Universidad de las Ciencias Informáticas

Facultad 4

Tema: Análisis y diseño de un sistema para la gestión automática de entidades del negocio.

Trabajo de Diploma para optar por el Título de

Ingeniero en Ciencias Informáticas.

Autores: Maria de los Angeles Gallart Avila

Julio López Hernández

Tutor: Ing. Yulier Matías León

Ciudad de la Habana

Junio de 2010

Frase

La construcción exitosa de toda máquina depende de la perfección de las herramientas empleadas. Quien sea un maestro en el arte de la fabricación de herramientas poseerá la clave para la construcción de todas las máquinas.

Charles Babbage

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores del trabajo “Análisis y Diseño de un sistema para la gestión automática de entidades del negocio” y autorizamos a la Facultad 4 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Autores:

_____

Maria de los Angeles Gallart Avila.

_____

Julio López Hernández.

Tutor:

_____

Yulier Matías León.

Agradecimientos

A Marjories Avila Alvarez mi madre que la amo con locura. A Gilbert Gallart González mi padre, que es el mejor. Bryan Gallart Biamut, mi hermanito bello. A mis abuelitos: Fela y Berman, Madoli, Pedro, Olidia. A mis tios y tías: Alexander y Alexey, Maivis y Mileidi, Edgar, Wilberto y Pedro. A mis primos Vladi, Ivita y al Negro. A Santiago Burey y Yelaine Biamut que también los quiero. A Yulier Matías mi tutor por tener paciencia conmigo. A Ricardo Pérez Velázquez. Mairelys Fernández González y Yunieski Escobar Requejo.

Gracias a todos...

Maria de los Angeles Gallart Avila

Agradecimientos

Empezaré por agradecerles a todos los se han hecho parte de este sueño de verme graduado. Mis compañeros de aula y mis amistades. A mi familia por siempre apoyarme en mis decisiones y ser el pedestal que me mantiene firme y seguro. A mis hermanitos que los amo mucho y que siempre están en mis pensamientos y de los cuales soy su guía. A mi tutor el Ing. Yulier Matías León por estar siempre atento, por su apoyo y dedicación. A los profesores que hicieron posible mi formación profesional. A mi compañera de tesis Maria de los Angeles Gallart Avila por soportarme todo este tiempo y tener paciencia conmigo. A una personita que es el motor impulsor de mi vida, mi motivo de vivir, mi todo. Mi mamita J.F AMC. Gracias por todo lo que me has dado y por la educación que me diste. A mi novia por haberme dado a conocer el verdadero amor, te quiero mi vida. A mi papá y mi padrastro por su apoyo y por confiar en mi. A mi tercer padre el eterno comandante en jefe Fidel Castro Ruz por darme la posibilidad de estar aqui hoy y poder graduarme, por eso le digo y siempre le diré "Comándate Ordene".

Y a todos los presentes muchas gracias.

Julio López Hernández

Resumen

En el presente documento se realiza el análisis y diseño de un sistema automático para la gestión de entidades del negocio. Se ajusta su formato a los lineamientos arquitectónicos establecidos por el proyecto Modernización del Sistema Bancario Cubano. Se describen las herramientas y técnicas usadas, se sigue la metodología y los procesos que RUP propone, se emplearon patrones dentro del flujo de trabajo. Los artefactos se generan según el lenguaje de modelado UML, apoyándose en el Visual Paradigm y el Erwin.

Durante la realización del Análisis y Diseño se originaron artefactos que permiten un mejor entendimiento de los requisitos y su transformación a un lenguaje común para los desarrolladores.

Para certificar la calidad del presente trabajo se aplicaron métricas, obteniendo resultados positivos.

Tabla de contenido

Tabla de Contenidos

Figuras

Tabla de Figuras

Tabla de Figuras

Introducción

Las Tecnologías de la Información y las Comunicaciones (TIC) forman parte de la cultura tecnológica que rodea al mundo y amplían las posibilidades de desarrollo social. Su impacto es perceptible en las empresas dedicadas a la comercialización de software.

El desarrollo de las organizaciones demanda una enorme cantidad de información. Para el hombre, la realización de este trabajo es demasiado engorrosa, por tal razón surge la necesidad de automatizar los procesos de gestión empresarial.

La automatización de los procesos de gestión facilita a la dirección de la empresa el máximo aprovechamiento de los recursos existentes. Estas soluciones permiten afrontar los desafíos con todas las garantías dentro de los nuevos marcos tecnológicos de trabajo y mercado. La informática enfrenta estos problemas y los relaciona, estudiando la mejor forma de proporcionar la información necesaria, con el fin de tomar decisiones. Para lograrlo esta ciencia estudia el diseño y la utilización de equipos, sistemas y procedimientos que permiten captar y tratar los datos adecuados para obtener información útil. Dentro de las herramientas utilizadas se encuentra las generadoras de código.

Una herramienta generadora de código permite optimizar el proceso de desarrollo de software, agilizándolo para obtener el producto en menor tiempo. Además es usada para construir programas de forma automática evitando que los programadores tengan que escribir el código de forma manual.

En la Universidad de las Ciencias Informática (UCI) se llevan a cabo proyectos productivos en los que ocurren demoras en el proceso de desarrollo de software debido a que se emplea mucho tiempo en la implementación de la Gestión de Entidades de Negocio.

Las empresas generan cambios en el negocio y por tanto cambia el sistema que gestiona sus entidades y procesos. Se distingue como situación problemática la existencia de tecnologías complejas y diversas para desarrollar software, que demanda mayor preparación en el ingeniero informático a la hora de implementar y además los proyectos dedican mucho tiempo al desarrollo de CU de sencilla y mediana complejidad, los cuales se pueden implementar haciendo uso de herramientas automáticas. Las empresas se ven afectada por la demora, lo que hace notable la necesidad de agilizar el proceso de desarrollo de software.

Se traza como **problema a resolver** ¿Cómo transformar la gestión de entidades del negocio para el desarrollo de software, a un lenguaje entendible por los programadores, que permita la posterior implementación?

Como guía de la investigación se plantea la siguiente **idea a defender:** con el análisis y diseño de un sistema automático para la gestión de entidades del negocio, se facilitará su posterior implementación.

Se toma como **objeto de estudio** el proceso desarrollo del software. De ello se deriva que el **campo de acción** sean las herramientas para la generación de código.

Se define como **objetivo general de la investigación** elaborar el análisis y diseño de un sistema que permita automatizar el desarrollo de software para la gestión de entidades del negocio.

Para dar cumplimiento a los objetivos se planificaron las **tareas** correspondientes:

Estudiar las herramientas generadoras de código existentes con características similares al sistema que se quiere diseñar.

Analizar los requerimientos del sistema.

Estudiar las herramientas, metodologías y tecnología a usar para la elaboración de la propuesta de solución.

Realizar el análisis de los caso de uso del sistema tratado.

Efectuar el diseño de los casos de uso del sistema.

Estudiar las métricas para la validación de la propuesta de diseño que se elabore.

Validar la propuesta de diseño elaborada.

Del presente trabajo se espera como **resultado** el análisis y diseño de los casos de uso para la gestión de modelo y la generación de código, consiguiendo un lenguaje descifrable por los implementadores que sirva como base para su posterior implementación.

Para el desarrollo de la investigación se utilizaron diferentes **métodos** con el objetivo de proveer a la investigación de bases fundamentadas para su exitosa evolución.

Los métodos de investigación **teóricos** permiten estudiar y descubrir las características del objeto de estudio que no son directamente observables.

Analítico - Sintético permite buscar la esencia de los fenómenos, los rasgos que lo caracterizan y los distinguen. Su objetivo en una investigación es analizar las teorías, documentos, para la extracción de los elementos más importantes que se relacionan con el objeto de estudio.

Inductivo - Deductivo son formas de razonamiento que permiten llegar a un grupo de conocimientos generalizadores, tanto desde el análisis de lo particular a lo general, como desde el análisis de elementos generalizadores a uno de menor nivel de generalización.

Análisis Histórico – Lógico permite estudiar, de forma analítica, la trayectoria histórica real de los fenómenos, su evolución y desarrollo.

Modelación admite la creación de modelos, (propuestas, alternativas, estrategias).

El documento consta con un Resumen, Tabla de Contenidos, Introducción, Capítulos seguidos de conclusiones parciales, Conclusiones generales, Recomendaciones, Bibliografía, Anexos y Glosario de Términos.

Capítulo 1: Brinda un breve acercamiento a los principales conceptos asociados al dominio del problema y que son abordados a lo largo del trabajo. Incluye además lo referente al estudio del arte en el que está enmarcada la investigación.

Capítulo 2: Contiene el análisis de los requerimientos, necesarios para cumplir todas las funcionalidades de la gestión de entidades del negocio, estableciendo la base para su diseño.

Capítulo 3: Propone el diseño detallado del funcionamiento interno de cada uno de los procesos identificados en el análisis, incluyendo el diseño de la interacción de las clases de cada proceso para demostrar técnicamente como trabaja cada uno de ellos. Se emplearán además las validaciones necesarias para demostrar el correcto desempeño del diseño realizado.

Capítulo 1: Fundamentación teórica

1.1 Introducción

El capítulo se enmarca en los aspectos relacionados al objeto de estudio definido con anterioridad para el presente trabajo. Se realiza una valoración de algunos sistemas existentes que generan código, en los que se tienen en cuenta los conceptos asociados al campo de acción del problema propuesto. Se presentarán elementos claves para poder realizar el análisis y diseño de los requisitos funcionales. Se menciona y describe la metodología, notación, tecnologías y herramientas que se emplearán para el desarrollo de la solución.

1.2 MDE

La Ingeniería Dirigida por Modelos (en inglés *Model-Driven Engineering*, MDE) es una metodología de desarrollo de software, se centra en la creación de modelos o abstracciones basados principalmente en los conceptos del dominio particular. El uso de modelos permite aumentar el nivel de abstracción con que se realizan los sistemas. La idea promovida por MDE es usar modelos con diferentes niveles de abstracción para el desarrollo de sistemas. El manejo del enfoque MDE facilita la comunicación de ideas, ya que estas se pueden expresar de manera explícita y no diluida entre interminables líneas de código.

Un Paradigma de modelado MDE se considera eficiente si los modelos cobran sentido desde el criterio de los usuarios. [The Enterprise Architect Building An Agile Enterprise. (2009)]

1.2.1. MDA

La Arquitectura Dirigida por Modelos, en inglés Model Driven Architecture (MDA), es una propuesta del grupo OMG[1] (en inglés Object Managment Group). Es un Marco de trabajo para el desarrollo de software; su principal característica es la definición de modelos como elementos de primer orden en el diseño e implementación del software. MDA considera diferentes tipos de modelos, en función del nivel de abstracción de los mismos: los requisitos del sistema son detallados en el Modelo Independiente de

[1] OMG: La OMG (Object Management Group) es una asociación sin fines de lucro formada por grandes corporaciones, muchas de ellas de la industria del software, como IBM, Apple, Sun Microsystems y Microsoft. Se encarga de la definición y el mantenimiento de estándares para aplicaciones de la industria de la computación. Algunos de los estándares definidos por la OMG son el UML y el CORBA, que ofrece interoperabilidad multiplataforma a nivel objetos de negocios.

Computación (en inglés Computation Independent Model, CIM); en los Modelos Independientes de Plataforma (en inglés Platform Independent Model, PIM) se representa la funcionalidad del sistema sin considerar la plataforma final y los Modelos Específicos de Plataforma (en inglés Platform Specific Model, PSM) se obtienen al combinar las especificaciones contenidas en el PIM con los detalles de la plataforma seleccionada. A partir de los diferentes PSMs se pueden generar automáticamente distintas implementaciones del sistema.

El proceso de transformación de modelos es un tema que ha sido ampliamente estudiado y que cobra relevancia bajo el enfoque de MDA. La transformación de modelos está sustentada en la definición de las reglas para convertir un modelo de un nivel de abstracción a otro basándose en lenguajes y mecanismos estándares, con el propósito de lograr una solución genérica al ambicionado sueño de "programar modelos".

MDA permite unir diferentes implementaciones mediante puentes. Y brinda la posibilidad de realizar sistemas robustos y de gran escalabilidad. [IBM. (2004)]

1.3 Herramientas generadoras de código

1.3.1. OptimalJ

OptimalJ implementa el marco de trabajo MDA usando tecnologías como MOF, UML, XMI, WSDL, J2EE[2].

Posee tres tipos de modelos:

Modelo de dominio: Modelo que describe el sistema a un alto nivel de abstracción. Corresponde al PIM de la aplicación. Dentro de este modelo se puede definir los Patrones de Dominio (en inglés Domain Patterns). Además está formado por el modelo de clase y el modelo de servicio.

Modelo de aplicación: Modelo del sistema desde el punto de vista de una tecnología determinada (J2EE), y su transformación la realizan los Patrones de Tecnología. Contiene los PSM de la aplicación. Se genera automáticamente a partir de un modelo del dominio y está formado por tres modelos: Modelo de base de datos, Modelo de interfaz web y Modelo EJB.

2 J2EE: Java Platform, Enterprise Edition o Java EE es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en lenguaje JAVA con arquitectura de N niveles distribuidos, basándose en componentes de software modulares se ejecuta sobre servidores aplicaciones.

Modelo de código: Código de la aplicación, generado a partir de un modelo haciendo uso de Patrones de Código. El código de la lógica de negocio EJB, base de datos y presentación (Web), puede generarse automáticamente de forma separada, o todo a la vez. Todo el código generado por OptimalJ, se sitúa en bloques protegidos, aunque deja bloques libres para que el programador genere extensiones. [ALS. OptimalJ de Compuware. (2002-2007)]

1.3.2. AndroMDA

AndroMDA (pronunciado "Andrómeda") es un marco generador extensible que se adhiere a la Arquitectura Dirigida por el Modelo (MDA). Los modelos de las herramientas UML serán transformados en componentes de despliegue de su plataforma favorita (J2EE, Spring, .NET). A diferencia de otros juegos de herramientas MDA, AndroMDA viene con una serie de cartuchos confeccionados (en inglés ready-made) de las herramientas objetivo, como el desarrollo Struts, JSF, Spring e Hibernate. AndroMDA también contiene un conjunto de herramientas para la construcción de sus propios cartuchos o personalizar los existentes. Al usarlo, puede construir un generador de código personalizado que utiliza la herramienta de UML. [AndroMDA. (2010)]

1.3.3. Spring Roo

Herramienta de desarrollo de aplicaciones para implementadores de Java. Se pueden construir fácilmente aplicaciones Java completas en cuestión de minutos. Se diferencia de otras herramientas, centrándose en: mayor productividad de Java; utiliza las API de Java y cuenta con un nivel extremadamente alto de usabilidad y un depósito avanzado.

Spring Roo es una herramienta de código abierto, pero no cumple con los requerimientos que se propone el sistema. [Spring Source. (2010)]

1.3.4. CodeCharge Studio

CodeCharge Studio es la principal y más productiva solución para crear aplicaciones web orientadas a base de datos con el mínimo esfuerzo en programación. Soporta casi todas las bases de datos y tecnologías web. El constructor de aplicaciones incluido es un potente generador automático de aplicaciones que convierte instantáneamente una base de datos en una aplicación web con accesos protegidos por usuarios, con búsqueda, listados, consultas.

CodeCharge Studio ofrece una velocidad de generación de código integrado ofreciendo un uso completo. Es un IDE de desarrollo de gran alcance. Puede generar rápida y dinámicamente, bug-sitios web gratis en PHP, ASP, JSP, Perl, ColdFusion, ASP.NET. Y luego edita y personifica las aplicaciones con editores de código de gran alcance. Puede utilizar cualquier editor HTML para modificar el código HTML + CSS Web en diseños creados por CodeCharge Studio.

Beneficios

Reduce el tiempo de desarrollo, elimina las tareas de programación que consume y construye aplicaciones Web robustas.

Minimiza errores de programación y ortográficos, estructurando bien el código.

Limitaciones

A pesar de todas las funciones que realiza la herramienta tiene una limitante: es propietario y de pago. [CodeCharge Studio]

1.3.5. CodeSmith

CodeSmith: generador de código a base de plantillas para cualquier lenguaje. Por el empleo de propiedades se puede personalizar el código generado. CodeSmith es 100% extensible porque le permite al usuario crear tipos de propiedades de encargo. Su sintaxis es similar a ASP.NET, usa C*, VB.NET o JSCRIPT que son lenguas puras de sus plantillas, que pueden dar salida a cualquier lenguaje a base de ASCII; usa XML para conducirlas. En CodeSmith las capacidades se ajustan permitiendo combinar el código generado y escrito a mano dentro de un solo archivo. CodeSmith incluye muchos rasgos, todos diseñados para ayudarle a escribir su código más rápido y con menos defectos.

