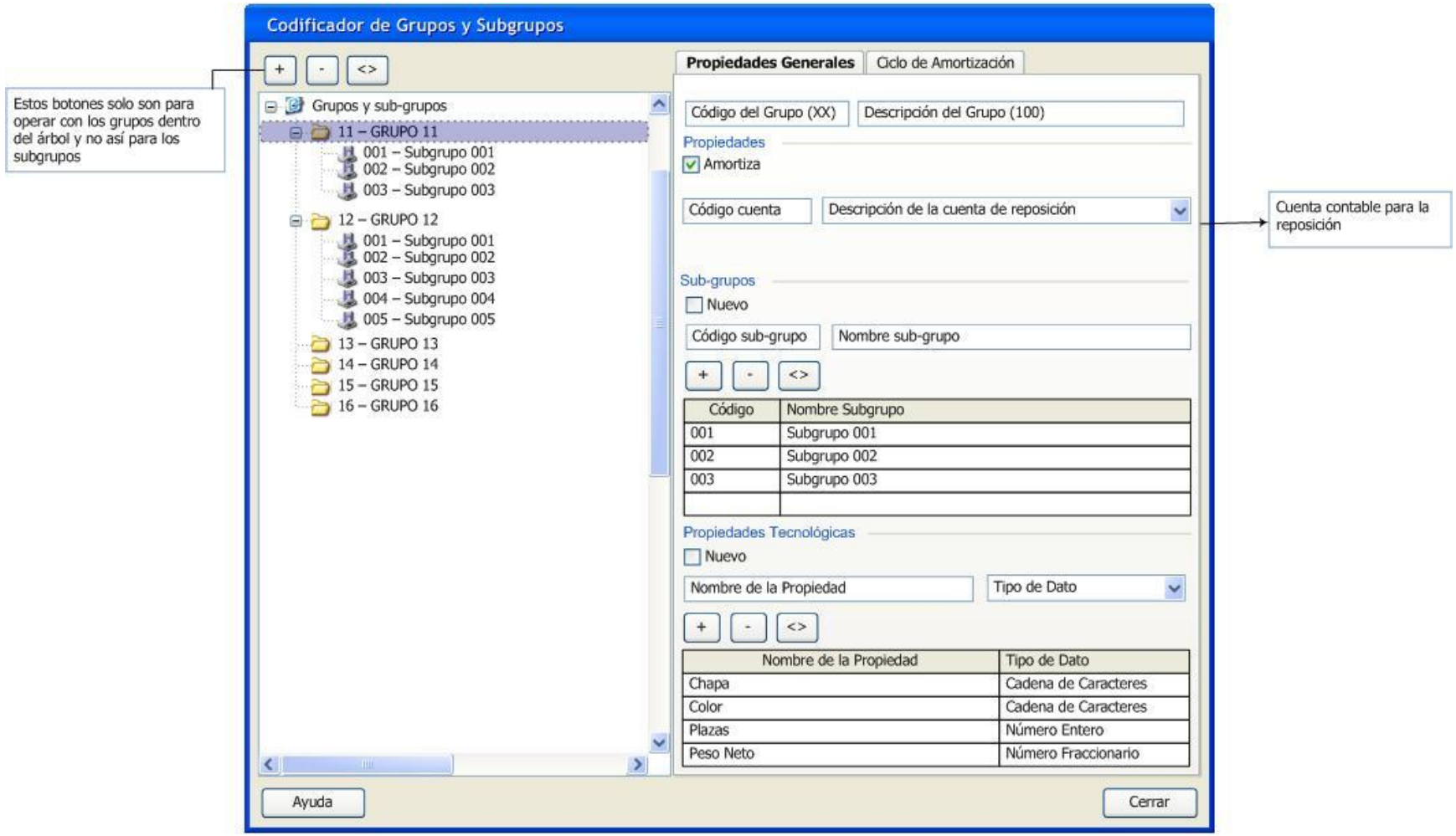

sd Escenario : Modificar Almacen
Gestion::AccRegistroAlmacenes
Gestion::AccAlmacen
Fabrica::FabricaGestionAlmacenes
«interface»
Interfaces::IGtrRegistroAlmacenes
«interface»
Interfaces::IDaoComun
btnModificar_CClick()
Load()
LlenarComponentes(IAlmacen)
btnAceptar_CClick()
CapturarAlmacenEventHandler(IAlmacen)
CrearGtrAlmacenes() :
IGtrRegistroAlmacenes
ActualizarAlmacen(IAlmacen)
SalvarActualizar(Object)

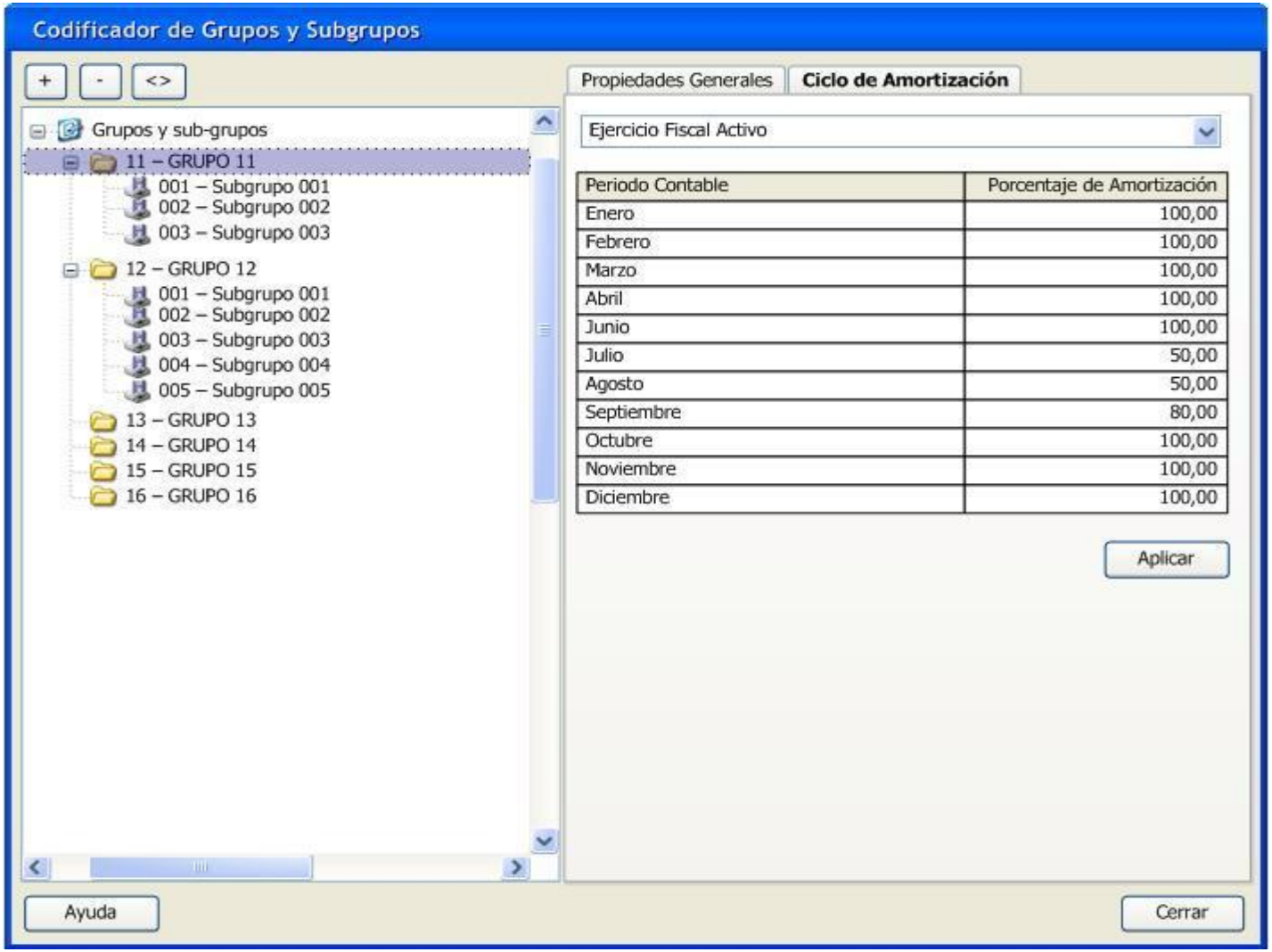


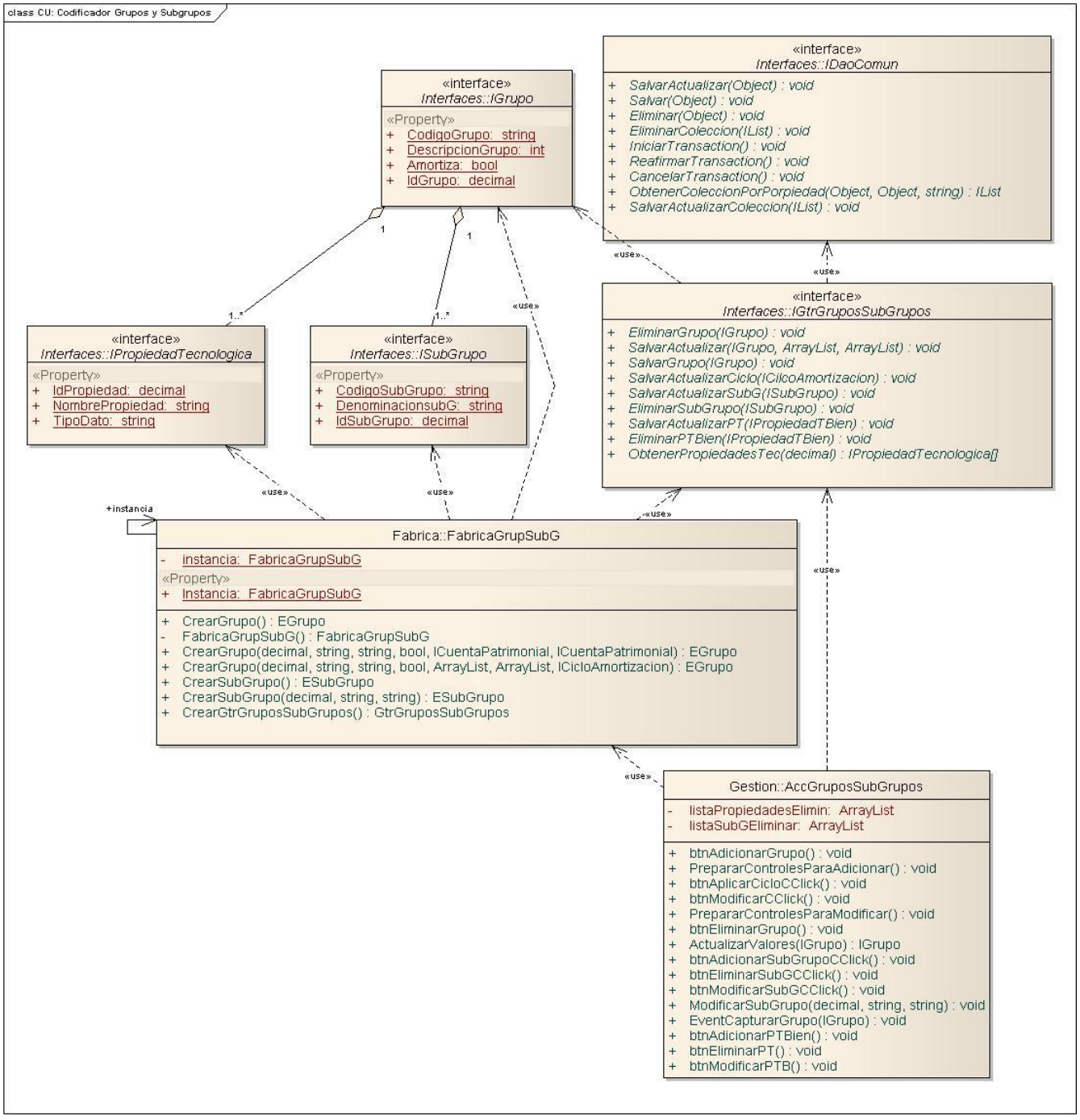
Anexo 6. RCU: Gestionar Grupos Y Subgrupos

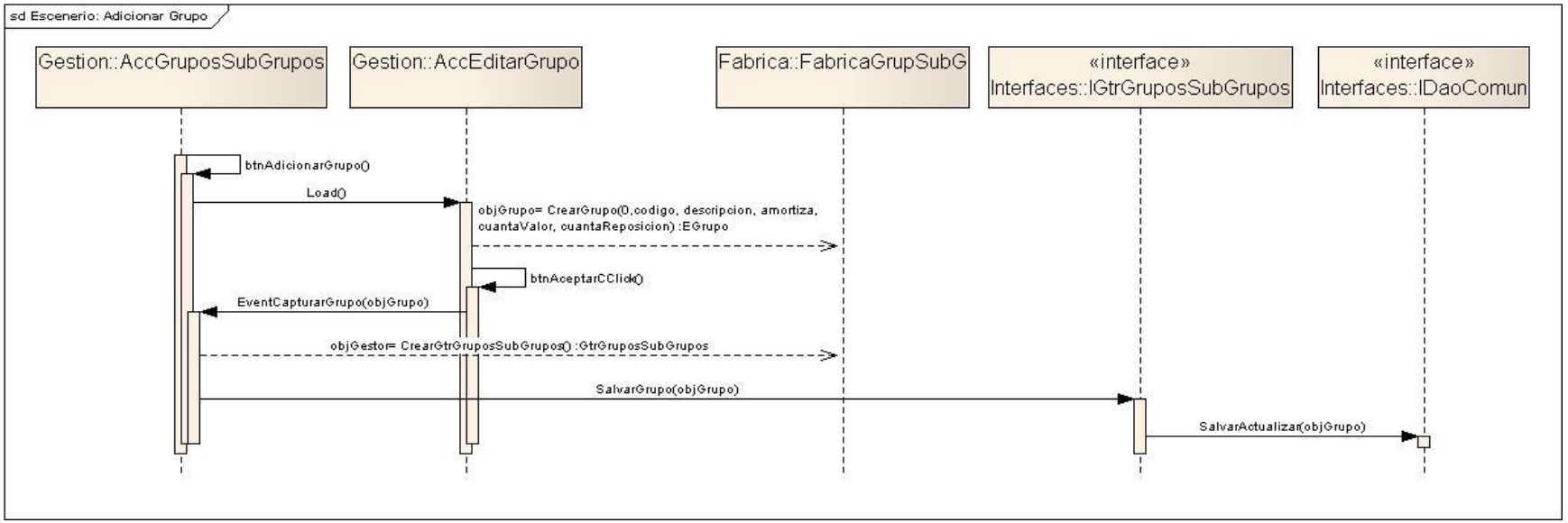
Prototipos de Interfaz de usuario, diagrama de Clases del diseño y diagrama de Interacción para el CU: Gestionar Grupos y Subgrupos. Este CU posibilita crear, modificar y eliminar grupos y subgrupos del sistema, es de gran importancia ya que estos grupos y subgrupos poseen características que definirán el comportamiento de los bienes que le sean asociados.

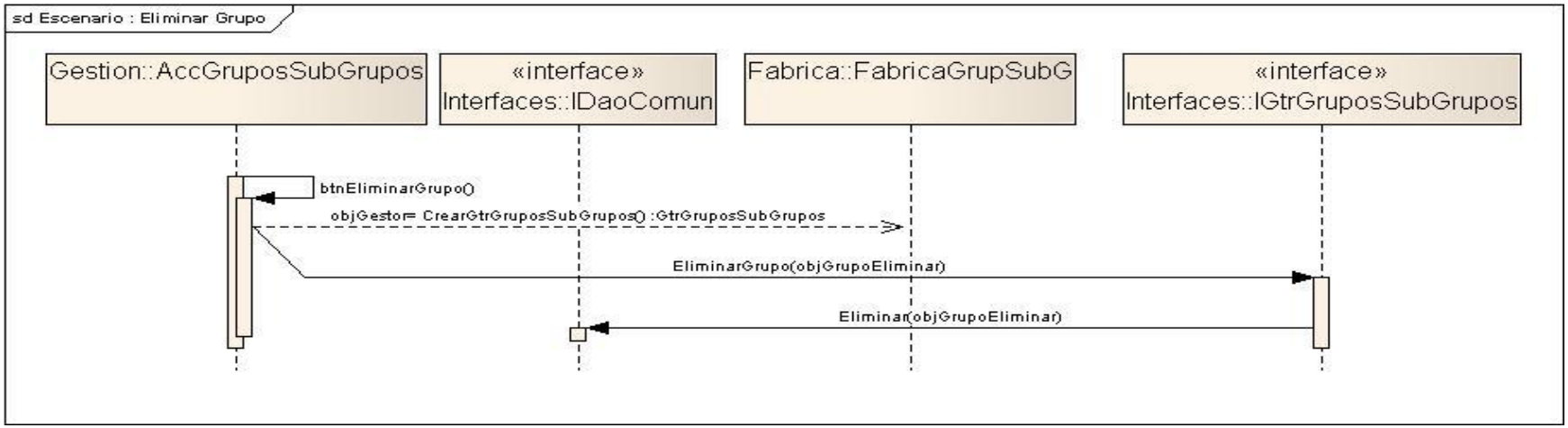


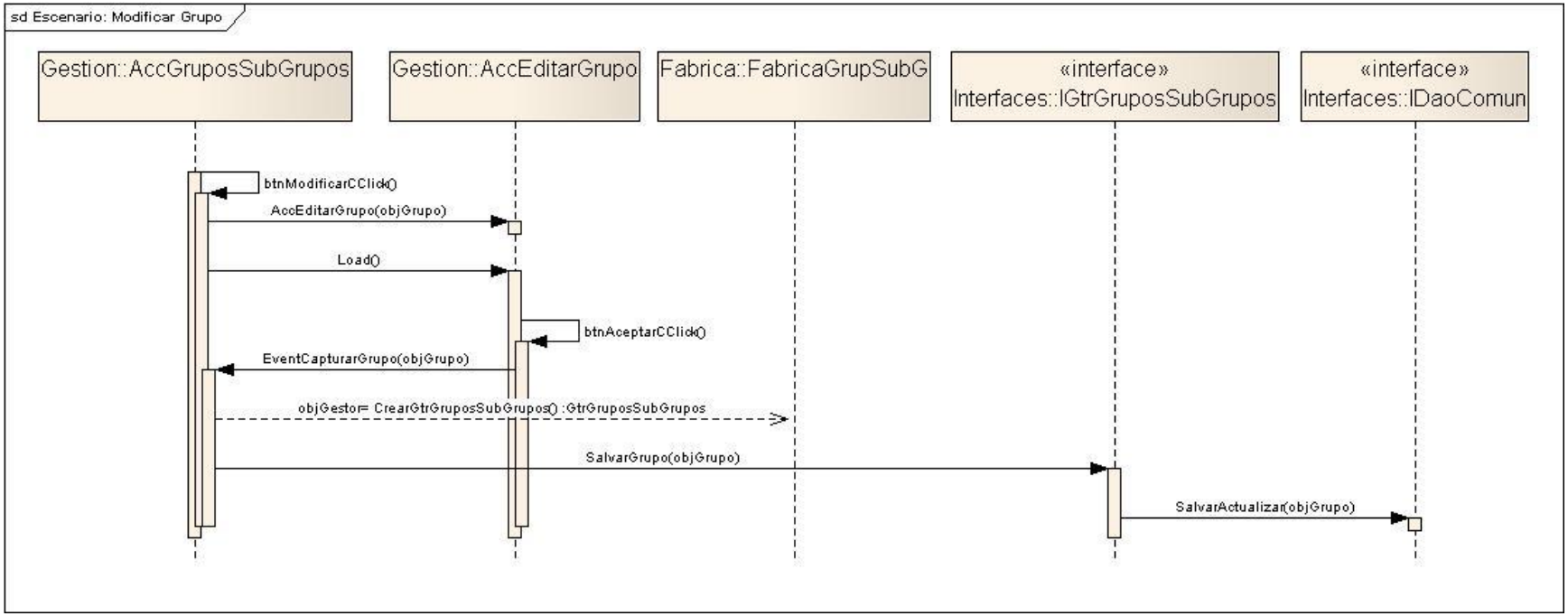

Codificador de Grupos y Subgrupos
Estos botones solo son para operar con los grupos dentro del árbol y no así para los subgrupos
Grupos y sub-grupos
11 – GRUPO 11
001 – Subgrupo 001
002 – Subgrupo 002
003 – Subgrupo 003
12 – GRUPO 12
001 – Subgrupo 001
002 – Subgrupo 002
003 – Subgrupo 003
004 – Subgrupo 004
005 – Subgrupo 005
13 – GRUPO 13
14 – GRUPO 14
15 – GRUPO 15
16 – GRUPO 16
Propiedades Generales
Ciclo de Amortización
Código del Grupo (XX)
Descripción del Grupo (100)
Propiedades
Amortiza
Código cuenta
Descripción de la cuenta de reposición
Cuenta contable para la reposición
Sub-grupos
Nuevo
Código sub-grupo
Nombre sub-grupo
Código
Nombre Subgrupo
001
Subgrupo 001
002
Subgrupo 002
003
Subgrupo 003
Propiedades Tecnológicas
Nuevo
Nombre de la Propiedad
Tipo de Dato
Nombre de la Propiedad
Tipo de Dato
Chapa
Cadena de Caracteres
Color
Cadena de Caracteres
Plazas
Número Entero
Peso Neto
Número Fraccionario
Ayuda
Cerrar