Esta herramienta tiene como limitación que es privativo. [CodeSmith Tools]

1.3.6. OpenXava

Solo para JAVA, no soporta la gestión de modelo. Permite crear aplicaciones de gestión con Java (AJAX).

Desarrollar con Ruby On Rails, Spring MVC o cualquier otro marco de trabajo MVC[3] (Modelo-Vista-Controlador) es más lento que con OpenXava, porque solo se escribe el modelo; los controladores se rehúsan y la vista se genera de forma automática.

En OpenXava solo se escribe el modelo, los controladores se reutilizan y la vista se genera automáticamente. Es potente, extensible, y divertido para un experto de Java. Pero tiene como limitante que evita el MVC para conseguir su objetivo y es solo para Java. [OpenXava.]

1.3.7. GeneXus

GeneXus es una poderosa herramienta multiplataforma para el desarrollo de aplicaciones.

Permite el desarrollo incremental de aplicaciones de negocios. Genera el 100% de la aplicación, y mantiene automáticamente el modelo de datos, la información y las aplicaciones. GeneXus es usado para desarrollar complejas aplicaciones de misión crítica con extensas bases de datos, que incluyen tanto aplicaciones centralizadas como distribuidas y basadas en Web. Valida sus requerimientos en la etapa de diseño, a través de prototipos 100% funcionales. Genera la base de datos y cuando cambian los requerimientos, automáticamente realiza un análisis de impacto y propaga los cambios.

Plataformas para GeneXus:

Java/J2EE, .NET, .NET Compact Framework, Ruby.

Sistemas Operativos:

IBM OS/400, LINUX, UNIX, Windows NT/2000/2003 Servers, Windows NT/2000/XP, Windows Mobile.

Internet:

Java/J2EE, ASP.NET, CGI, HTML, Web Services.

Sistemas de Gestión de Base de Datos:

IBM DB2, Informix, Microsoft SQL Server, MySQL, Oracle, PostgreSQL.

Lenguajes:

Java/J2EE, C#, Ruby, COBOL, RPG, Visual Basic, Visual FoxPro.

[3] MVC: Patrón de arquitectura modelo-vista-controlador. Basado en acciones Struts.

Servidores Web:

Microsoft IIS, Apache, Web Sphere.

Múltiples Arquitecturas:

Arquitecturas en múltiples capas, basadas en Web, cliente/servidor y Centralizada (AS/400, iSeries, System i).

Herramientas de Business Intelligence y Work_ow:

Soluciones de Reporting, Data Warehousing y Workflow para todas las plataformas soportadas.

GeneXus es un sistema muy completo pero es propietario y de pago. [Genexus.]

1.4 Valoración del estado del arte

Después del análisis de las herramientas se llegó a la conclusión de que cada una tiene una solución diferente al problema de la generación de código. Las entradas de los datos no son las mismas para cada una de ellas y se rigen por sus requisitos y alcance específicos. Los requerimientos del sistema a diseñar son diferentes a los presentados por otras aplicaciones, lo que dificulta la posibilidad a adoptar una solución existente, por lo que se propone realizar un análisis y diseño utilizando como base a los requisitos de los sistemas estudiados incluyéndoles nuevas funcionalidades.

1.5 Ingeniería de software

La Ingeniería de Software es una tecnología multicapa en la que, según Pressman, se pueden identificar: los métodos, a través de los que se indica cómo construir técnicamente el software; procesos, que son el fundamento de la Ingeniería de Software y el elemento de unión que mantiene juntas las capas de la tecnología; herramientas, que constituyen el soporte automático o semiautomático para el proceso y los métodos.

La Ingeniería de Software es la aplicación práctica del conocimiento científico en el diseño, construcción de programas de computadora y la documentación requerida para desarrollar, operar (funcionar) y mantener los programas. [Pressman, Roger S.]

Figura 1: Ingeniería de software es una tecnología multicapa.

1.6 Metodología de desarrollo

Las metodologías están en constante evolución, con el objetivo de solucionar los problemas existentes en la producción de software, los que son cada vez más complejos. Estas abarcan procedimientos, técnicas, herramientas y documentación que se utiliza en la creación de un producto de software. Una metodología es un proceso.

Un proceso de desarrollo de software es la definición del conjunto de actividades que guían los esfuerzos de las personas implicadas en el proyecto, a modo de plantilla explica los pasos necesarios para terminar el proyecto. No existe una metodología de software universal. Las características de cada proyecto (equipo de desarrollo, recursos) exigen que el proceso sea configurable.

Dentro de las metodologías más usadas se pueden mencionar: RUP, XP (en inglés Extreme Programing) y MSF (en inglés Microsoft Solución Framework). RUP se adapta mejor a los proyectos a largo plazo, dividiendo el ciclo de vida del software en cuatro fases desarrolladas iterativamente y abarcando seis disciplinas ingenieriles y tres de apoyo.

Es necesario utilizar una metodología para dar solución al problema planteado, que explique los pasos para realizar un software correctamente. [Pressman, Roger S.]

RUP

Proceso Unificado Racional (RUP): es una metodología para la Ingeniería de Software que va más allá del análisis y diseño orientado a objetos, proporciona una familia de técnicas que soportan el ciclo completo del proceso de desarrollo de software. El resultado es un proceso basado en componentes. RUP está preparado para desarrollar grandes y complejos proyectos, unifica los mejores elementos de metodologías anteriores y utiliza el Lenguaje Unificado de Modelado (UML), para la representación visual.

Ventajas de RUP

Valoración en cada fase que permite cambios de objetivos.

Funciona bien en proyectos de creación.

Análisis y seguimiento detallado en cada una de las fases.

Desventajas de RUP

La evaluación de riesgos es compleja.

El cliente deberá ser capaz de describir detalladamente cómo quiere el producto para poder acordar un alcance del proyecto. [El Proceso Unificado de Desarrollo de Software. (2000)]

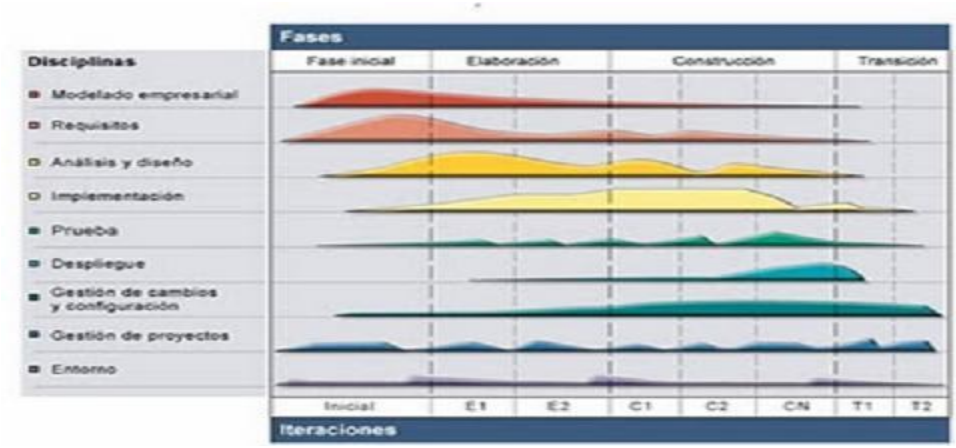


Figura 2: Metodología: Proceso Unificado Relacional (RUP).

1.7 Lenguajes

1.7.1. Lenguaje de modelado

UML

Lenguaje Unificado de Modelado (en inglés Unified Modeling Language) surge debido a la necesidad de estandarizar la manera de representar gráficamente los modelos. Permite comunicar los diseños gráficos realizados a otros desarrolladores y sirve de apoyo en los procesos de análisis de un problema. Es un lenguaje de modelado que permite visualizar, especificar, construir y documentar los artefactos de un sistema de software; por lo que se tendrá en cuenta a la hora de llevar a cabo el análisis y diseño de la herramienta a diseñar. [Rubiano, Ing. Sandra Merchán.(2005)]

1.7.2. Lenguaje de programación

Java

Es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems. El lenguaje en toma mucha sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Una de las principales características del lenguaje Java es su capacidad de que el código funcione sobre cualquier plataforma de software y hardware. Además resuelve los problemas de complejidad en los grandes sistemas.

1.8 Sistemas y herramientas para la gestión de base de datos

Los sistemas de gestión de base de datos (SGBD), (en inglés DataBase Management System, abreviado DBMS) son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan.

Erwin

Herramienta que facilitará el modelado de la Base de Datos correspondiente a la herramienta que se diseñará debido a las facilidades que brinda, específicamente; modela entidades, relaciones de dependencia entre una entidad y otra, si exporta su llave primaria como identificadora (único registro) o no-identificadora (varios registros) para alguna clave, genera los scripts para crear la BD, maneja varias BD (SyBase, Oracle, SQLServer, Paradox, Informix). [Erwin. (2010)]

1.9 Herramientas de desarrollo

1.9.1. Herramienta CASE

Las herramientas CASE constituyen un conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo del software, completo o en algunas fases.

Las herramientas CASE ayudan a los gestores y practicantes de la ingeniería del software en todas las actividades asociadas a los procesos de software. Automatizan las actividades de gestión de proyectos, gestionan todos los productos de los trabajos elaborados a través del proceso y ayudan a los ingenieros en el trabajo de análisis, diseño e implementación. Al disponer de automatización, contribuye a que el usuario elabore resultados adicionales y personalizados, que no serán fáciles ni prácticos de producir sin

el soporte de estas herramientas. Constituyen un complemento importante de las prácticas de ingeniería del software.

La principal ventaja de la utilización de una herramienta CASE, es que logra una mejor calidad de los procesos realizados por lo que aumenta la productividad. [Herramienta CASE]

Visual Paradigm 6.4

Visual Paradigm usa un lenguaje estándar para todo el equipo de desarrollo y facilita la comunicación. Es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado ayuda a una rápida construcción de aplicaciones de calidad, mejores y con menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y crear documentación. Entre sus principales características podemos encontrar:

Soporta aplicaciones web.

Compatibilidad entre ediciones. [Visual Paradigm for UML]

1.10 Framework

El marco de trabajo es una estructura conceptual y tecnológica de soporte definida, con artefactos o módulos de software concretos. Puede incluir soportes de programas, bibliotecas y un lenguaje interpretado, que se utilizan para implementar la estructura de un modelo para una aplicación. Se efectúa con el objetivo de promover la reutilización de código, dando la posibilidad de no perder tiempo en lo que está hecho. Existen diferentes tipos de marco de trabajo, para diferentes propósitos, algunos orientados al desarrollo de aplicaciones web, o para un determinado sistema operativo o lenguaje. Un marco de trabajo representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Brinda una estructura, una metodología de trabajo lo cual extiende o utiliza las aplicaciones del dominio. [Pérez Betancourt. (2009)]

1.10.1. Spring Framework

Spring se basa en la técnica Inversión de Control (IoC), método de programación en cual el flujo de ejecución de un programa se invierte respecto a los métodos tradicionales de programación, en los que la interacción se expresa de forma imperativa haciendo llamadas a procedimientos o funciones; técnica que

promueve el bajo acoplamiento, a partir de la inyección de dependencias (DI), entre los objetos y una implementación de desarrollo según el paradigma de Programación Orientación a Aspectos (AOP) que presenta una estructura simplificada para el desarrollo y utilización de aspectos (módulos en inglés multiple object crosscutting).

Spring básicamente es un contenedor, que se encarga de gestionar, administrar el ciclo de vida y de la configuración de las clases de la aplicación. No tiene ningún costo, ni se necesita licencia para utilizarlo, brinda la posibilidad de incursionar en la utilización de esta aplicación, además de estar disponible todo el código fuente de este marco de trabajo en el paquete de instalación.

Spring integra las diferentes tecnologías existentes en un solo marco de trabajo posibilitando un desarrollo más sencillo y eficaz.

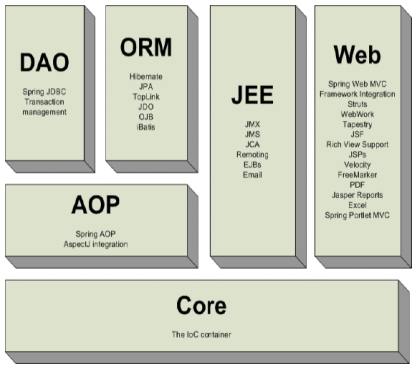


Figura 3: Arquitectura del marco de trabajo Spring.

Core: Es el núcleo de Spring. Permite técnicas de Inversión de Control (IoC) como la inyección de dependencias.

AOP: Proporciona una implementación de programación orientada a aspectos, permitiendo definir puntos de corte e interceptores.

DAO: Proporciona el acceso a una capa de abstracción JDBC (en inglés Java Database Connectivity) con una forma de administrar transacciones desde el módulo AOP como es el caso de Hibernate y JDO.

ORM: Provee capas de integración para APIs de mapeo objeto-relacional.

Web: Posibilita determinadas características apropiadas para el desarrollo de aplicaciones web e integración con otros frameworks (Struts, JSF, Tapestry).

JEE: Acceso e interacción con servicios Enterprise.

Spring ofrece mucha libertad a los desarrolladores de Java y soluciones bien documentadas y fáciles de usar para las prácticas comunes. [Pérez Betancourt. (2009)]

Spring Web Flow

Spring Web Flow es un módulo de Spring, dirigido a la definición y gestión de los flujos de páginas dentro de una aplicación web definiéndolos mediante estados y transiciones. Consiente en la secuencia de páginas por las que pasa una aplicación en función de la conversación que tenga con el usuario. En dependencia de las opciones y resultados de las operaciones de proceso que el usuario escoja, la aplicación seguirá una ruta específica de páginas. Es una plataforma web de alto nivel que incrementa la productividad, calidad y facilidad para realizar pruebas en el proceso de desarrollo. [Introducción a Spring Web Flow]

1.10.2. Hibernate Framework

Hibernate es una solución objeto relacional de mapeo ORM (en inglés Object-Relational Mapping) para Java, de código abierto y tiene licencia eximida de costo. Es un entorno de trabajo que tiene como objetivo facilitar la persistencia de objetos Java en bases de datos relacionales y al mismo tiempo la consulta de estas bases de datos para obtener objetos. Está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos realizada. También tiene la funcionalidad de crear la base de datos a partir de un modelo objetual existente. [Pérez Betancourt. (2009)]

1.11 Métricas para validar diseño orientado a objetos

Las métricas para sistemas OO deben ajustarse a las características que distinguen el software OO del software convencional. Estas métricas hacen hincapié en el encapsulamiento, la herencia, complejidad de clases y polimorfismo. Teniendo como objetivo principal comprender mejor la calidad del producto, estimar la efectividad del proceso y mejorar la calidad del trabajo realizado a nivel de proyecto.

Métricas aplicadas a clases: la clase encapsula a las operaciones (procedimientos) y a los atributos (datos). La clase suele ser el “predecesor” de las subclases (que a veces se denominan “descendientes”) que heredan sus atributos y operaciones. La clase suele colaborar con otras clases. Todas estas características se pueden utilizar como bases de las métricas que a continuación se exponen.

Conjunto de métricas CK (Chidamber y Kemerer)

Los Métodos Ponderados por Clase (MPC), son aquellos donde se definen n métodos de complejidad C_1, C_2..., C_n, para una clase C. La métrica de complejidad especificada debe normalizarse de tal modo que la complejidad nominal para un método tome el valor 1.0. MPC = Ó C_i, para i = 1 hasta n.

El número de métodos y su complejidad es un indicador razonable de la cantidad de esfuerzo necesaria para implementar y comprobar una clase. Además, cuanto mayor sea el número de métodos, más complejo será el árbol de herencia, (todas las subclases heredan el método de sus predecesores). Finalmente, a medida que el número de métodos crece para una clase dada; es más probable que se vuelva cada más específico de la aplicación, imitando por tanto su potencial de reutilización. Por todas estas razones, MPC debería mantener un valor tan bajo como sea razonable.

Árbol de Profundidad de la Herencia (APH), esta métrica se define como la longitud máxima desde el nodo hasta la raíz del árbol. A medida que crece el APH, es más probable que las clases de niveles inferiores hereden muchos métodos. Esto da lugar a posibles dificultades cuando se intenta predecir el comportamiento de una clase. Una jerarquía de clases profunda (con un valor grande de APH) lleva también a una mayor complejidad de diseño. Por el lado positivo, los valores grandes de APH implican que se pueden reutilizar muchos métodos.