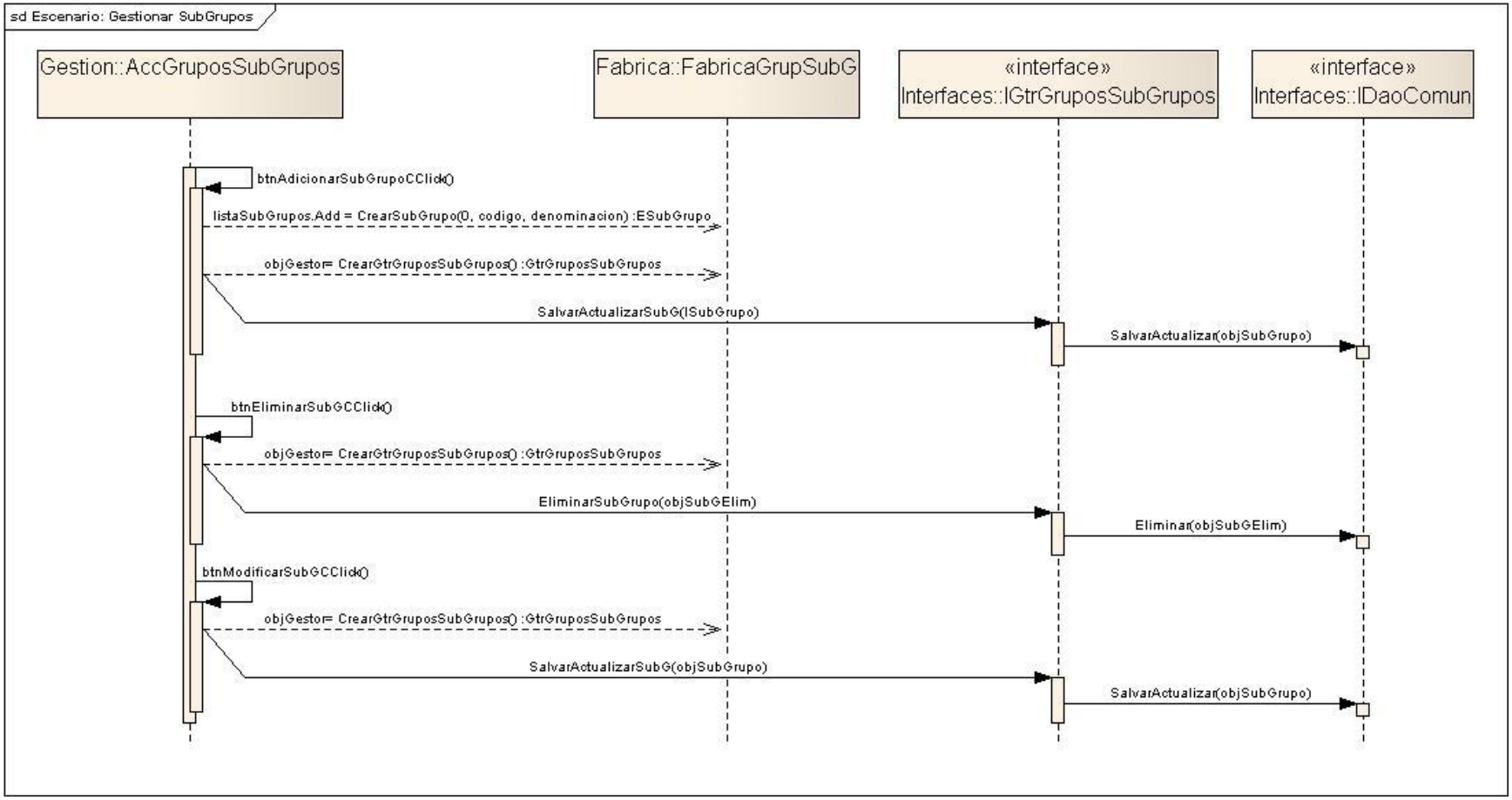

Codificador de Grupos y Subgrupos
+
-
<>
Grupos y sub-grupos
11 – GRUPO 11
001 – Subgrupo 001
002 – Subgrupo 002
003 – Subgrupo 003
12 – GRUPO 12
001 – Subgrupo 001
002 – Subgrupo 002
003 – Subgrupo 003
004 – Subgrupo 004
005 – Subgrupo 005
13 – GRUPO 13
14 – GRUPO 14
15 – GRUPO 15
16 – GRUPO 16
Propiedades Generales
Ciclo de Amortización
Ejercicio Fiscal Activo
Periodo Contable
Porcentaje de Amortización
Enero 100,00
Febrero 100,00
Marzo 100,00
Abril 100,00
Junio 100,00
Julio 50,00
Agosto 50,00
Septiembre 80,00
Octubre 100,00
Noviembre 100,00
Diciembre 100,00
Aplicar
Ayuda
Cerrar

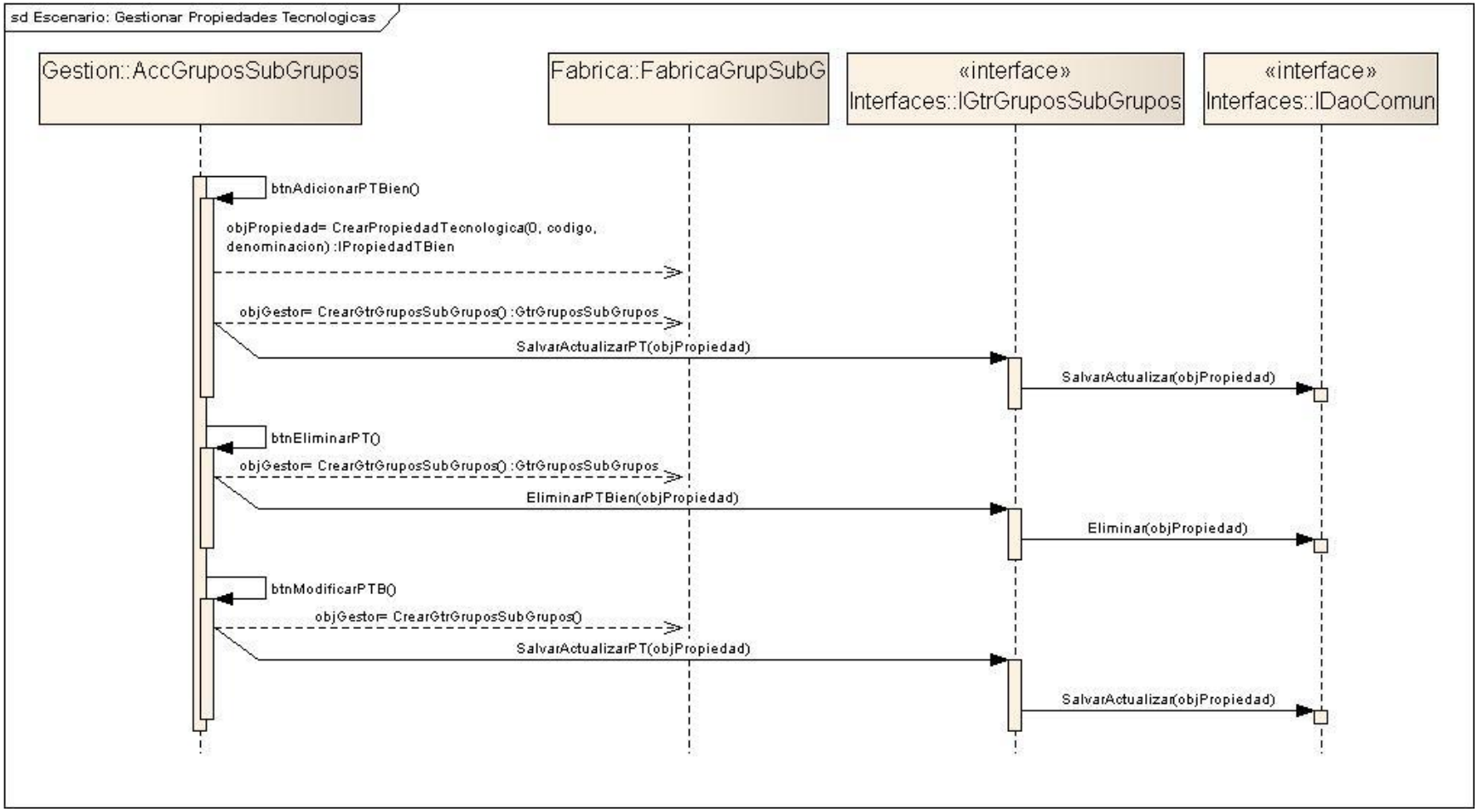

class CU: Codificador Grupos y Subgrupos
«interface»
Interfaces::IGrupo
«Property»
+ CodigoGrupo: string
+ DescripcionGrupo: int
+ Amortiza: bool
+ IdGrupo: decimal
«interface»
Interfaces::IDaoComun
+ SalvarActualizar(Object) : void
+ Salvar(Object) : void
+ Eliminar(Object) : void
+ EliminarColeccion(IList) : void
+ IniciarTransaction() : void
+ ReafirmarTransaction() : void
+ CancelarTransaction() : void
+ ObtenerColeccionPorPorpiedad(Object, Object, string) : IList
+ SalvarActualizarColeccion(IList) : void
«interface»
Interfaces::IPropiedadTecnologica
«Property»
+ IdPropiedad: decimal
+ NombrePropiedad: string
+ TipoDato: string
«interface»
Interfaces::ISubGrupo
«Property»
+ CodigoSubGrupo: string
+ DenominacionsubG: string
+ IdSubGrupo: decimal
«interface»
Interfaces::IGtrGruposSubGrupos
+ EliminarGrupo(IGrupo) : void
+ SalvarActualizar(IGrupo, ArrayList, ArrayList) : void
+ SalvarGrupo(IGrupo) : void
+ SalvarActualizarCiclo(ICilcoAmortizacion) : void
+ SalvarActualizarSubG(ISubGrupo) : void
+ EliminarSubGrupo(ISubGrupo) : void
+ SalvarActualizarPT(IPropiedadTBien) : void
+ EliminarPTBien(IPropiedadTBien) : void
+ ObtenerPropiedadesTec(decimal) : IPropiedadTecnologica[]
Fabrica::FabricaGrupSubG
- instancia: FabricaGrupSubG
«Property»
+ Instancia: FabricaGrupSubG
+ CrearGrupo() : EGrupo
- FabricaGrupSubG() : FabricaGrupSubG
+ CrearGrupo(decimal, string, string, bool, ICuentaPatrimonial, ICuentaPatrimonial) : EGrupo
+ CrearGrupo(decimal, string, string, bool, ArrayList, ArrayList, ICicloAmortizacion) : EGrupo
+ CrearSubGrupo() : ESubGrupo
+ CrearSubGrupo(decimal, string, string) : ESubGrupo
+ CrearGtrGruposSubGrupos() : GtrGruposSubGrupos
Gestion::AccGruposSubGrupos
- listaPropiedadesElimin: ArrayList
- listaSubGEliminar: ArrayList
+ btnAdicionarGrupo() : void
+ PrepararControlesParaAdicionar() : void
+ btnAplicarCicloCClick() : void
+ btnModificarCClick() : void
+ PrepararControlesParaModificar() : void
+ btnEliminarGrupo() : void
+ ActualizarValores(IGrupo) : IGrupo
+ btnAdicionarSubGrupoCClick() : void
+ btnEliminarSubGCClick() : void
+ btnModificarSubGCClick() : void
+ ModificarSubGrupo(decimal, string, string) : void
+ EventCapturarGrupo(IGrupo) : void
+ btnAdicionarPTBien() : void
+ btnEliminarPT() : void
+ btnModificarPTB() : void
«use»
1
1..*
+instancia