Número de Descendientes (NDD), las subclases que son inmediatamente subordinadas a una clase son denominadas descendientes. A medida que crece el número de descendientes, se incrementa la reutilización, pero también es cierto, que a medida que aumenta NDD, la abstracción representada por la clase predecesora puede verse diluida. Esto significa que existe la posibilidad de que algunos de los descendientes no sean realmente miembros propios de la clase predecesora.

Métricas propuestas por Lorenz y Kidd

En el libro de métricas realizado por Lorenz y Kidd, dividen las métricas basadas en clases en cuatro categorías: tamaño, herencia, valores internos y valores externos. Las métricas orientadas a tamaños para una clase 00 se centran en cálculos de atributos y de operaciones para una clase individual, y promedian los valores para el sistema 00 en su totalidad. Las métricas basadas en herencia se centran en la forma en que se reutilizan las operaciones a lo largo y ancho de la jerarquía de clases. Las métricas para valores internos de clase examinan la cohesión y asuntos relacionados con el código, y las métricas orientadas a valores externos examinan el acoplamiento y la reutilización.

Tamaño de Clase (TC), el tamaño general de una clase se puede determinar empleando las medidas siguientes: el número total de operaciones (tanto operaciones heredadas como privadas de la instancia) que están encapsuladas dentro de la clase; El número de atributos (tanto atributos heredados como atributos privados de la Instancia) que están encapsulados en la clase.

Si existen valores grandes de TC mostrarán que una clase puede tener demasiada responsabilidad, lo cual reducirá la reutilización de la clase y complicará la implementación y la comprobación, por otra parte cuanto menor sea el valor medio para el tamaño, más probable es que las clases existentes dentro del sistema se puedan reutilizar ampliamente.

Número de Operaciones Invalidadas por una subclase (NOI), existen casos en que una subclase sustituye una operación heredada de su superclase por una versión especializada para su propio uso, y a esto se le denomina invalidación. Los grandes valores de NOI suelen indicar un problema de diseño ya que si NOI es elevado, entonces el diseñador ha violado la abstracción implicada por la superclase. Esto da lugar a una jerarquía de clases débil, y a un software 00 que pueda resultar difícil de comprobar y modificar.

En el capítulo 3 se exponen las métricas que se usan en el proceso de validación de la propuesta de diseño.

1.12 Conclusiones Parciales

En este capítulo se abordaron los conceptos relacionados con el problema, se realizó una presentación de las técnicas y herramientas seleccionadas, además se plasmó un estudio de varios sistemas vinculados al proceso y de esta forma se llega a la conclusión de que aunque se puede utilizar algunas funcionalidades de estos sistemas es necesario la realización de un producto con funcionalidades nuevas. Además se plasmaron las herramientas, metodología y lenguaje que se utilizará a lo largo del ciclo de vida del software.

Capítulo 2: Análisis

2.1. Introducción

En este capítulo se realiza el análisis de los requerimientos funcionales, descritos en la captura de requisitos que debe cumplir el sistema, para una mayor comprensión de que debe hacer el sistema. Se muestran artefactos que agregan un valor importante para la realización del diseño en posteriores etapas; concretamente: los diagramas de clases de análisis y de interacción (colaboración).

Según Pressman el análisis de requisitos es un proceso de descubrimiento, refinamiento, modelado y especificación.

2.2. Modelo de Dominio

El modelo de dominio es una representación gráfica de las clases conceptuales del mundo real, no de componentes de software. Este modelo describe como se estructura y enlaza la información, cuya función principal es ayudar a comprender el problema. A continuación se presentará el modelo de dominio correspondiente al sistema en cuestión. [Modelo del Dominio]

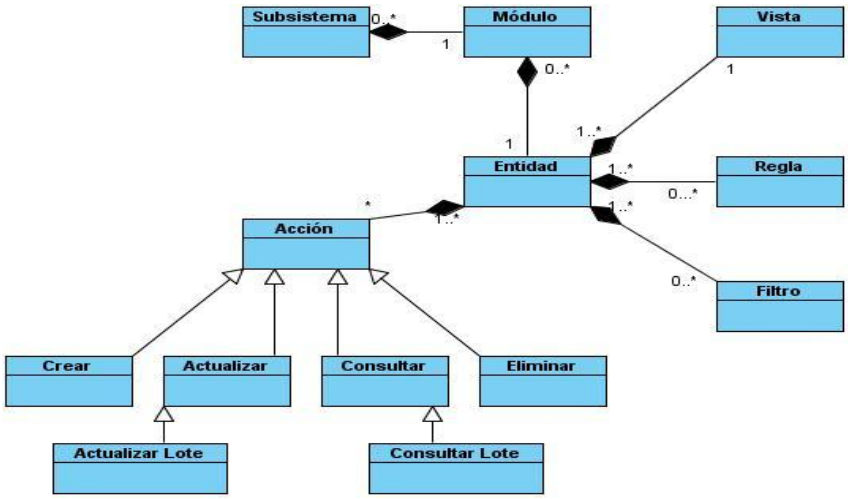


Figura 4: Modelo de Dominio.

2.3. Diagrama de Caso de Uso del Sistema

Un diagrama de caso de uso es una especie de diagrama de comportamiento, que describe lo que hace un sistema desde el punto de vista de un observador externo. Muestra la relación entre los actores y los

casos de uso del sistema, así como las relaciones entre ellos a través de relaciones de inclusión, extensión y generalización / especialización. [Diagramas de Caso de Uso. (2009)]

El diagrama de casos de uso del sistema que se presenta a continuación corresponde a la captura de requisitos de los procesos fundamentales que deben ser desarrollados

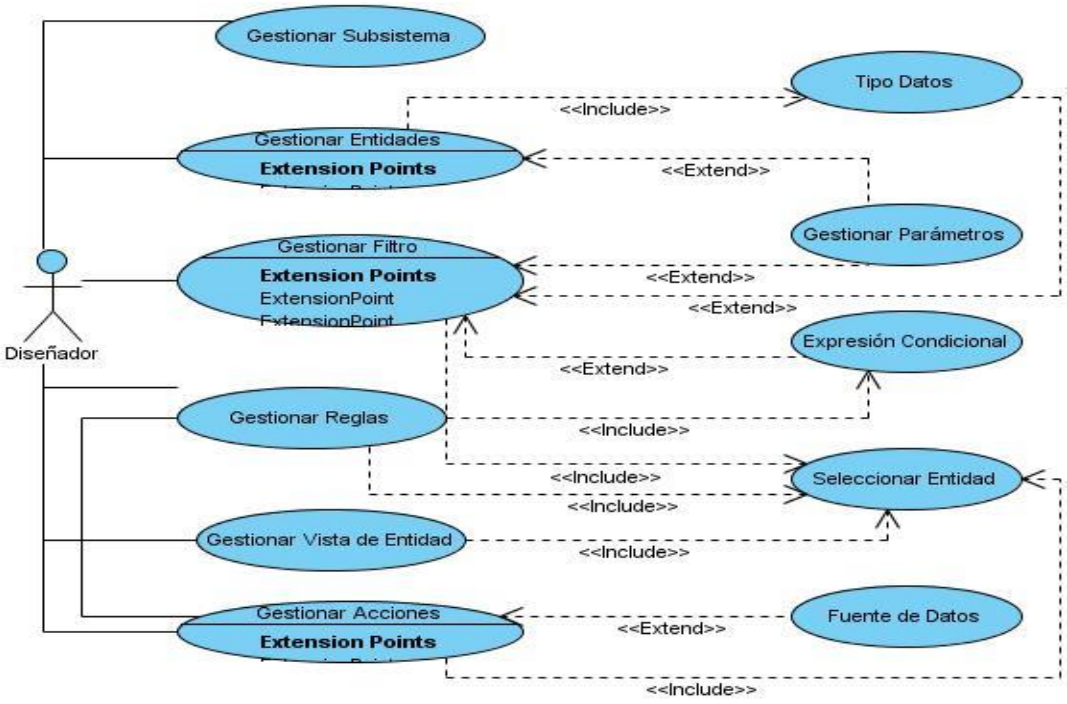


Figura 5: Diagrama de Casos de Uso General.

En el Diagrama antes presentado, cuando se instancia un caso de uso incluido o extendido, no se especifica en el flujo normal de eventos del caso de uso base sino de manera independiente.

Los casos de uso comunes son:

- Gestionar Parámetros.
- Seleccionar Entidad.
- Seleccionar Tipo de Datos.
- Seleccionar Fuente de Datos.
- Generar Expresión Condicional.

2.4. Análisis

El análisis consiste en transformar los requisitos funcionales en un diseño de clases, en el cual se vean las relaciones e interacciones que existen entren los requerimientos. Se tiene presente en este proceso una arquitectura robusta, que permita adaptar el sistema al entorno de implementación que se está diseñando.

- Gestionar Entidad.
- Gestionar Subsistemas.
- Gestionar Reglas.
- Gestionar Filtro.
- Gestionar Acciones.
- Gestionar Vistas.

Clases de Análisis: se centran en los requisitos funcionales y son evidentes en el dominio del problema porque representan sus conceptos y relaciones. Tienen atributos y entre ellas se establecen relaciones de asociación, agregación / composición, generalización / especialización y tipos asociativos.

Siempre encajan en tres estereotipos básicos.

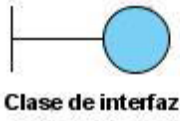
Nombre	Características	Representación
Interfaz	Modelan la interacción entre el sistema y sus actores.	 Clase de interfaz
Control	Representa coordinación secuencial, transacciones y control de otros objetos. Encapsula el control de un CU en concreto. Se usa para representar derivaciones y cálculos complejos.	 Clase de control
Entidad	Modelan información que posee larga vida y que es a menudo persistente.	 Clase de entidad

Tabla 1: Estereotipos de las clases del análisis.

Realización de los casos de uso en el análisis: colaboración por parte del modelo de análisis en el que se muestra cómo se ejecuta un determinado caso de uso[4] en términos de clases del análisis y de sus objetos del análisis en interacción.

2.5. Diagramas de clases de análisis

Un diagrama de clases del análisis es un artefacto que representa los conceptos de un dominio del problema. Es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos, y relaciones.

Cuando se selecciona el Adicionar, perteneciente a cualquier módulo, mediante el caso de uso Buscar, no se especifican los diagramas, porque son iguales a los del caso de uso Adicionar.

Diagramas realizados en la etapa de Análisis:

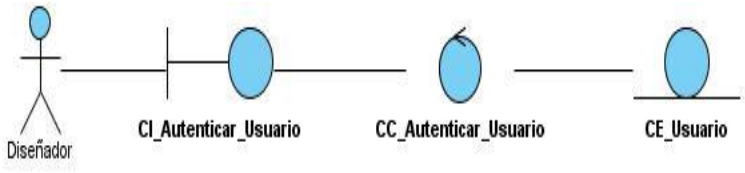


Figura 6: Diagrama de clases de análisis para el Caso de uso: Autenticar usuario.

Diagrama de clases de análisis general.

[4] Un caso de uso es una descripción de las secuencias de interacciones que se producen entre el actor y el sistema; expresa una unidad coherente de funcionalidad. [Ver Anexos: Descripción de Casos de Uso]

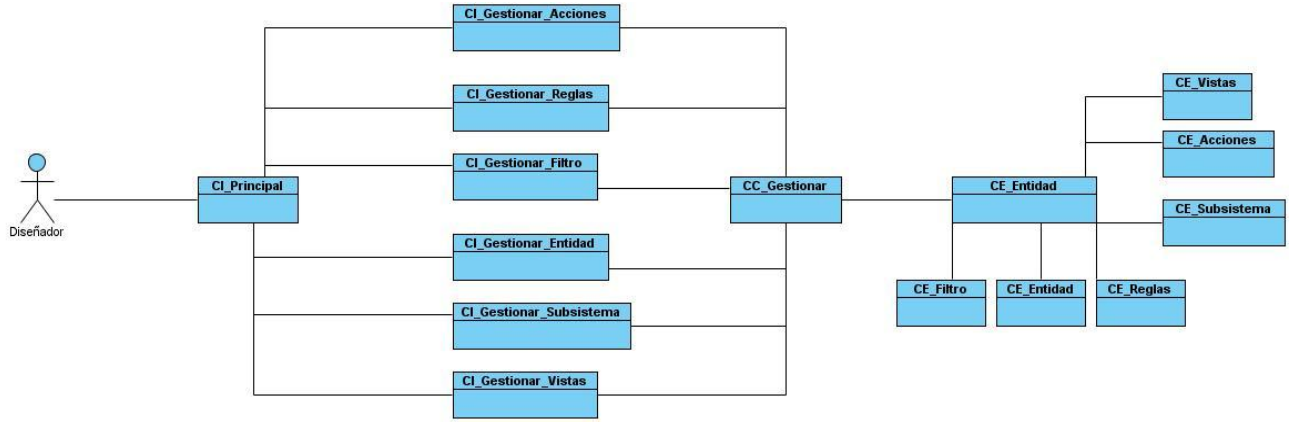


Figura 7: Diagrama de clases de análisis general.

Diagramas de clases de análisis para el Módulo: Gestionar Subsistema.

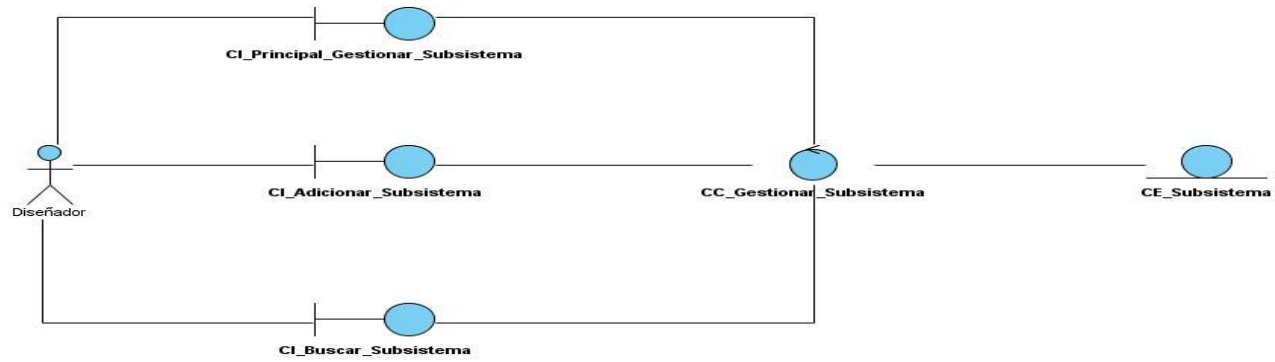


Figura 8: Diagrama de clases de análisis para el Módulo: Gestionar Subsistema.

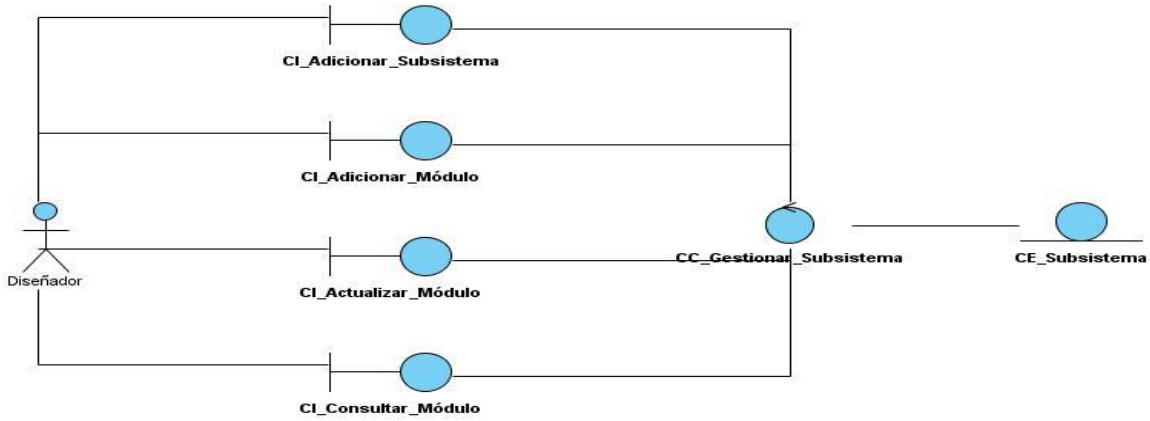


Figura 9: Diagrama de clases de análisis para el Módulo: Gestionar Subsistema / Caso de Uso: Adicionar Subsistema.