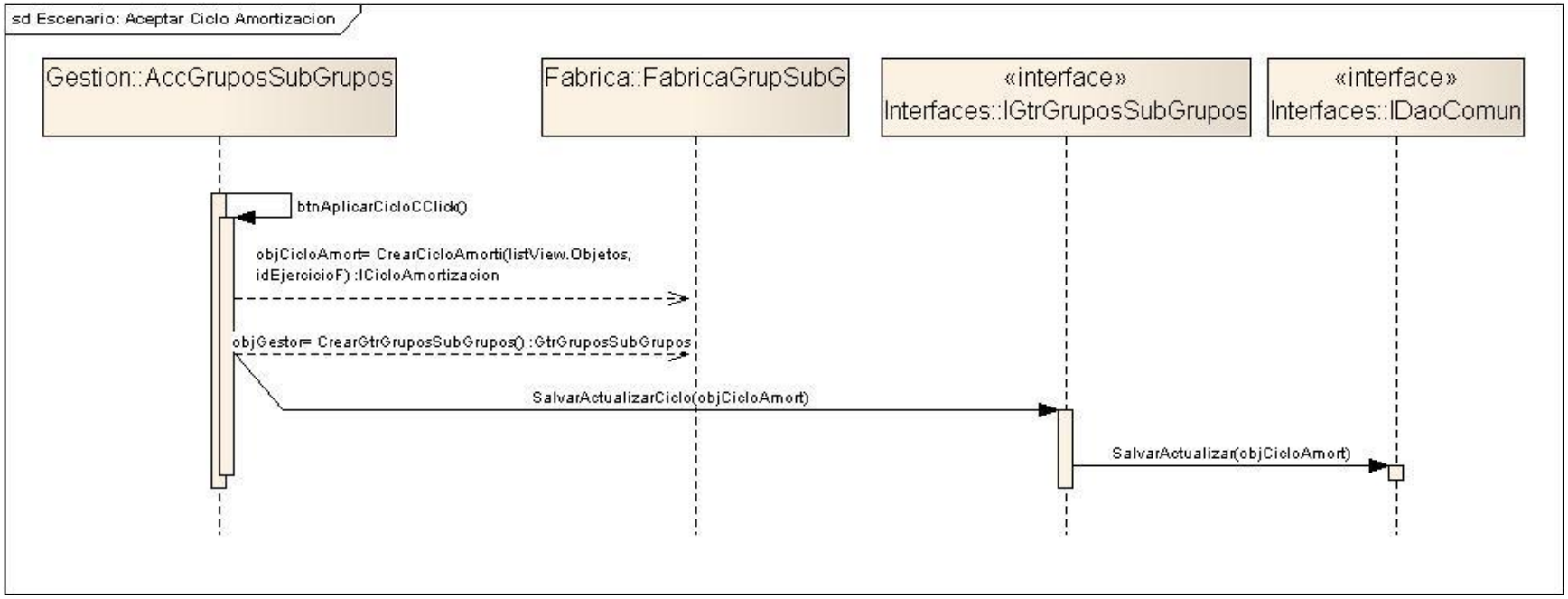

sd Escenerio: Adicionar Grupo
Gestion::AccGruposSubGrupos
Gestion::AccEditarGrupo
Fabrica::FabricaGrupSubG
«interface»
Interfaces::IGtrGruposSubGrupos
«interface»
Interfaces::IDaoComun
btnAdicionarGrupo()
Load()
objGrupo= CrearGrupo(0,codigo, descripcion, amortiza, cuantaValor, cuantaReposicion) :EGrupo
btnAceptarCClick()
EventCapturarGrupo(objGrupo)
objGestor= CrearGtrGruposSubGrupos() :GtrGruposSubGrupos
SalvarGrupo(objGrupo)
SalvarActualizar(objGrupo)


sd Escenario : Eliminar Grupo
Gestion::AccGruposSubGrupos
«interface»
Interfaces::IDaoComun
Fabrica::FabricaGrupSubG
«interface»
Interfaces::IGtrGruposSubGrupos
btnEliminarGrupo()
objGestor= CrearGtrGruposSubGrupos() :GtrGruposSubGrupos
EliminarGrupo(objGrupoEliminar)
Eliminar(objGrupoEliminar)


sd Escenario: Modificar Grupo
Gestion::AccGruposSubGrupos
Gestion::AccEditarGrupo
Fabrica::FabricaGrupSubG
«interface»
Interfaces::IGtrGruposSubGrupos
«interface»
Interfaces::IDaoComun
btnModificarCClick()
AccEditarGrupo(objGrupo)
Load()
btnAceptarCClick()
EventCapturarGrupo(objGrupo)
objGestor= CrearGtrGruposSubGrupos() :GtrGruposSubGrupos
SalvarGrupo(objGrupo)
SalvarActualizar(objGrupo)

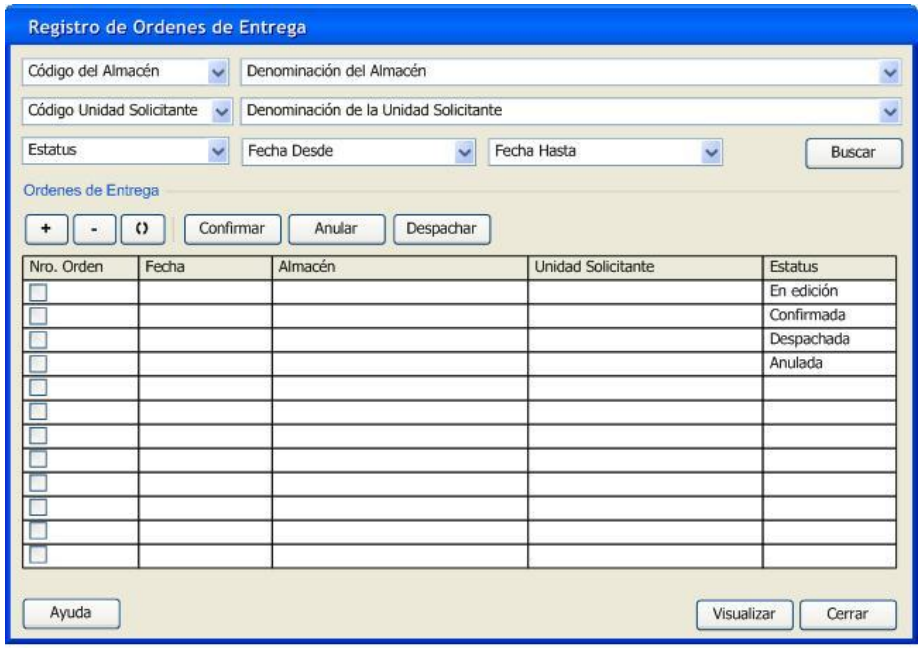

sd Escenario: Gestionar SubGrupos
Gestion::AccGruposSubGrupos
Fabrica::FabricaGrupSubG
«interface»
Interfaces::IGtrGruposSubGrupos
«interface»
Interfaces::IDaoComun
btnAdicionarSubGrupoCClick()
listaSubGrupos.Add = CrearSubGrupo(0, codigo, denominacion) :ESubGrupo
objGestor= CrearGtrGruposSubGrupos() :GtrGruposSubGrupos
SalvarActualizarSubG(ISubGrupo)
SalvarActualizar(objSubGrupo)
btnEliminarSubGCClick()
objGestor= CrearGtrGruposSubGrupos() :GtrGruposSubGrupos
EliminarSubGrupo(objSubGElim)
Eliminar(objSubGElim)
btnModificarSubGCClick()
objGestor= CrearGtrGruposSubGrupos() :GtrGruposSubGrupos
SalvarActualizarSubG(objSubGrupo)
SalvarActualizar(objSubGrupo)

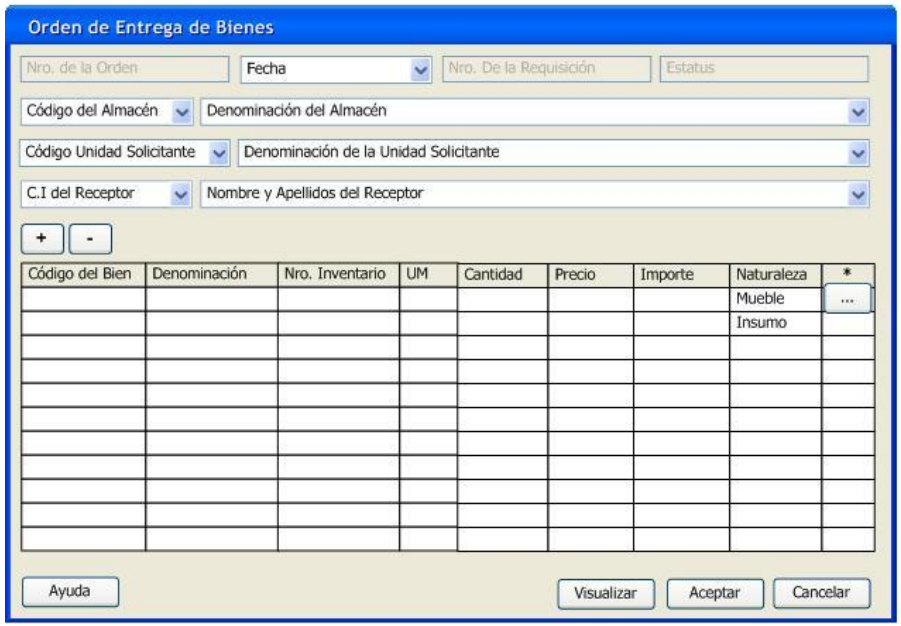

sd Escenario: Gestionar Propiedades Tecnologicas
Gestion::AccGruposSubGrupos
Fabrica::FabricaGrupSubG
«interface»
Interfaces::IGtrGruposSubGrupos
«interface»
Interfaces::IDaoComun
btnAdicionarPTBien()
objPropiedad= CrearPropiedadTecnologica(0, codigo, denominacion) :IPropiedadTBien
objGestor= CrearGtrGruposSubGrupos() :GtrGruposSubGrupos
SalvarActualizarPT(objPropiedad)
SalvarActualizar(objPropiedad)
btnEliminarPT()
objGestor= CrearGtrGruposSubGrupos() :GtrGruposSubGrupos
EliminarPTBien(objPropiedad)
Eliminar(objPropiedad)
btnModificarPTB()
objGestor= CrearGtrGruposSubGrupos()
SalvarActualizarPT(objPropiedad)
SalvarActualizar(objPropiedad)

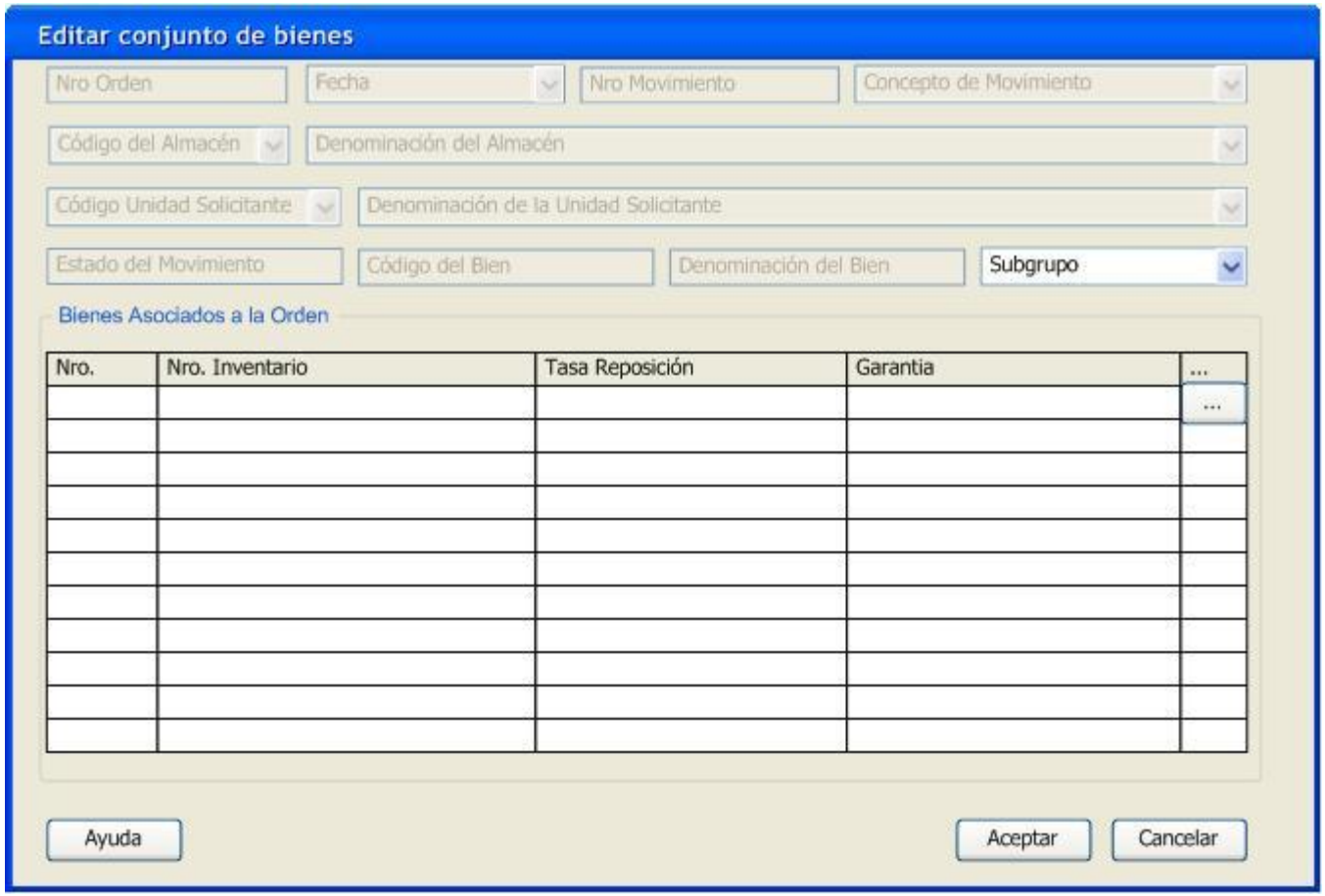


Anexo 7. RCU Registrar Órdenes de Entrega

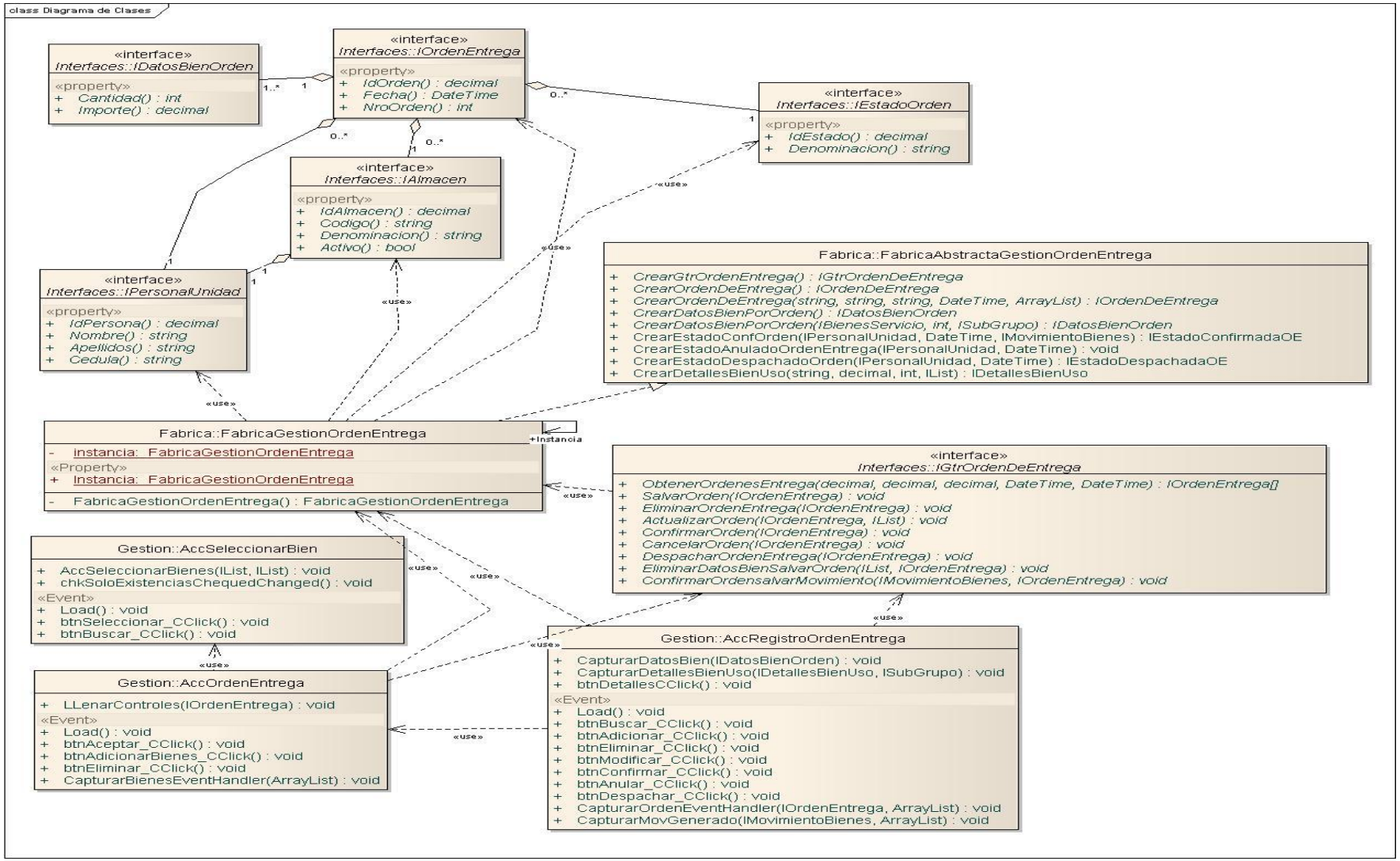
Prototipos de Interfaz de usuario, diagrama de Clases del diseño, diagrama de transición de estados y diagrama de Interacción para el CU: Registrar Órdenes de Entrega este caso de uso permite editar y mantener el control sobre todas las órdenes de entrega generadas. En el presente documento se muestran solo algunos de los escenarios correspondientes al CU, para mayor referencia ver el fichero *Control de Bienes.EAP* adjunto a al trabajo.

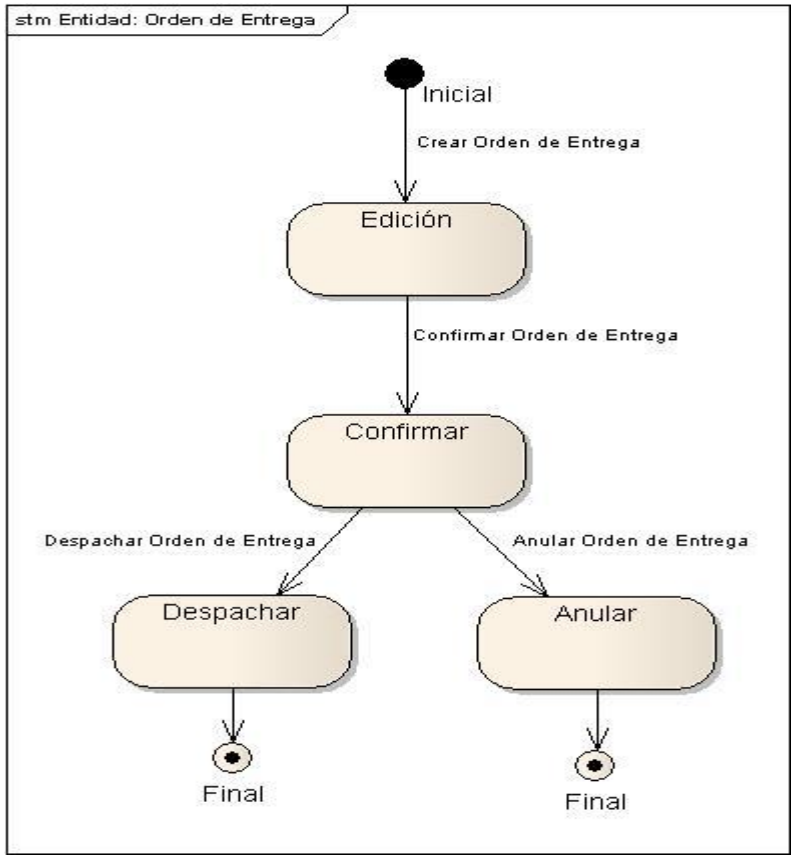

Registro de Ordenes de Entrega
Código del Almacén
Denominación del Almacén
Código Unidad Solicitante
Denominación de la Unidad Solicitante
Estatus
Fecha Desde
Fecha Hasta
Buscar
Ordenes de Entrega
+
-
Confirmar
Anular
Despachar
Nro. Orden
Fecha
Almacén
Unidad Solicitante
Estatus
En edición
Confirmada
Despachada
Anulada
Ayuda
Visualizar
Cerrar