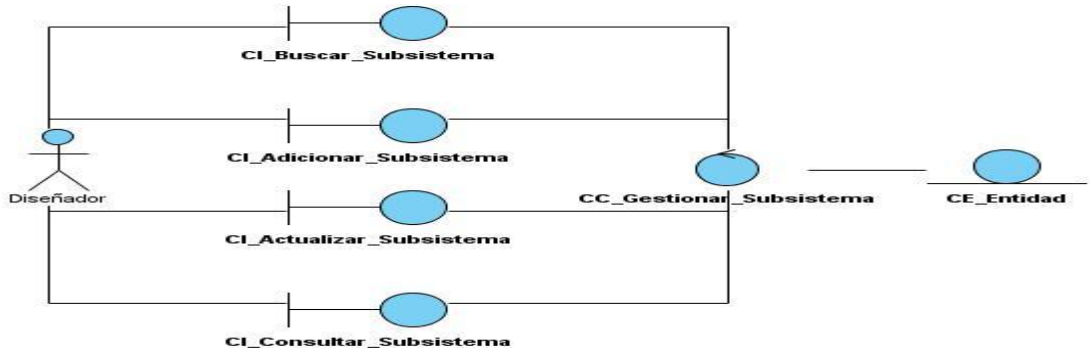


Figura 10: Diagrama de clases de análisis para el Módulo: Gestionar Subsistema / Caso de Uso: Buscar Subsistema.

2.6. Diagramas de interacción (Colaboración)

Diagrama que muestra la interacción de los roles organizadamente, basándose en los objetos que toman parte ella y los enlaces entre objetos con los mensajes que intercambian. [Conceptos básicos en un Diagrama de Colaboración]

Diagramas de interacción (Colaboración) para el Caso de Uso: Autenticar Usuario.

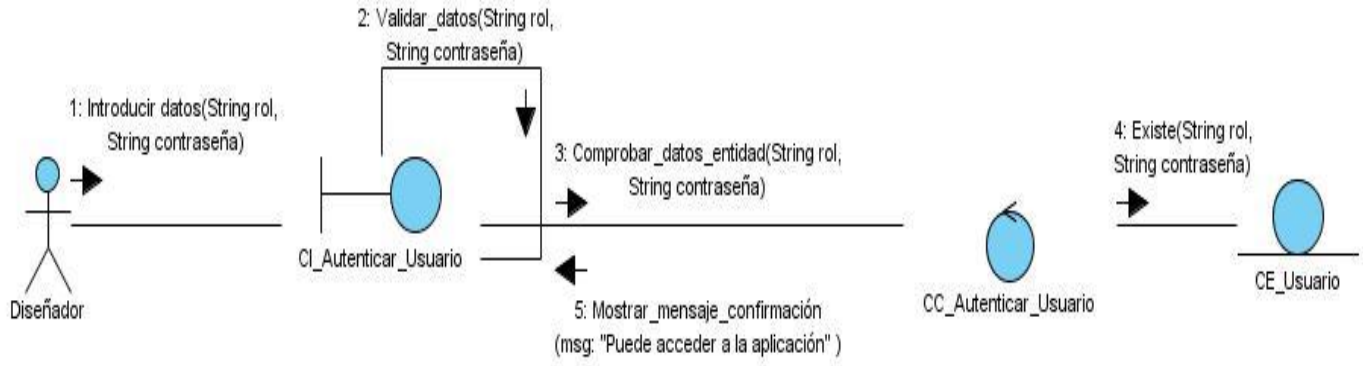


Figura 11: Diagrama de colaboración: Autenticar usuario.

Diagramas de interacción (Colaboración) para el Módulo: Gestionar Subsistema.

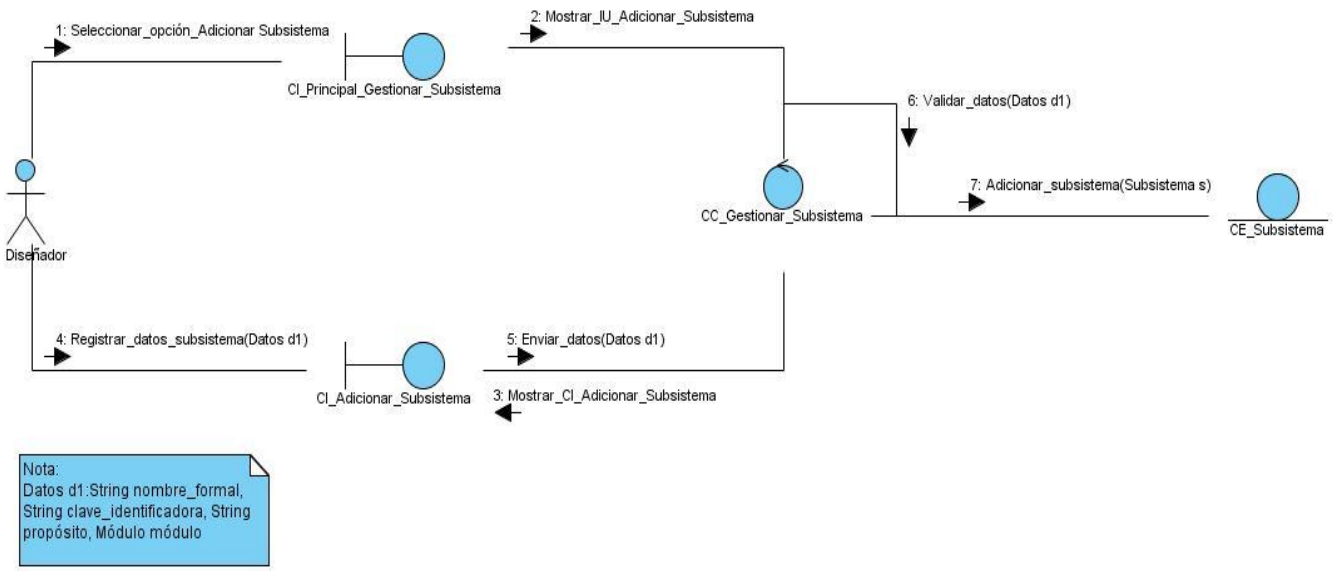


Figura 12: Diagrama de colaboración para el Módulo: Gestionar Subsistema / Caso de Uso: Adicionar Subsistema.

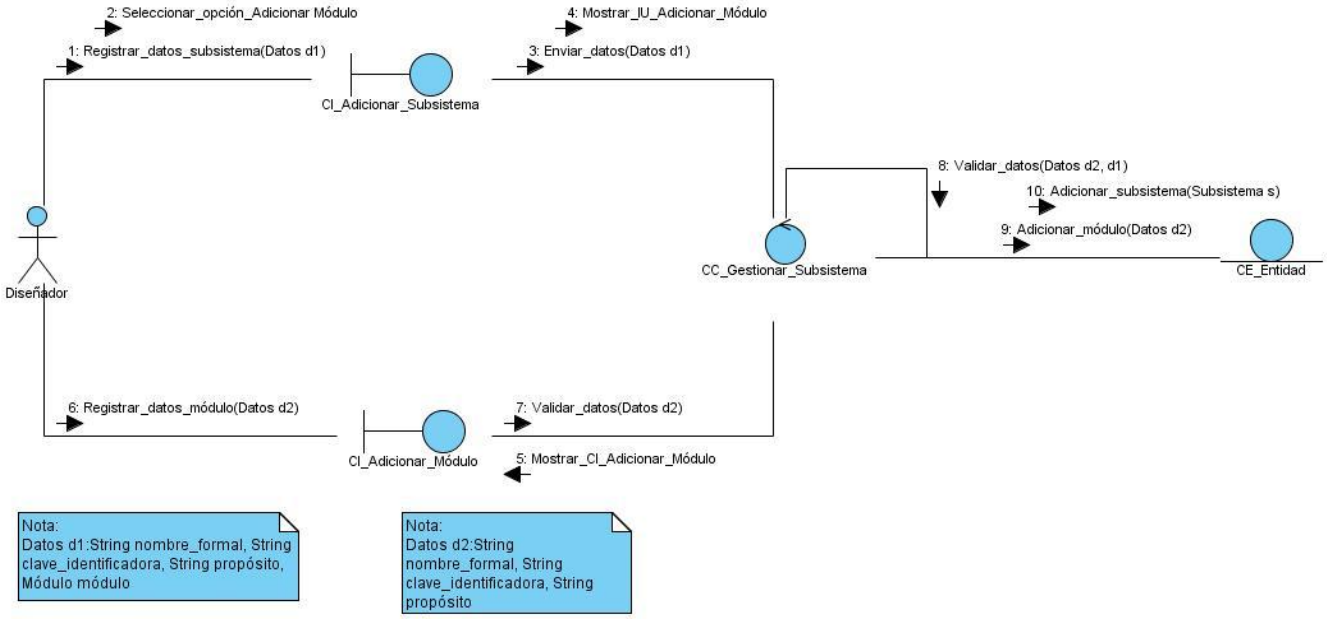


Figura 13: Diagrama de colaboración para el Módulo: Gestionar Subsistema / Caso de Uso: Adicionar Subsistema / Escenario: Adicionar Módulo.

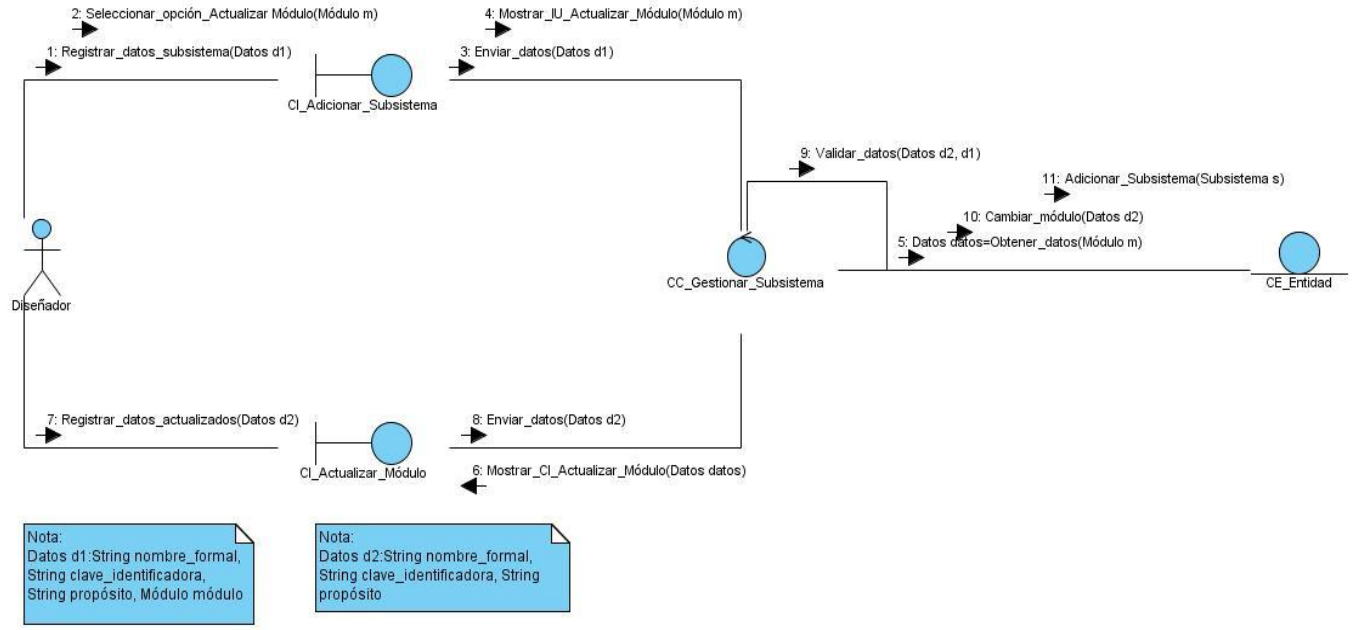


Figura 14: Diagrama de colaboración para el Módulo: Gestionar Subsistema / Caso de Uso: Adicionar Subsistema / Escenario: Actualizar Módulo.

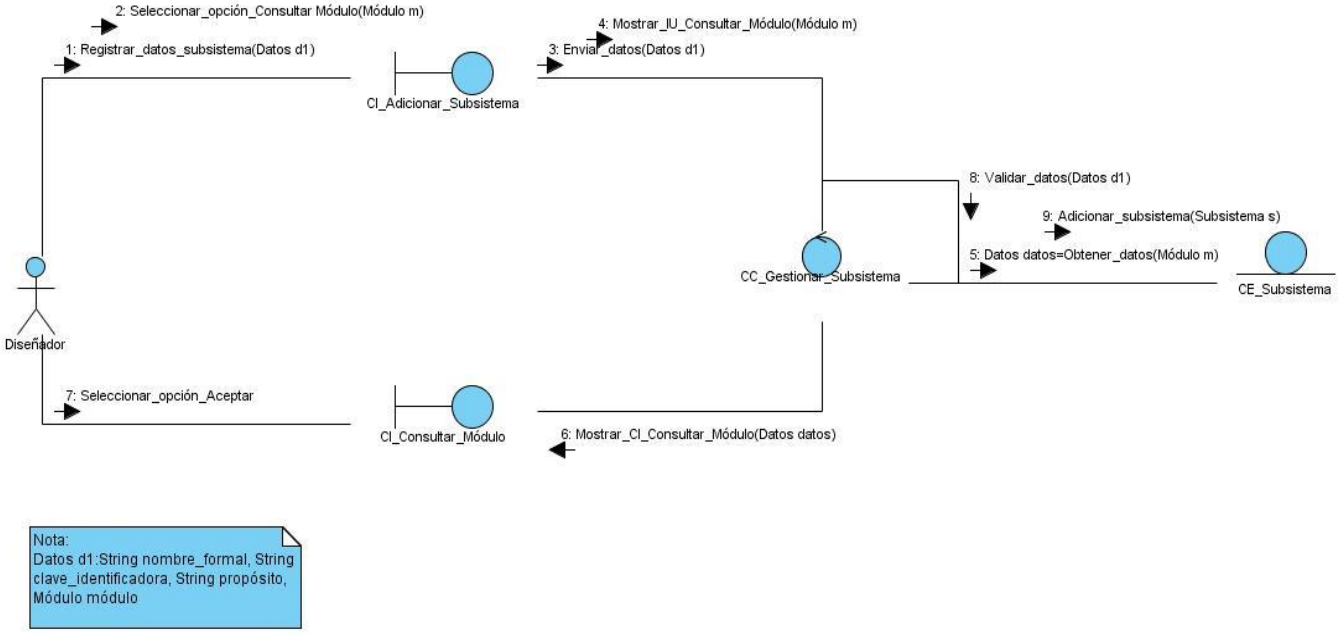


Figura 15: Diagrama de colaboración para el Módulo: Gestionar Subsistema / Caso de Uso: Adicionar Subsistema / Escenario: Consultar Módulo.

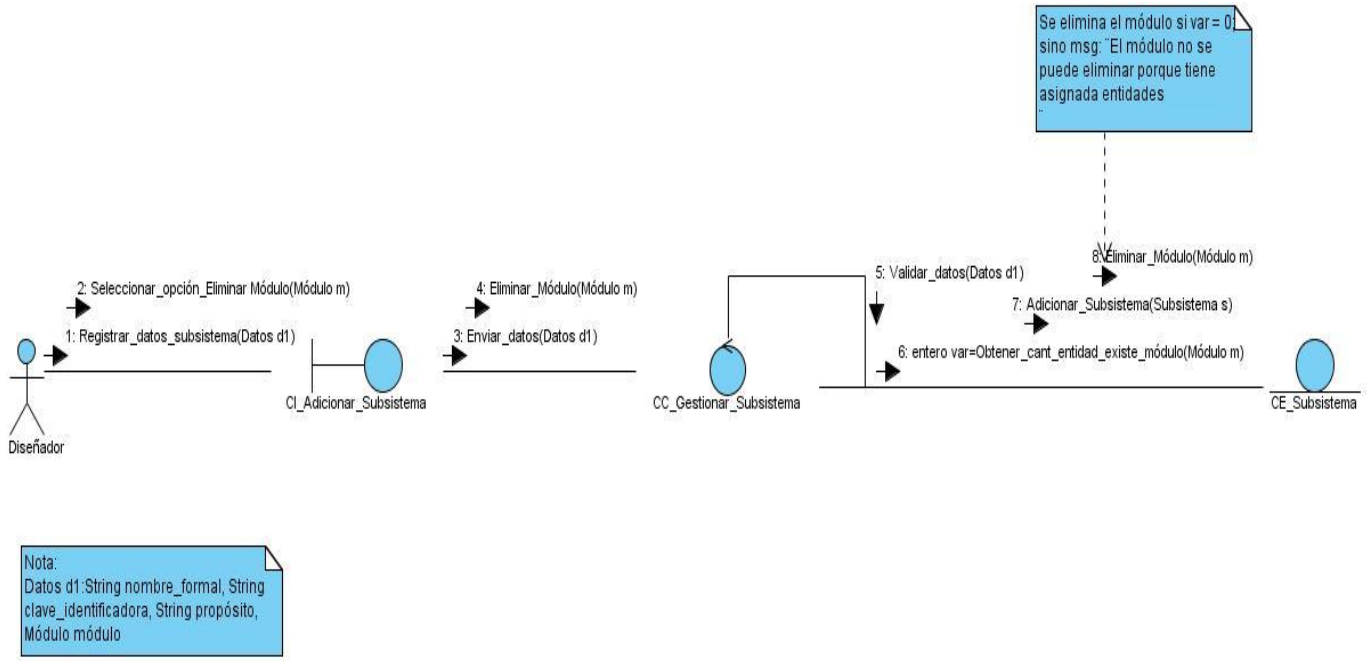


Figura 16: Diagrama de colaboración para el Módulo: Gestionar Subsistema / Caso de Uso: Adicionar Subsistema / Escenario: Eliminar Módulo.