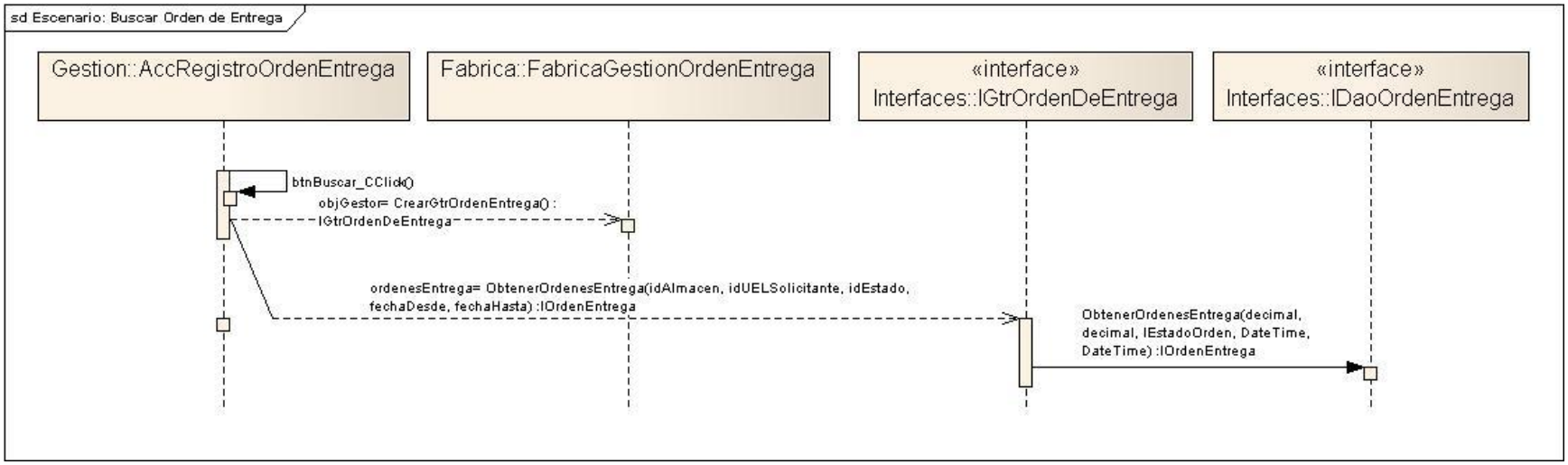

Orden de Entrega de Bienes
Nro. de la Orden
Fecha
Nro. De la Requisición
Estatus
Código del Almacén
Denominación del Almacén
Código Unidad Solicitante
Denominación de la Unidad Solicitante
C.I del Receptor
Nombre y Apellidos del Receptor
+
-
Código del Bien
Denominación
Nro. Inventario
UM
Cantidad
Precio
Importe
Naturaleza
Mueble
Insumo
Ayuda
Visualizar
Aceptar
Cancelar

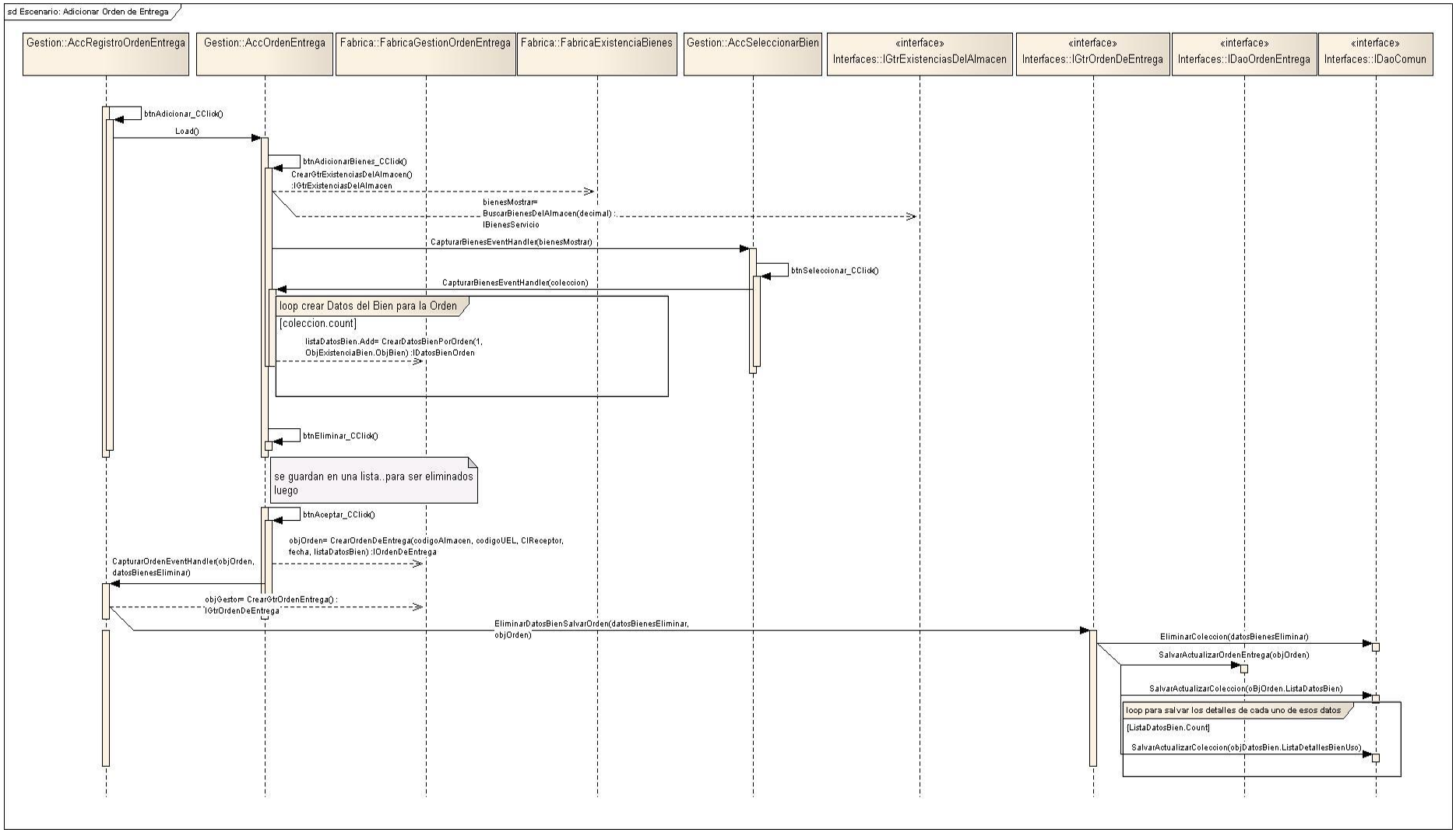

Editar conjunto de bienes
Nro Orden
Fecha
Nro Movimiento
Concepto de Movimiento
Código del Almacén
Denominación del Almacén
Código Unidad Solicitante
Denominación de la Unidad Solicitante
Estado del Movimiento
Código del Bien
Denominación del Bien
Subgrupo
Bienes Asociados a la Orden
Nro.
Nro. Inventario
Tasa Reposición
Garantia
...
Ayuda
Aceptar
Cancelar

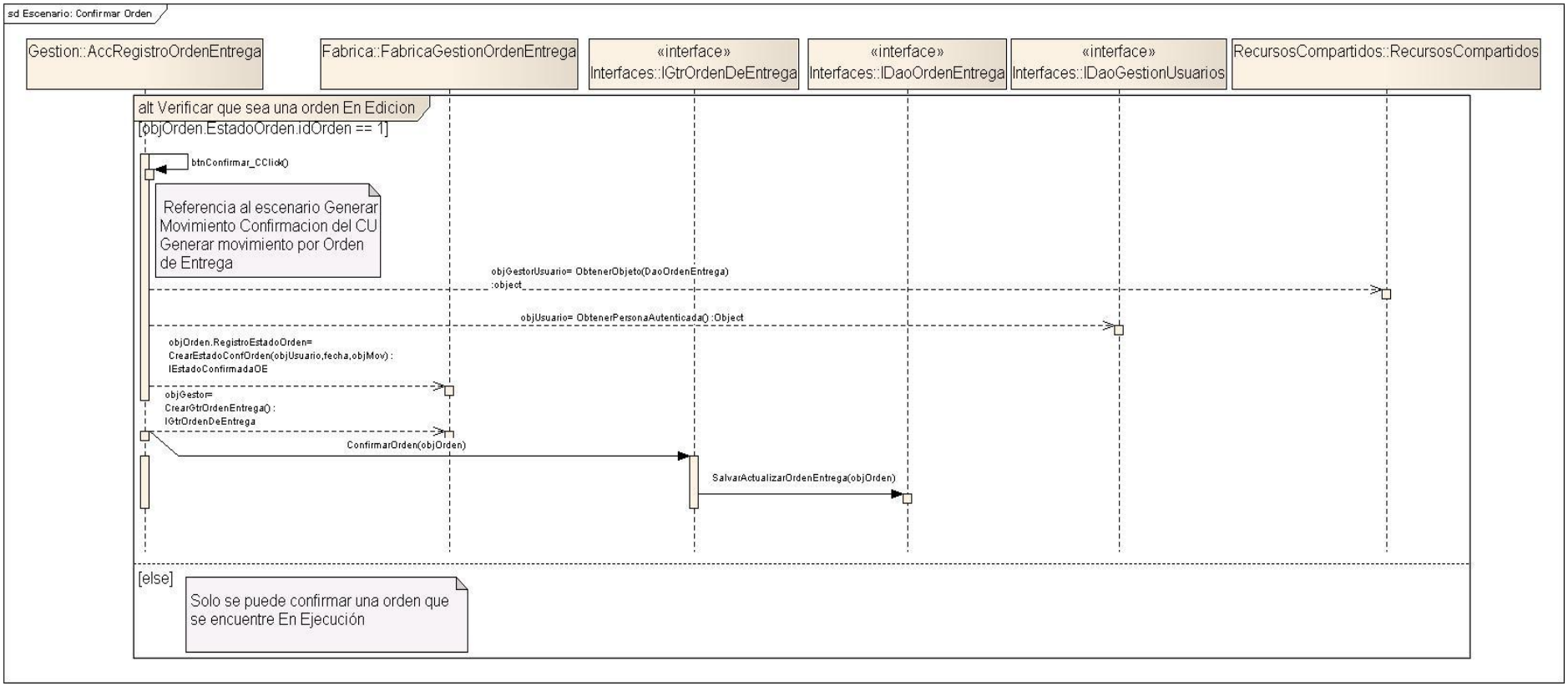
Este formulario es de tipo flotante, y es después el que se emplea para editar en cualquier momento el bien