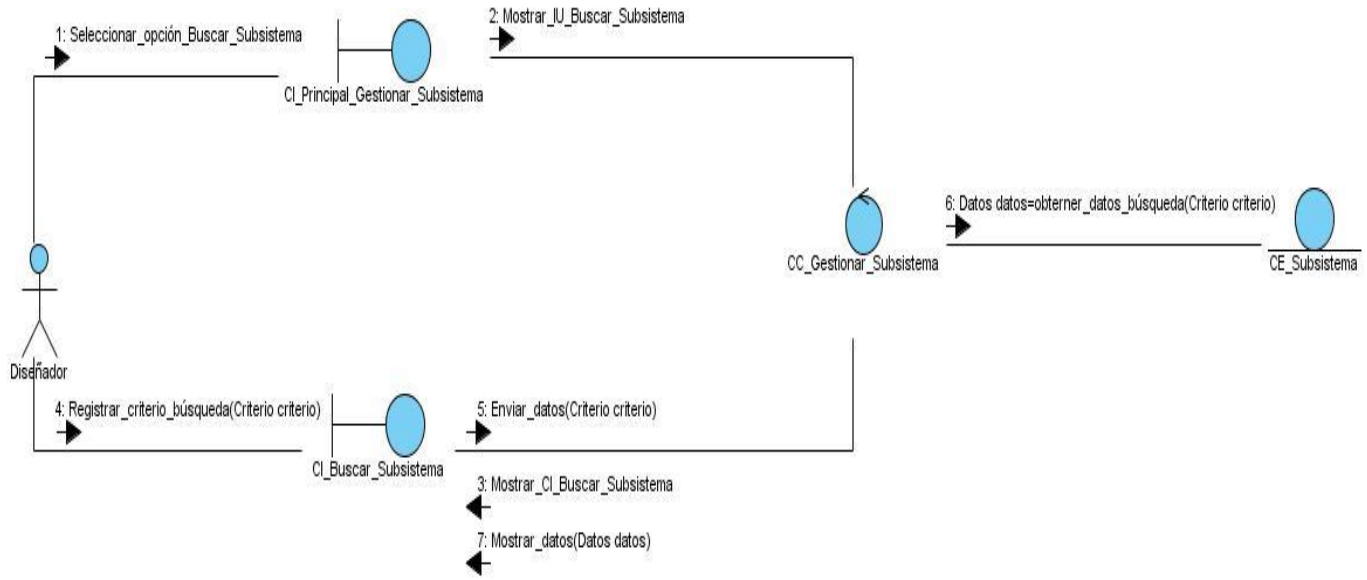


Figura 17: Diagrama de colaboración para el Módulo: Gestionar Subsistema / Caso de Uso: Buscar Subsistema.

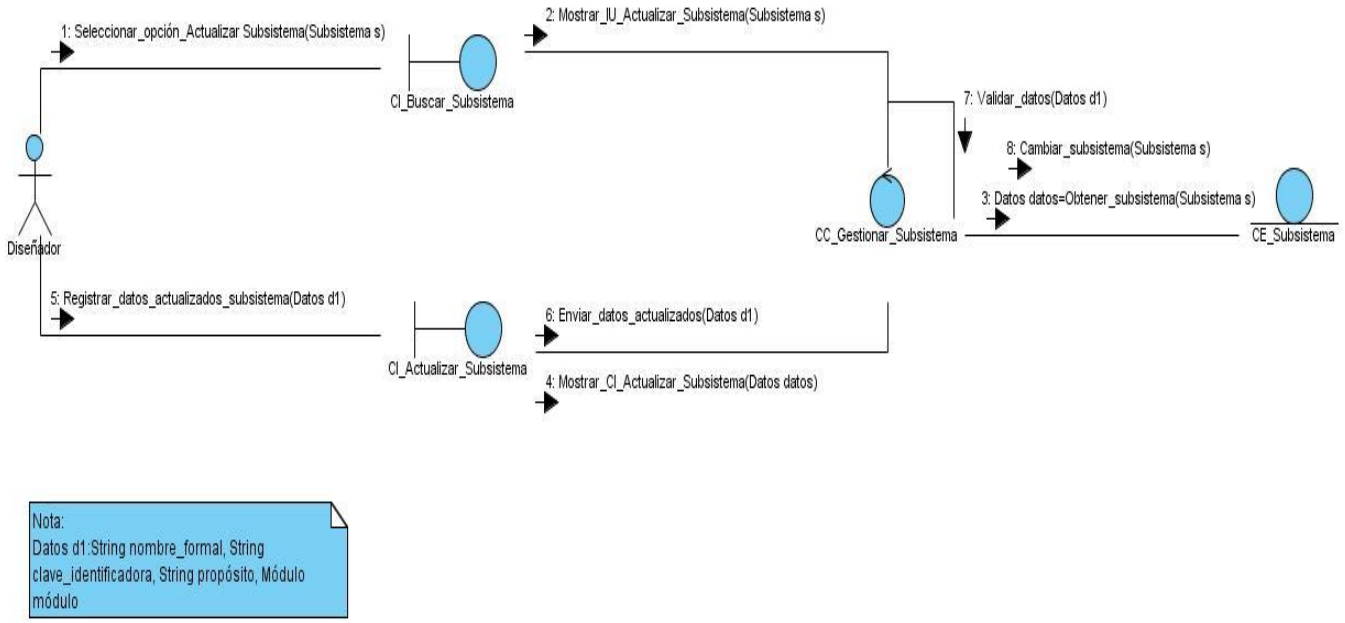


Figura 18: Diagrama de colaboración para el Módulo: Gestionar Subsistema / Caso de Uso: Buscar Subsistema / Escenario: Actualizar Subsistema.

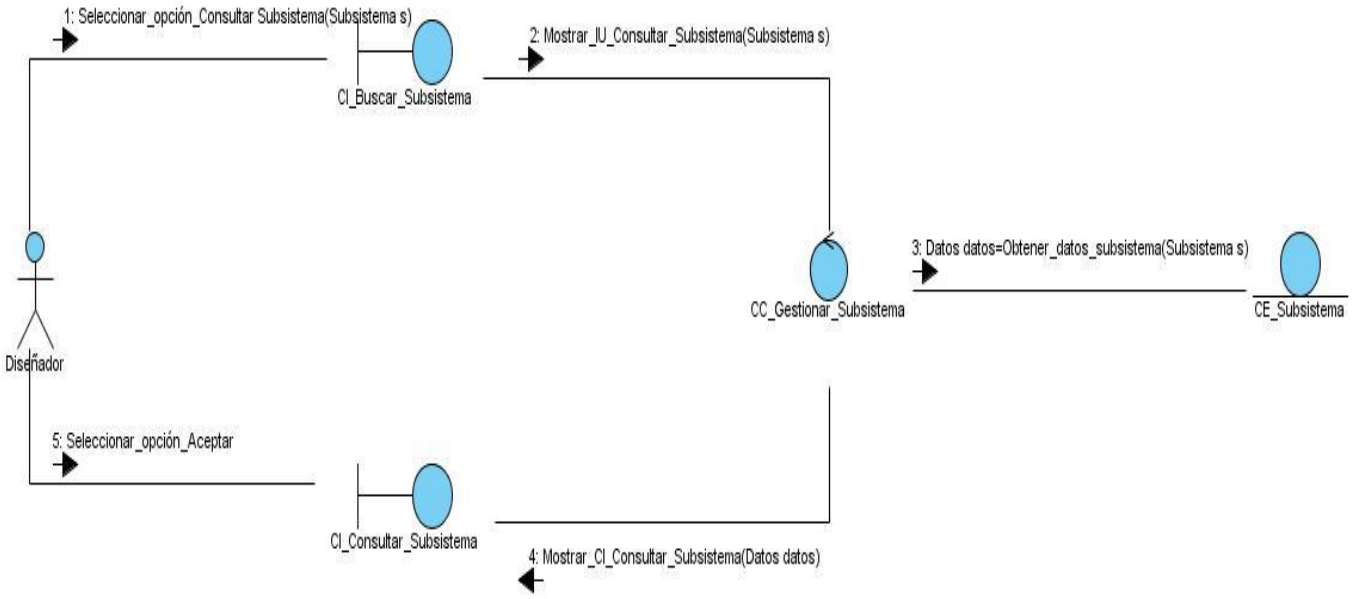


Figura 19: Diagrama de colaboración para el Módulo: Gestionar Subsistema / Caso de Uso: Buscar Subsistema / Escenario: Consultar Subsistema.

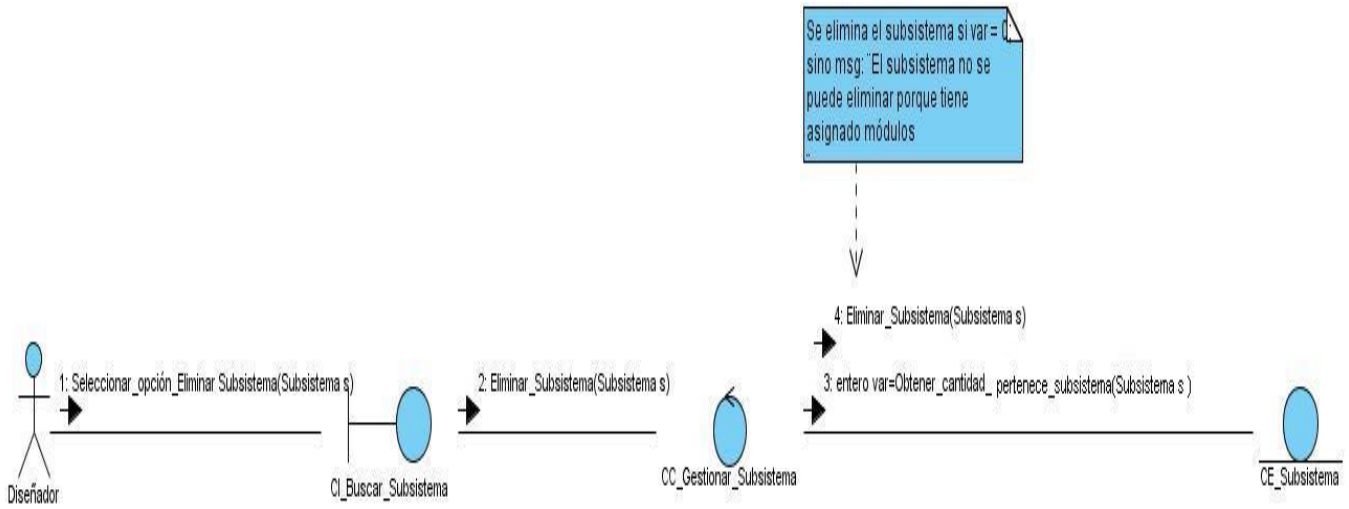


Figura 20: Diagrama de colaboración para el Módulo: Gestionar Subsistema / Caso de Uso: Buscar Subsistema / Escenario: Eliminar Subsistema.

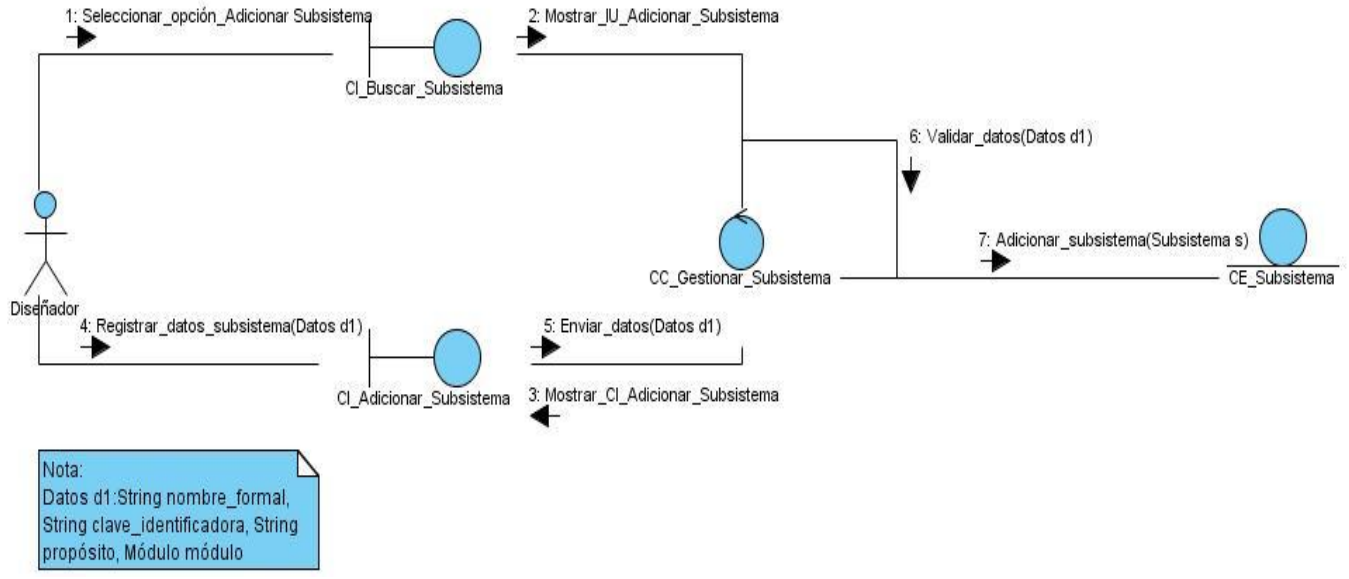


Figura 21: Diagrama de colaboración para el Módulo: Gestionar Subsistema / Caso de Uso: Buscar Subsistema / Escenario: Adicionar Módulo.

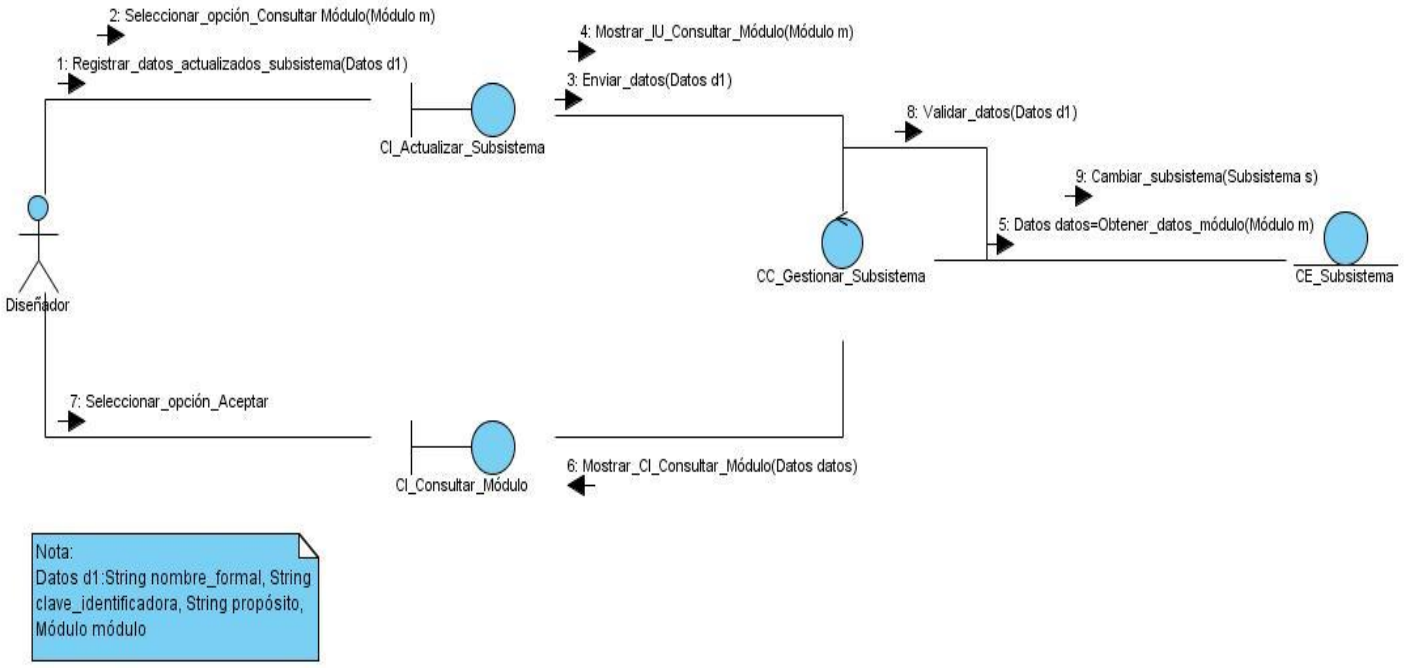


Figura 22: Diagrama de colaboración para el Módulo: Gestionar Subsistema / Caso de Uso: Buscar Subsistema / Escenario: Consultar Módulo.

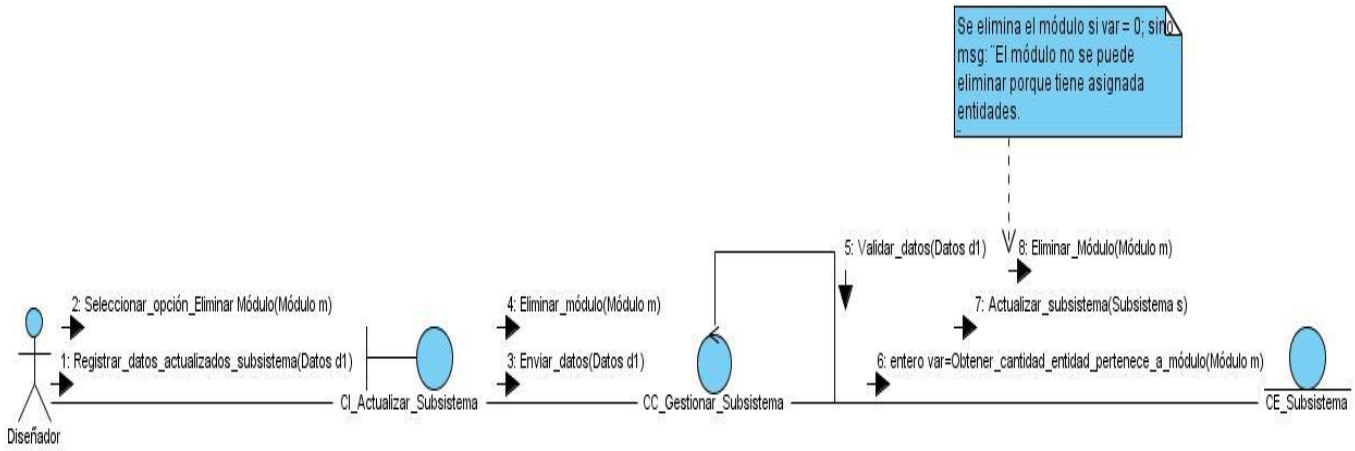


Figura 23: Diagrama de colaboración para el Módulo: Gestionar Subsistema / Caso de Uso: Buscar Subsistema / Escenario: Eliminar Módulo.

2.7. Conclusiones Parciales

Se realizó el análisis de los requisitos funcionales del sistema y se generaron los artefactos pertenecientes a este flujo de trabajo, además de incluir una breve descripción de los artificios.

Al realizarse el análisis se logró una mejor comprensión del sistema y se deja todo listo para la etapa de diseño.

Capítulo 3: Diseño y Validación

3.1. Introducción

En este capítulo se recogen los elementos considerados para elaborar el diseño del sistema para la gestión automática de entidades del negocio. Se describe la arquitectura y patrones a usar durante la realización de los artefactos pertenecientes al diseño.

El capítulo recoge además, las validaciones, porque es de suma importancia para la ingeniería de software certificar todo lo realizado. Medir permite tener una visión profunda y clara del trabajo que se está realizando. Se aplican métricas de diseño orientado a objeto y se analizan los resultados para determinar la calidad del trabajo.

3.2. Descripción de la Arquitectura

Diseño de las capas lógicas

Es necesario tener una organización para alcanzar una mejor comprensión a la hora de implementar o desplegar el sistema. Es importante agrupar en los subsistemas, los módulos que estén estrechamente relacionados según las funcionalidades que realizan para separarlos por diferentes capas lógicas teniendo en cuenta la naturaleza de los mismos.

Capa de presentación: Capa que se divide en dos porciones. La primera es una subcapa del lado del servidor, que es la encargada de recibir los pedidos de la interfaz de usuario, controla el flujo de presentación del sistema y envía las respuestas correspondientes a la interfaz de usuario; está relacionada con la capa de negocios y de dominio. Y la segunda subcapa que está en el lado del cliente, que utiliza los componentes visuales de Java Script para manejar los eventos y validaciones del lado del cliente.

Capa de negocios: Esta capa se fracciona en varias subcapas según sean necesarias y estén relacionadas con el negocio, se tienen dos principales: Fachada es una de las subcapas principales, se exponen todas las funcionalidades que la capa de presentación necesite, y esta invoca los métodos de la subcapa de desarrollo del negocio. En la capa de desarrollo del negocio se implementa el negocio de los módulos en cuestión, y se accederá si es necesario a la capa de acceso a datos, otras capas de negocio, y / o a la capa de dominio.

Capa de acceso a datos: Capa en la que se implementan los métodos encargados de interactuar con el gestor de base de datos y tiene dependencia solo de la capa de dominio.

Capa de dominio: Se declaran todas las clases que representan entidades del negocio. Estas clases de dominio están presentes en todas las capas anteriormente descritas. A continuación se muestra una figura con la estructura de las capas lógicas. [Pérez Betancourt. (2009)]

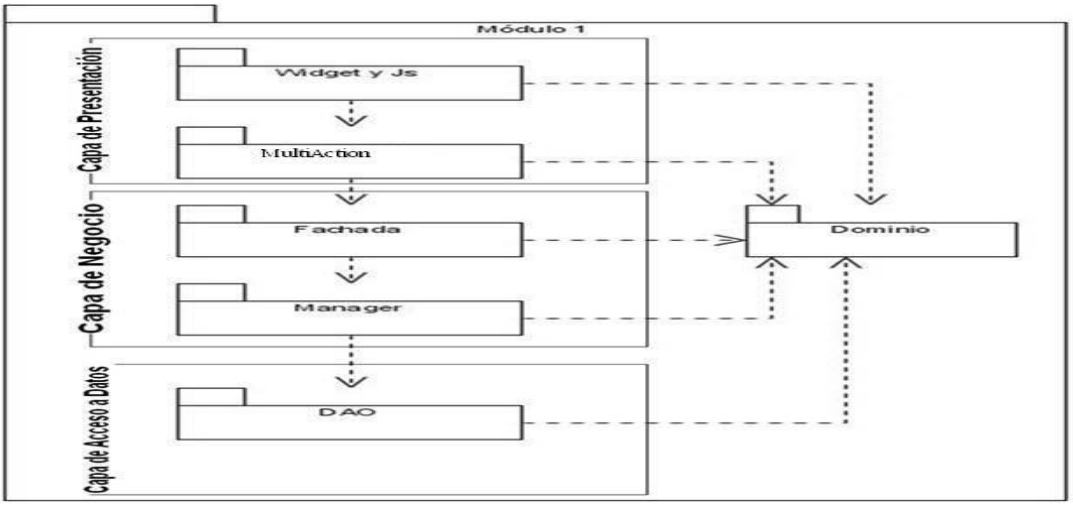


Figura 24: Estructura de Capas Lógicas del sistema.

Se muestra una figura de las capas lógicas y los marcos de trabajo que se usan en cada una.

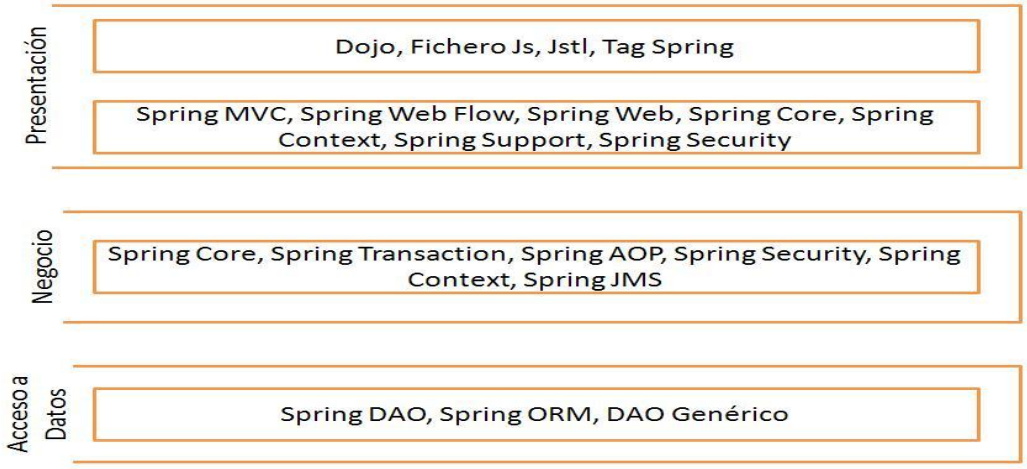


Figura 25: Estructura de Capas Lógicas y marcos de trabajo usados.

3.3. Diseño de la solución

3.3.1. Patrones

Cada patrón describe un problema que ocurre de forma repetitiva en un ambiente determinado y luego detalla el núcleo de la solución de ese problema, de manera que se pueda usar las veces que sea necesario.

Patrones de asignación de responsabilidades GRASP[5]

En los patrones GRASP se codifican algunos de los principios que se aplican al preparar los diagramas de interacción.

Experto: La aplicación de este patrón permite a cada clase, desarrollar tareas que pueden realizar según la información que poseen.

Creador: Permite crear instancias de otras clases en correspondencia con la responsabilidad dada. Se logra conservar el encapsulamiento porque los objetos consiguen valerse de su propia información para realizar lo que se desea.

Bajo acoplamiento: Soluciona el inconveniente de dar soporte a una dependencia escasa y a un aumento de la reutilización.

Alta cohesión: Se utiliza para mantener la complejidad dentro de los límites manejables.

El diseño obtenido cumple con los patrones: Experto, Creador, Bajo acoplamiento y Alta cohesión que permite la colaboración entre los elementos del diseño, sin que se vea afectada la reutilización y el entendimiento cuando se encuentran aislados. [Pérez Betancourt. (2009)]

3.3.2. Patrones estructurales

Facade: Utilizar este patrón provee una interfaz unificada, sencilla que hace de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas. Ayuda a la hora de entender y usar una biblioteca de software. Esto es posible porque el facade invoca métodos convenientes para tareas

5 GRASP: (En inglés) General Responsibility Assignment Software Patterns ((patrones generales de software para asignar responsabilidades), son patrones generales de software para asignación de responsabilidades, aunque se considera que más que patrones, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software.

comunes, puede reducir la dependencia de código externo en los trabajos internos, y permite una mayor flexibilidad en el desarrollo de sistemas. [Pérez Betancourt. (2009)]

3.3.3. Patrones de comportamiento

Command: Parametriza los objetos por las acciones que realizan. Permite especificar, administrar y ejecutar solicitudes en tiempos distintos. Puede guardar un estado que permita deshacer la ejecución del comando. Soporta la capacidad de generar bitácoras que permitan la recuperación del estado en caso de fallar el sistema. Facilita la estructuración del sistema en torno a operaciones de alto nivel construidas con base en operaciones primitivas o de bajo nivel. Un comando desata el objeto invocador del receptor. Permite que las acciones sean objetos de primera clase y se puedan agrupar comandos de uso frecuente en comandos compuestos. [Garcia, Walter y Ospina, Gustavo]

3.3.4. Patrones de acceso a datos

DAO: Permite acceder a la fuente de datos y encapsular los objetos clientes, ocultando la fuente y el modo de acceder a ella. Los DAOs deben implementar los métodos que se declaran en la interfaz (InterfaceDAO). Además pueden efectuar otros métodos que no estén en la interfaz. Consiente el acceso a reglas de validación, esto es posible porque tiene capacidad de especificar relaciones entre tablas. [Pérez Betancourt. (2009)]

3.4. Modelo de diseño

Diagrama de clases

Clase de diseño: son abstracciones de clases claramente aprovechables en la implementación del software. Sus relaciones tienen un significado directo en el lenguaje de programación. En una clase del diseño, los métodos son los mismos que en una clase implementada.

Extensiones de UML para diseño web:

Server Page (página servidora): representa la página web que se ejecuta en el servidor.

Client Page (página cliente): una instancia de página cliente, es una página web, con formato HTML.

Form (formulario): colección de elementos de entrada que son parte de una página cliente.

<<Build>>: representa una asociación especial que relaciona las páginas clientes con las páginas servidor.

<<Link>>: expresa las asociaciones más comunes entre las páginas, en este caso la del hipervínculo.

<<Submit>>: es la relación que se establece siempre entre una página servidor y un formulario.

Una clase del diseño puede ser activa, en el sentido de que los objetos instanciados pueden mantener su propio hilo de ejecución concurrente con otros objetos.

Diagrama de clases del diseño: describe gráficamente las especificaciones de las clases del software y de las interfaces en una aplicación. El mismo contiene información como clases, asociaciones y atributos; interfaces con sus operaciones y constantes; métodos; información sobre los tipos de los atributos; navegabilidad; dependencias.

Nota: La capa de negocio, la de acceso a datos y la capa de dominio son las mismas para el caso de uso Buscar y Adicionar para todos los módulos.

Diagrama de clases de diseño general.

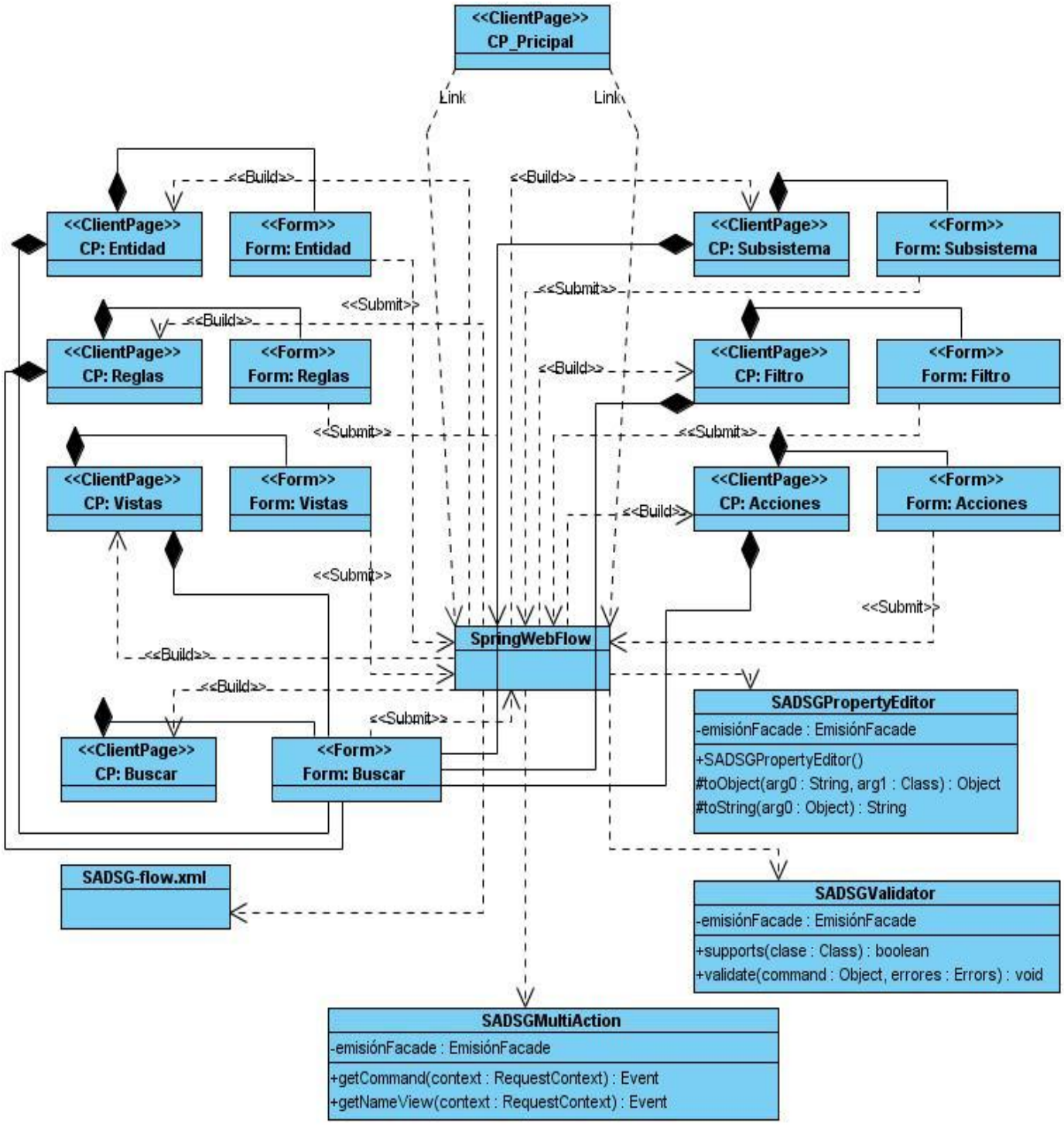


Figura 26: Diagrama de la capa de presentación general.