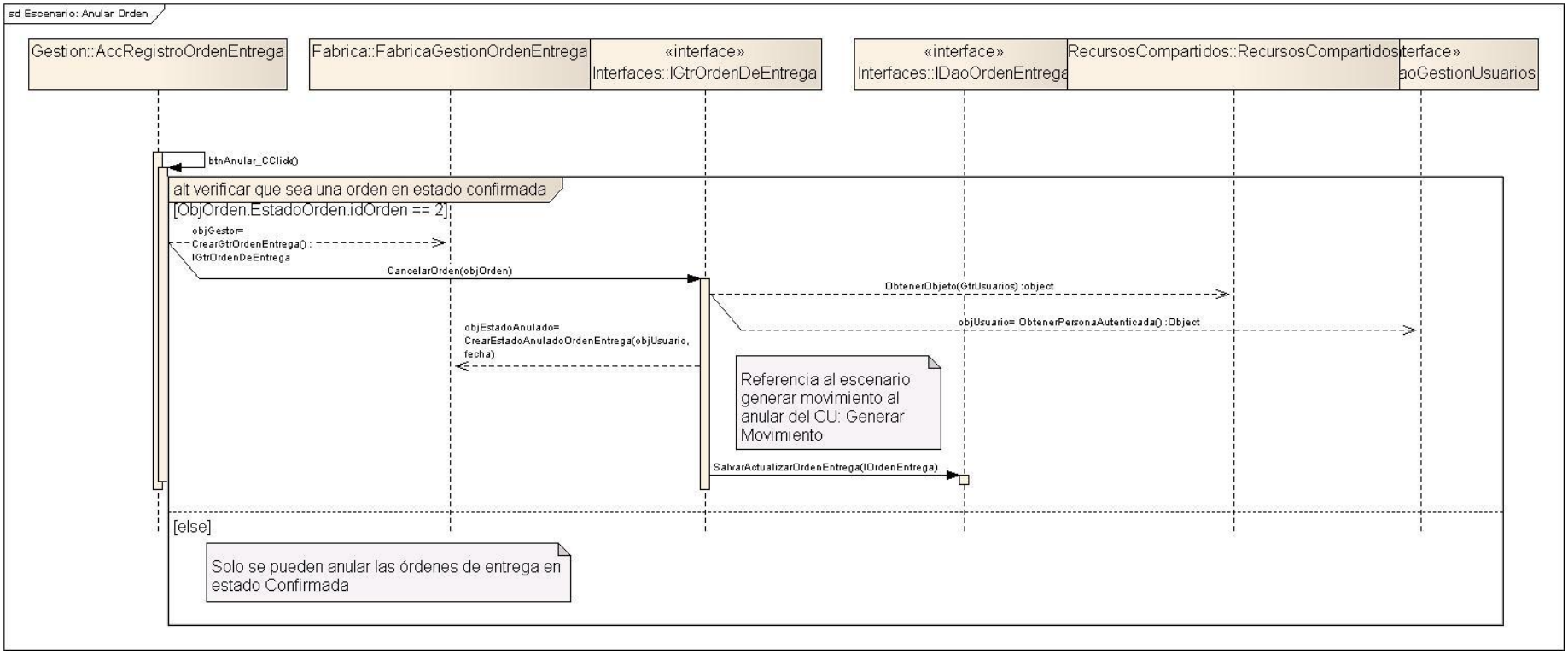

class Diagrama de Clases
«interface»
Interfaces::IDatosBienOrden
«property»
+ Cantidad() : int
+ Importe() : decimal
«interface»
Interfaces::IOrdenEntrega
«property»
+ IdOrden() : decimal
+ Fecha() : DateTime
+ NroOrden() : int
«interface»
Interfaces::IEstadoOrden
«property»
+ IdEstado() : decimal
+ Denominacion() : string
«interface»
Interfaces::IAlmacen
«property»
+ IdAlmacen() : decimal
+ Codigo() : string
+ Denominacion() : string
+ Activo() : bool
«interface»
Interfaces::IPersonalUnidad
«property»
+ IdPersona() : decimal
+ Nombre() : string
+ Apellidos() : string
+ Cedula() : string
Fabrica::FabricaAbstractaGestionOrdenEntrega
+ CrearGtrOrdenEntrega() : IGtrOrdenDeEntrega
+ CrearOrdenDeEntrega() : IOrdenDeEntrega
+ CrearOrdenDeEntrega(string, string, string, DateTime, ArrayList) : IOrdenDeEntrega
+ CrearDatosBienPorOrden() : IDatosBienOrden
+ CrearDatosBienPorOrden(IBienesServicio, int, ISubGrupo) : IDatosBienOrden
+ CrearEstadoConfOrden(IPersonalUnidad, DateTime, IMovimientoBienes) : IEstadoConfirmadaOE
+ CrearEstadoAnuladoOrdenEntrega(IPersonalUnidad, DateTime) : void
+ CrearEstadoDespachadoOrden(IPersonalUnidad, DateTime) : IEstadoDespachadaOE
+ CrearDetallesBienUso(string, decimal, int, IList) : IDetallesBienUso
Fabrica::FabricaGestionOrdenEntrega
- instancia: FabricaGestionOrdenEntrega
«Property»
+ Instancia: FabricaGestionOrdenEntrega
- FabricaGestionOrdenEntrega() : FabricaGestionOrdenEntrega
+Instancia
«interface»
Interfaces::IGtrOrdenDeEntrega
+ ObtenerOrdenesEntrega(decimal, decimal, decimal, DateTime, DateTime) : IOrdenEntrega[]
+ SalvarOrden(IOrdenEntrega) : void
+ EliminarOrdenEntrega(IOrdenEntrega) : void
+ ActualizarOrden(IOrdenEntrega, IList) : void
+ ConfirmarOrden(IOrdenEntrega) : void
+ CancelarOrden(IOrdenEntrega) : void
+ DespacharOrdenEntrega(IOrdenEntrega) : void
+ EliminarDatosBienSalvarOrden(IList, IOrdenEntrega) : void
+ ConfirmarOrdensalvarMovimiento(IMovimientoBienes, IOrdenEntrega) : void
Gestion::AccSeleccionarBien
+ AccSeleccionarBienes(IList, IList) : void
+ chkSoloExistenciasChequedChanged() : void
«Event»
+ Load() : void
+ btnSeleccionar_CClick() : void
+ btnBuscar_CClick() : void
Gestion::AccOrdenEntrega
+ LLenarControles(IOrdenEntrega) : void
«Event»
+ Load() : void
+ btnAceptar_CClick() : void
+ btnAdicionarBienes_CClick() : void
+ btnEliminar_CClick() : void
+ CapturarBienesEventHandler(ArrayList) : void
Gestion::AccRegistroOrdenEntrega
+ CapturarDatosBien(IDatosBienOrden) : void
+ CapturarDetallesBienUso(IDetallesBienUso, ISubGrupo) : void
+ btnDetallesCClick() : void
«Event»
+ Load() : void
+ btnBuscar_CClick() : void
+ btnAdicionar_CClick() : void
+ btnEliminar_CClick() : void
+ btnModificar_CClick() : void
+ btnConfirmar_CClick() : void
+ btnAnular_CClick() : void
+ btnDespachar_CClick() : void
+ CapturarOrdenEventHandler(IOrdenEntrega, ArrayList) : void
+ CapturarMovGenerado(IMovimientoBienes, ArrayList) : void
«use»


stm Entidad: Orden de Entrega
Inicial
Crear Orden de Entrega
Edición
Confirmar Orden de Entrega
Confirmar
Despachar Orden de Entrega
Anular Orden de Entrega
Despachar
Anular
Final
Final


sd Escenario: Buscar Orden de Entrega
Gestion::AccRegistroOrdenEntrega
Fabrica::FabricaGestionOrdenEntrega
«interface»
Interfaces::IGtrOrdenDeEntrega
«interface»
Interfaces::IDaoOrdenEntrega
btnBuscar_CClick()
objGestor= CrearGtrOrdenEntrega() :
IGtrOrdenDeEntrega
ordenesEntrega= ObtenerOrdenesEntrega(idAlmacen, idUELSolicitante, idEstado,
fechaDesde, fechaHasta) :IOrdenEntrega
ObtenerOrdenesEntrega(decimal,
decimal, IEstadoOrden, DateTime,
DateTime) :IOrdenEntrega


sd Escenario: Adicionar Orden de Entrega
Gestion::AccRegistroOrdenEntrega
Gestion::AccOrdenEntrega
Fabrica::FabricaGestionOrdenEntrega
Fabrica::FabricaExistenciaBienes
Gestion::AccSeleccionarBien
«interface» Interfaces::IGtrExistenciasDelAlmacen
«interface» Interfaces::IGtrOrdenDeEntrega
«interface» Interfaces::IDaoOrdenEntrega
«interface» Interfaces::IDaoComun
btnAdicionar_CClick()
Load()
btnAdicionarBienes_CClick()
CrearGtrExistenciasDelAlmacen() :IGtrExistenciasDelAlmacen
bienesMostrar= BuscarBienesDelAlmacen(decimal) : IBienesServicio
CapturarBienesEventHandler(bienesMostrar)
btnSeleccionar_CClick()
CapturarBienesEventHandler(coleccion)
loop crear Datos del Bien para la Orden
[coleccion.count]
listaDatosBien.Add= CrearDatosBienPorOrden(1, ObjExistenciaBien.ObjBien) :IDatosBienOrden
btnEliminar_CClick()
se guardan en una lista..para ser eliminados luego
btnAceptar_CClick()
objOrden= CrearOrdenDeEntrega(codigoAlmacen, codigoUEL, CIReceptor, fecha, listaDatosBien) :IOrdenDeEntrega
CapturarOrdenEventHandler(objOrden, datosBienesEliminar)
objGestor= CrearGtrOrdenEntrega() : IGtrOrdenDeEntrega
EliminarDatosBienSalvarOrden(datosBienesEliminar, objOrden)
EliminarColeccion(datosBienesEliminar)
SalvarActualizarOrdenEntrega(objOrden)
SalvarActualizarColeccion(oBjOrden.ListaDatosBien)
loop para salvar los detalles de cada uno de esos datos
[ListaDatosBien.Count]
SalvarActualizarColeccion(objDatosBien.ListaDetallesBienUso)