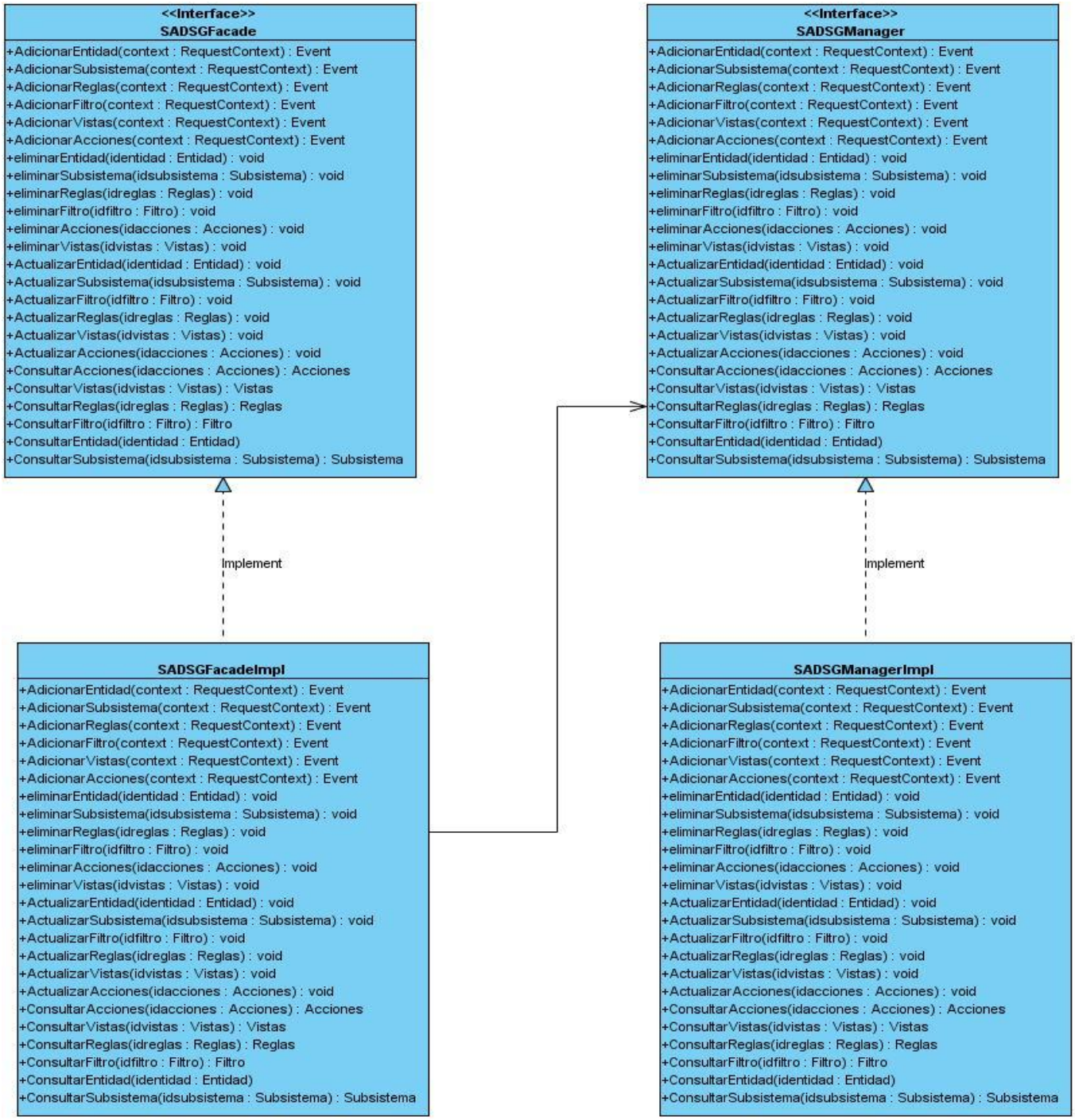


Figura 27: Diagrama de la capa de negocio general.

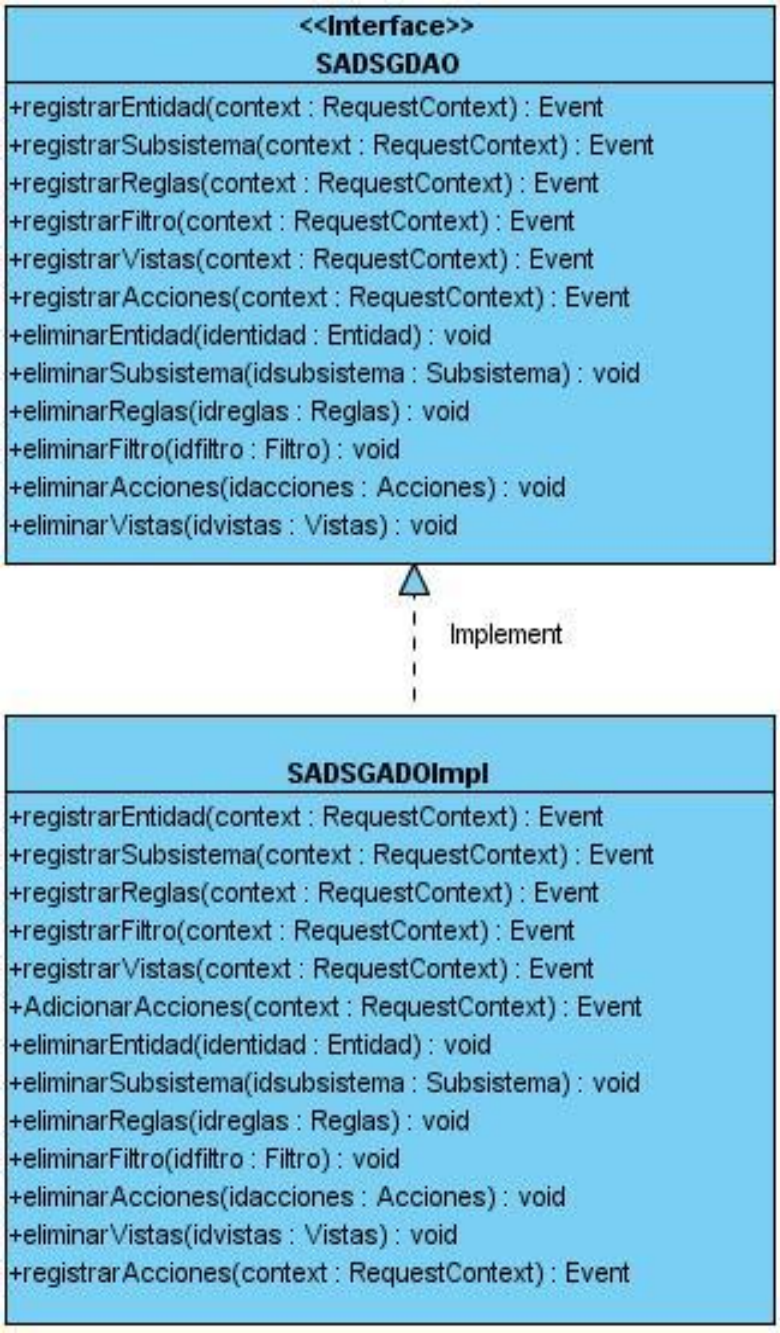


Figura 28: Diagrama de la capa de acceso a dato general.

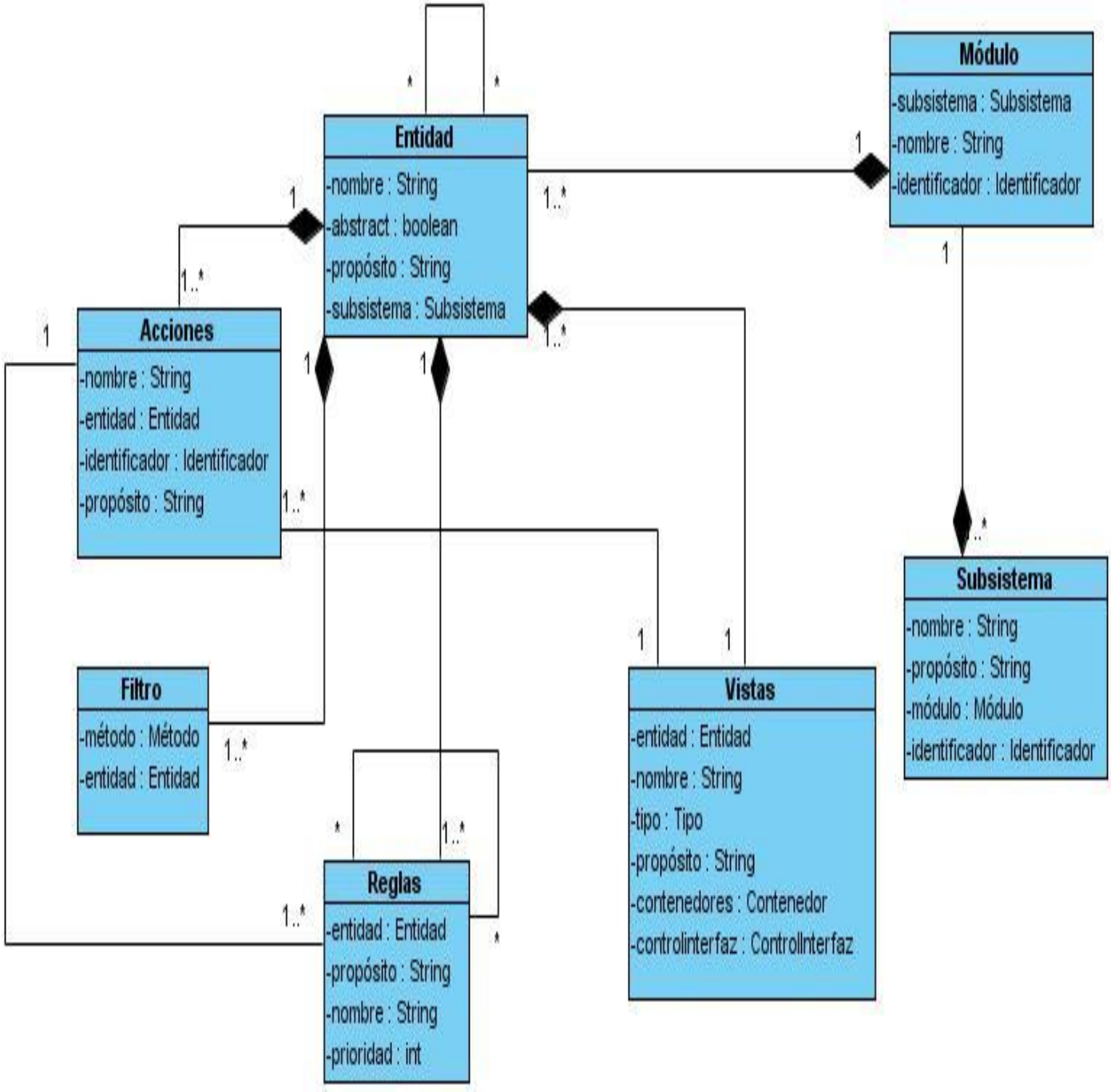


Figura 29: Diagrama de la capa de dominio general.

Diagrama de diseño para el Módulo: Gestionar Subsistema

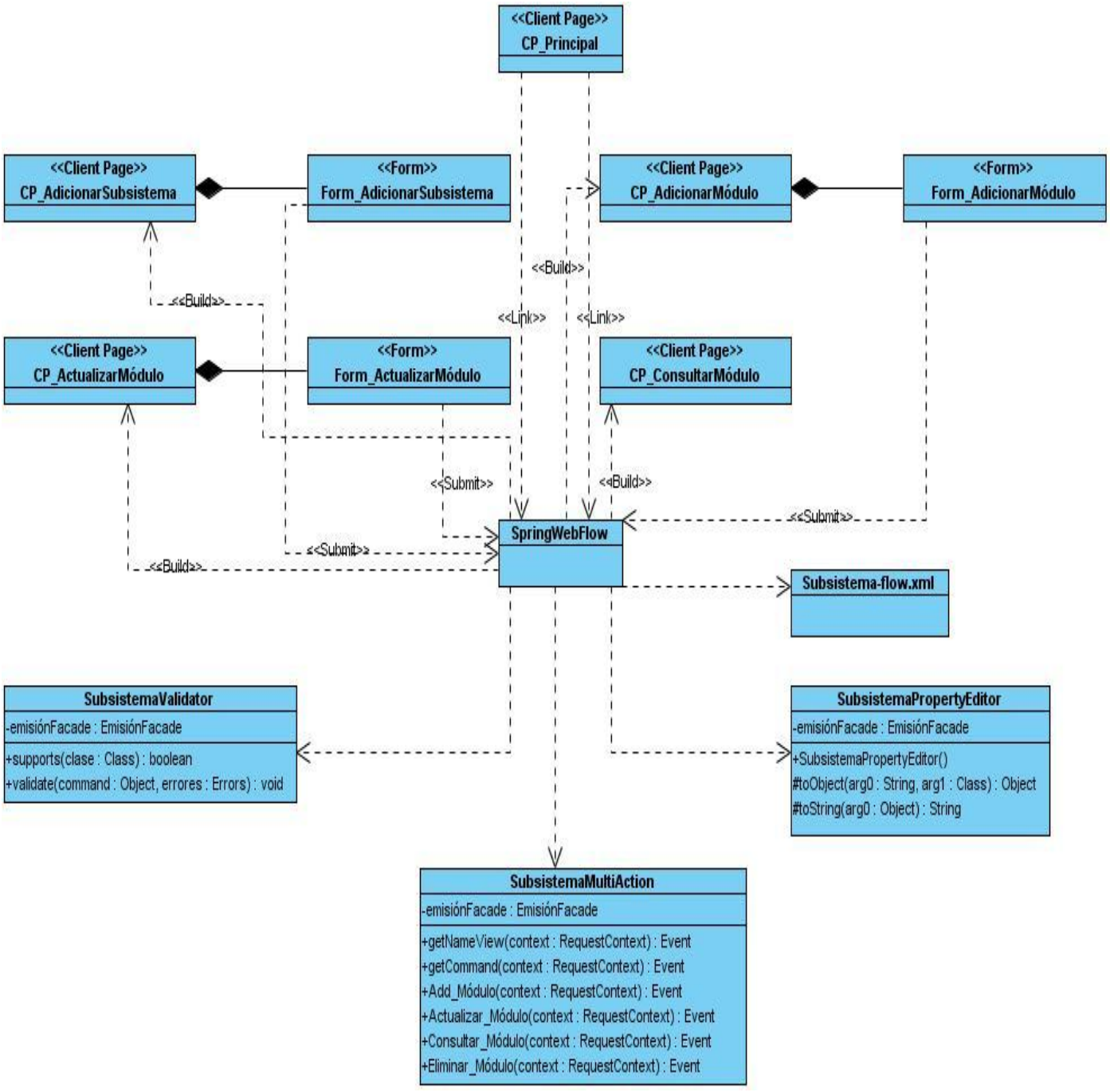


Figura 30: Diagrama de la capa de presentación para el Módulo: Gestionar Subsistema / Caso de Uso: Adicionar Subsistema.