sd Escenario: Confirmar Orden
Gestion::AccRegistroOrdenEntrega
Fabrica::FabricaGestionOrdenEntrega
«interface»
Interfaces::IGtrOrdenDeEntrega
«interface»
Interfaces::IDaoOrdenEntrega
«interface»
Interfaces::IDaoGestionUsuarios
RecursosCompartidos::RecursosCompartidos
alt Verificar que sea una orden En Edicion
[objOrden.EstadoOrden.idOrden == 1]
btnConfirmar_CClick()
Referencia al escenario Generar Movimiento Confirmacion del CU Generar movimiento por Orden de Entrega
objGestorUsuario= ObtenerObjeto(DaoOrdenEntrega) :object
objUsuario= ObtenerPersonaAutenticada() :Object
objOrden.RegistroEstadoOrden= CrearEstadoConfOrden(objUsuario,fecha,objMov) : IEstadoConfirmadaOE
objGestor= CrearGtrOrdenEntrega() : IGtrOrdenDeEntrega
ConfirmarOrden(objOrden)
SalvarActualizarOrdenEntrega(objOrden)
[else]
Solo se puede confirmar una orden que se encuentre En Ejecución


sd Escenario: Anular Orden
Gestion::AccRegistroOrdenEntrega
Fabrica::FabricaGestionOrdenEntrega
«interface»
Interfaces::IGtrOrdenDeEntrega
«interface»
Interfaces::IDaoOrdenEntrega
RecursosCompartidos::RecursosCompartidos
terface»
aoGestionUsuarios
btnAnular_CClick()
alt verificar que sea una orden en estado confirmada
[ObjOrden.EstadoOrden.idOrden == 2]
objGestor=
CrearGtrOrdenEntrega() :
IGtrOrdenDeEntrega
CancelarOrden(objOrden)
ObtenerObjeto(GtrUsuarios) :object
objUsuario= ObtenerPersonaAutenticada() :Object
objEstadoAnulado=
CrearEstadoAnuladoOrdenEntrega(objUsuario,
fecha)
Referencia al escenario
generar movimiento al
anular del CU: Generar
Movimiento
SalvarActualizarOrdenEntrega(IOrdenEntrega)
[else]
Solo se pueden anular las órdenes de entrega en
estado Confirmada

Anexo 8. Datos y Resultados de las Métricas OO

Clases	Métodos	Propiedades	Atributos	NOR	APH	NDD	ACO
DaoExistenciaBienesEnAlmacen	4	0	0	0	1	0	
DaoInventario	4	0	0	0	1	0	
DaoOrdenEntrega	4	0	0	0	1	0	
DaoRegistroAlmacen	2	0	0	0	1	0	
DaoMovimientoBienes	9	0	0	0	1	0	
EAlmacen	22	10	10	0	1	0	6
EBienesServicio	33	15	15	0	2	2	2
EBienInventariar	10	4	4	0	1	0	2
EConceptoMovimiento	16	7	7	0	1	0	3
EDatosBienOrden	14	6	6	0	1	0	4
EEstadoHojaInventario	10	4	4	0	1	0	2
EEstadoMovimiento	10	4	4	0	1	0	2
EExistenciaBien	14	6	6	0	1	0	2
EHojaInventario	20	9	9	0	1	0	4
EMovimientoBienes	22	10	10	0	1	0	6
EOrdenEntrega	22	10	10	0	1	0	7
EPersonalUnidad	8	6	6	0	1	0	2
ERenglon	18	8	8	0	1	0	1
EstadoOrden	8	3	3	0	1	0	1
ETipoMovimiento	8	3	3	0	1	0	1
EBienUso	40	19	19	0	1	0	4
ECicloAmortizacion	10	4	4	0	1	0	2
EConfiguracion	8	3	3	0	1	0	1
EDetallesBienUso	12	5	5	0	1	0	2
EDetallesPropTec	8	3	3	0	1	0	2
EEstadoAnuladaOE	5	3	3	1	1	0	1

Clases	Métodos	Propiedades	Atributos	NOR	APH	NDD	ACO
EEstadoConfirmadaOE	5	3	3	1	1	0	1
EEstadoConfirmadoMov	5	3	3	1	1	0	1
EEstadoDespachadaOE	5	3	3	1	1	0	1
EEstadoEdicionMov	5	3	3	1	1	0	1
EGrupo	16	7	7	0	1	0	3
EPropiedadTBien	8	3	3	0	1	0	2
EPropiedadTecnologica	14	6	6	0	1	0	3
ERegistroEstadoHojaInv	10	4	4	0	1	0	2
ERegistroEstadoMov	14	6	6	0	2	2	4
ERegistroEstadoOE	16	7	7	0	2	3	5
ESubGrupo	14	6	6	0	1	0	3
EUnidadEjecutoraLocal	20	9	9	0	1	0	3
FabricaExistenciaBienes	5	4	2	1	1	1	
FabricaGestionAlmacenes	5	4	2	1	1	1	
FabricaGestionOrdenEntrega	10	10	2	1	1	1	
FabricaInventario	10	10	2	1	1	1	
FabricaMovimientoBienes	13	12	2	1	1	1	
FabricaBienUso	3	2	2	1	1	1	
FabricaGrupSubG	9	8	2	1	1	1	
GtrExistenciasDelAlmacen	4	3	2	0	1	0	
GtrInventario	7	6	2	0	1	0	
GtrMovimientoBienes	8	7	2	0	1	0	
GtrOrdenDeEntrega	10	9	2	0	1	0	
GtrRegistroAlmacenes	5	4	2	0	1	0	
GtrGruposSubGrupos	10	9	2	0	1	0	
AccAlmacen	4	0	2	0	1	0	
AccConfirmarHojaInventario	4	0	2	0	1	0	
AccDesincorporacionBienes	6	0	2	0	1	0	

Clases	Métodos	Propiedades	Atributos	NOR	APH	NDD	ACO
AccExistenciaBienEnAlmacen	3	0	2	0	1	0	
AccExistenciasDelAlmacen	4	0	2	0	1	0	
AccGenerarHojaInventario	5	0	2	0	1	0	
AccHojaInventario	5	0	2	0	1	0	
AccIncorporacionBienes	6	0	2	0	1	0	
AccInventarioAlmacenes	11	0	2	0	1	0	
AccOrdenEntrega	7	0	2	0	1	0	
AccReasignacionBienes	5	0	2	0	1	0	
AccRegistroAlmacenes	8	0	2	0	1	0	
AccRegistroExistenciasEnAlm	5	0	2	0	1	0	
AccRegistroMovimientoDeBienes	5	0	2	0	1	0	
AccRegistroOrdenEntrega	10	0	2	0	1	0	
AccSeleccionarBien	5	0	2	0	1	0	
AccTipoDeConceptoMovimiento	4	0	2	0	1	0	
AccGruposSubGrupos	19	2	2	0	1	0	
AccEditarGrupo	4	0	2	0	1	0	
AccEditarConjuntoBienes	6	0	2	0	1	0	
AccEditarBien	6	0	2	0	1	0	
	9.78		3.75				

Glosario de Términos

Paradigma: El concepto paradigma procede del griego *paradeigma*, que significa “ejemplo” o “modelo”. En principio, se aplicaba a la gramática (para definir su uso en un cierto contexto) y a la retórica (para referirse a una parábola o fábula). A partir de la década del ’60, comenzó a utilizarse para definir a un modelo o patrón en cualquier disciplina científica o contexto cognitivo. Luego el filósofo y científico estadounidense Thomas Kuhn se encargo de ampliar el concepto refiriéndose al término como el conjunto de prácticas que definen una disciplina científica durante un período específico de tiempo.

Acta: Reseña escrita, fehaciente y auténtica de todo acto productor de efectos Jurídicos.

Amortización: Son las reducciones graduales de la deuda a través de pasos periódicos sobre el capital prestado o recuperación de los fondos invertidos en un activo de una empresa (devolución de una deuda o de un capital tomado en préstamo). En los activos intangibles, refiere a la parte del costo de dicho activo que sebe ser absorbido dentro de los gastos de los ejercicios económicos que resulten beneficiados del mismo.

Asiento Contable: Anotación que se le hace en el debe o haber de una cuenta.

Bien: Las cosas que pueden ser objeto de propiedad pública o privada.

Bienes Inmueble: Representa el valor de los terrenos, edificios e instalaciones pertenecientes al Fisco Estatal, con las siguientes excepciones:

- Las construcciones de uso y dominio público, tales como las obras viales, sanitarias, hidráulicas y similares.
- Los recursos naturales como baldíos, bosques, minas y yacimientos.

Bienes Muebles: Representa el valor de los objetos y equipos de naturaleza y carácter permanente o semipermanente, es decir, que no desaparezcan al primer uso y que estén al servicio de las dependencias del Gobierno Regional.

Contabilidad: Representación cuantitativa, sistemática y metódica de los diferentes fenómenos producidos por la actividad económica de los elementos que forman el patrimonio del organismo.

Glosario de Términos

Data Access Objects (DAO): Siglas en inglés de Data Access Object (objeto de acceso a datos). Se trata de un objeto que realiza una labor de interfaz entre la aplicación y los datos de la misma.

Depreciación: Es la recuperación periódica de una parte de la inversión hecha en un activo fijo tangible.

Encapsulamiento: Se refiere a la capacidad de agrupar y condensar en un entorno con límites bien-definidos distintos elementos.

Extend: Este tipo de relación refleja situaciones particulares en un caso de uso que pueden ser tratadas (extendidas) por otro.

Framework: Es una estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado. Puede incluir soporte de programas, bibliotecas y un lenguaje de scripting entre otros softwares para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Heredar: Permite que una clase posea las características de otra, sin tener que reescribir el código.

Instanciación: Proceso de creación de un objeto a partir de una clase.

Open-Source: Código abierto (en inglés open source). Conlleva como idea más importante la posibilidad de acceder al código fuente, modificarlo y redistribuirlo como se considere conveniente, estando sujeto al marco legal que brinda el open source.

Persistencia*:* Designa la capacidad de un objeto trascender en el espacio/tiempo salvando su estado en algún dispositivo físico justo antes de ser destruido, mediante algún mecanismo y en un determinado formato. Existe desde su creación hasta que se destruye.

Reutilización*:* Es la acción de volver a utilizar.

Unidad Administradora Central (UAC): Es Dirección Administración u otra de igual competencia de cada organismo, manejará créditos centralizados y créditos propios mediante órdenes de pago directo, e igualmente girará órdenes de avances o adelanto de fondos a las Unidades Administradoras desconcentradas y manejará el fondo de anticipo que se le asigne caja chica y fondos en avance.

Unidad Administradora Desconcentrada (UAD): Son los que con tal carácter determine la máxima autoridad del organismo ordenador. Estas Unidades deben tener una organización administrativa que les

permita manejar fondos en anticipo por un monto igual o superior a 2 500 unidades tributarias y, podrán manejar además, fondos en avance y cajas chicas, así como ordenar pagos directos contra el Tesoro Nacional, previa autorización de la máxima autoridad del organismo.

Unidad Ejecutora Local (UEL): Es la Unidad Operadora de menor nivel responsable de la ejecución física de una actividad de un proyecto.

Unidades Administradoras (UA): Están constituidas por las organizaciones administrativas de los organismos, que tienen la responsabilidad de ejecutar los créditos presupuestarios manejados mediante órdenes de pago directo al proveedor o beneficiario, avances o adelantos de fondos a funcionarios.