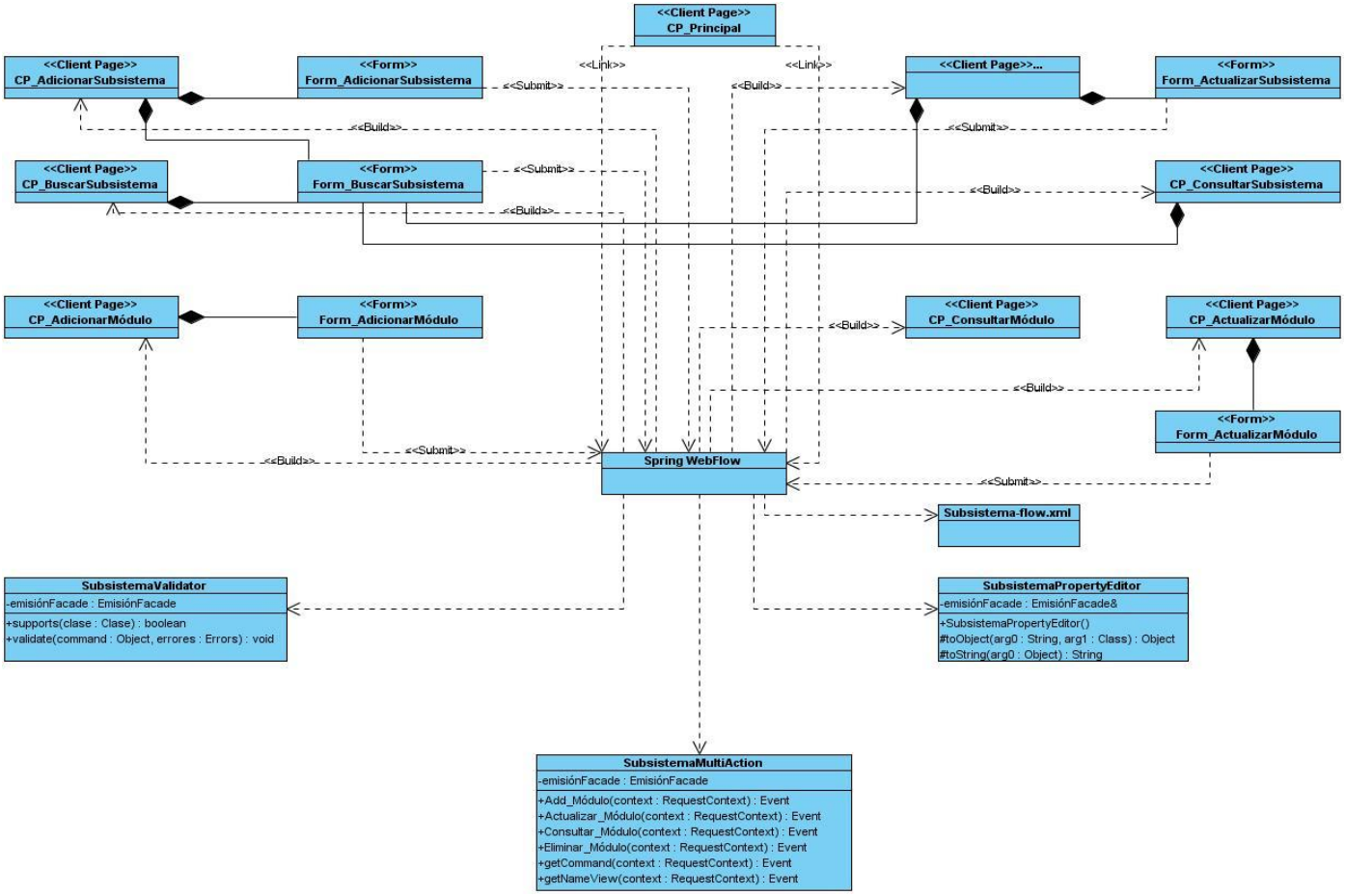


Figura 31: Diagrama de la capa de presentación para el Módulo: Gestionar Subsistema / Caso de Uso: Buscar Subsistema.

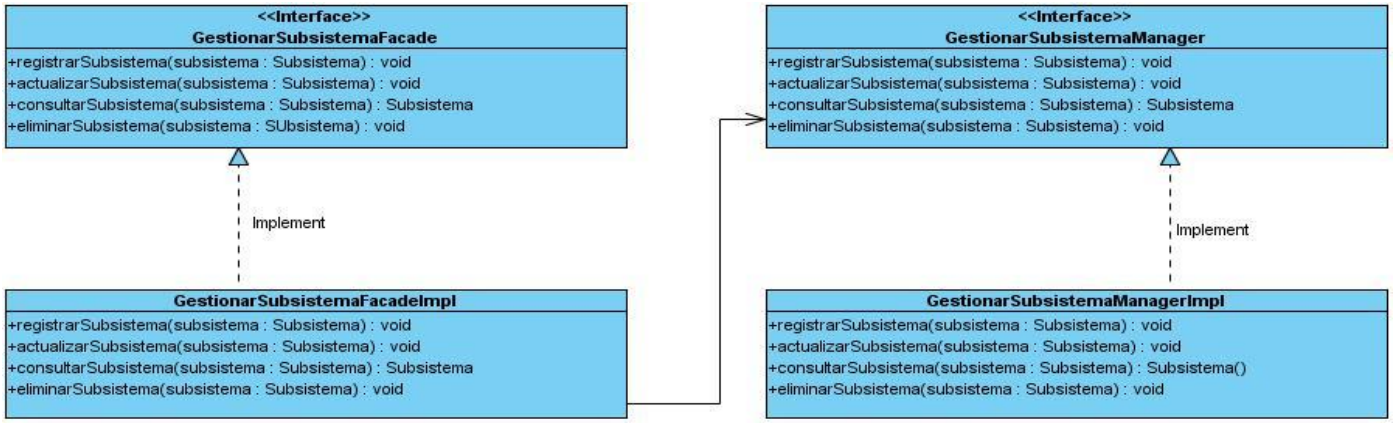


Figura 32: Diagrama de la capa de negocio para el Módulo: Gestionar Subsistema / Caso de Uso: Adicionar Subsistema.

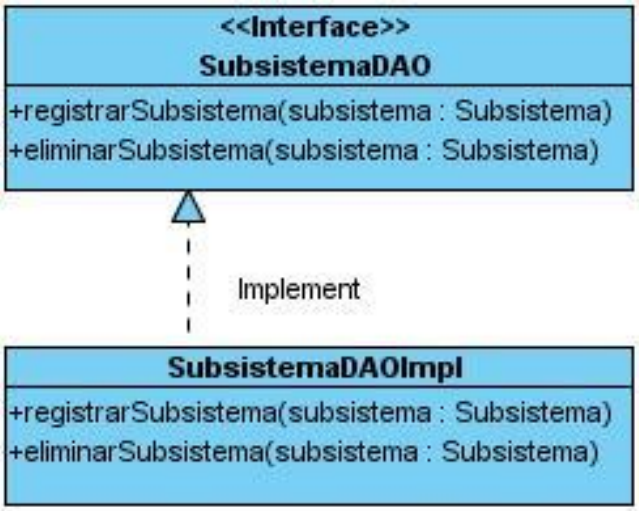


Figura 33: Diagrama de la capa de acceso a datos para el Módulo: Gestionar Subsistema Caso de Uso: Adicionar Subsistema.

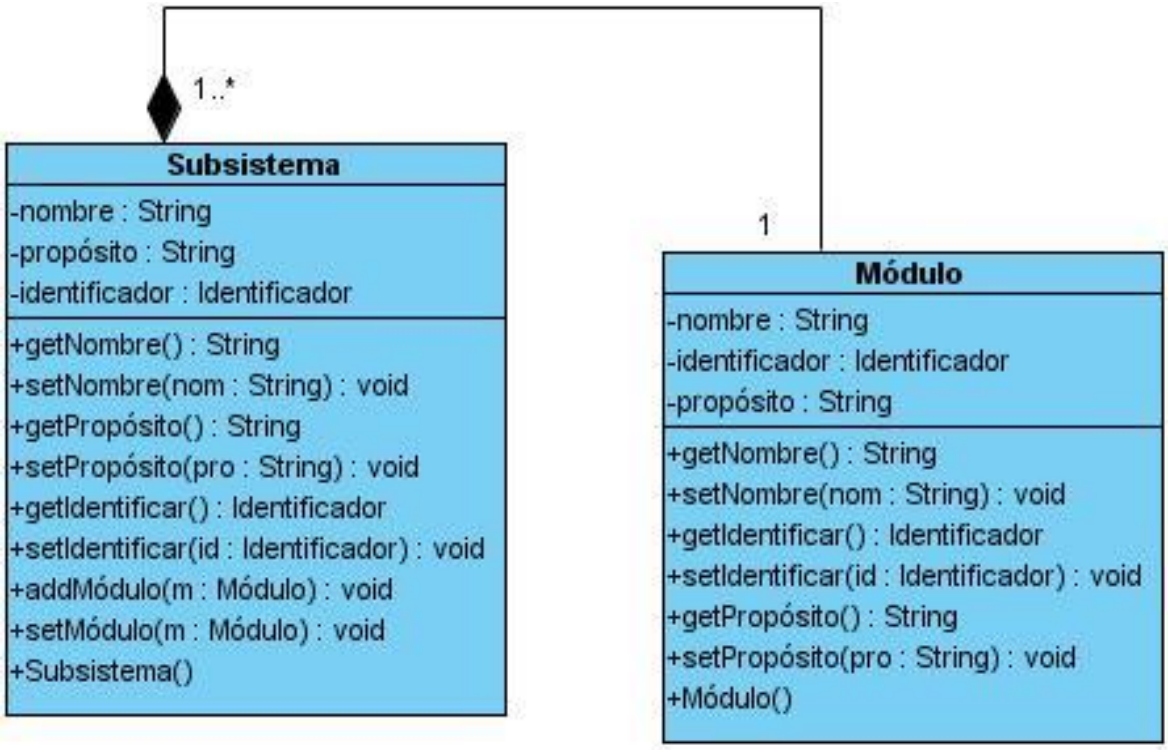


Figura 34: Diagrama de la capa de dominio para el Módulo: Gestionar Subsistema / Caso de Uso: Adicionar Subsistema.

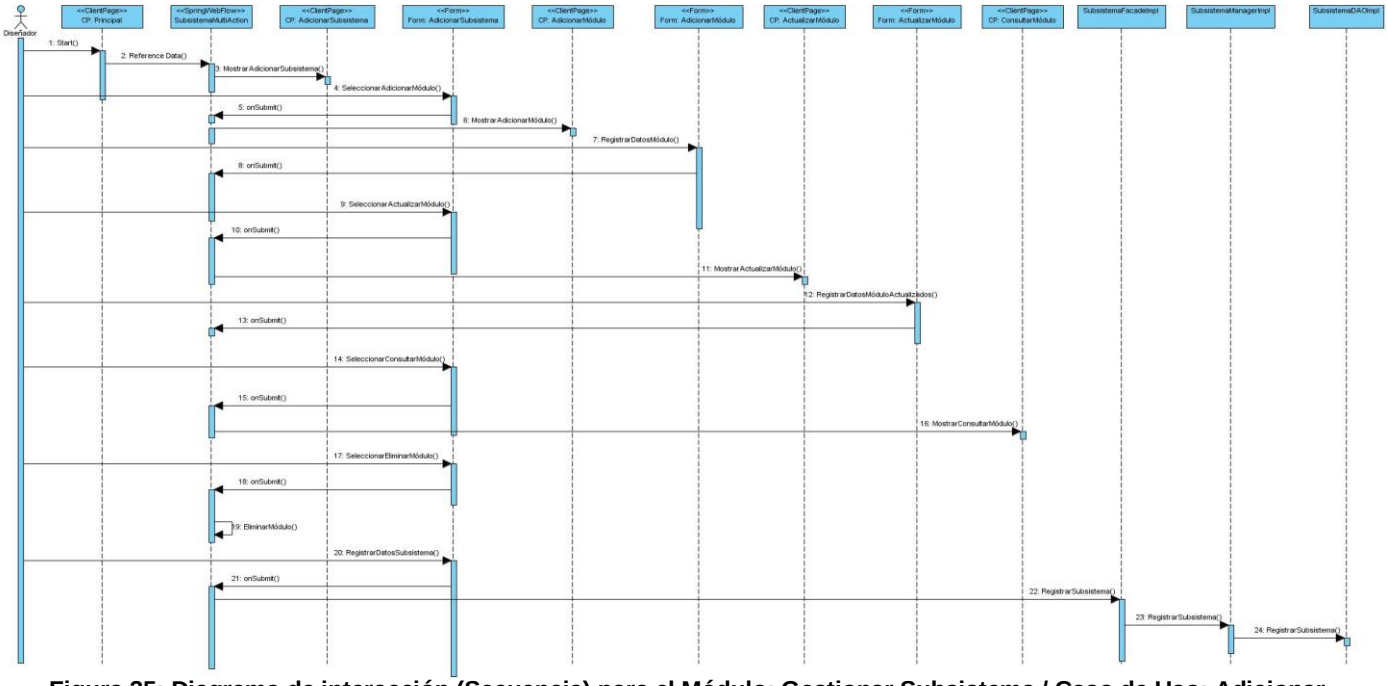


Figura 35: Diagrama de interacción (Secuencia) para el Módulo: Gestionar Subsistema / Caso de Uso: Adicionar Subsistema.

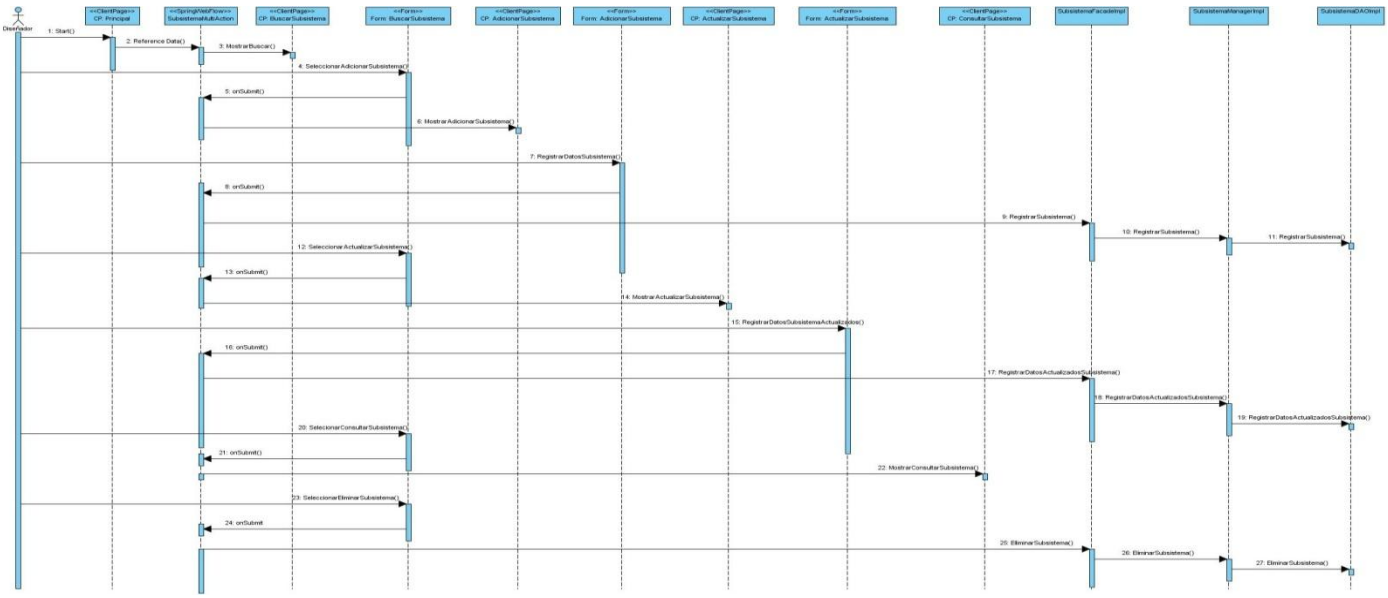


Figura 36: Diagrama de interacción (Secuencia) para el Módulo: Gestionar Subsistema / Caso de Uso: Buscar Subsistema.

3.5. Diagrama de paquete

Este diagrama muestra cómo el sistema está dividido en agrupaciones lógicas y cómo estas se relacionan. Cada paquete puede asignarse a un individuo o a un grupo y las relaciones entre ellos pueden indicar el orden de desarrollo requerido. [Spark Systems. (2000-2007)]

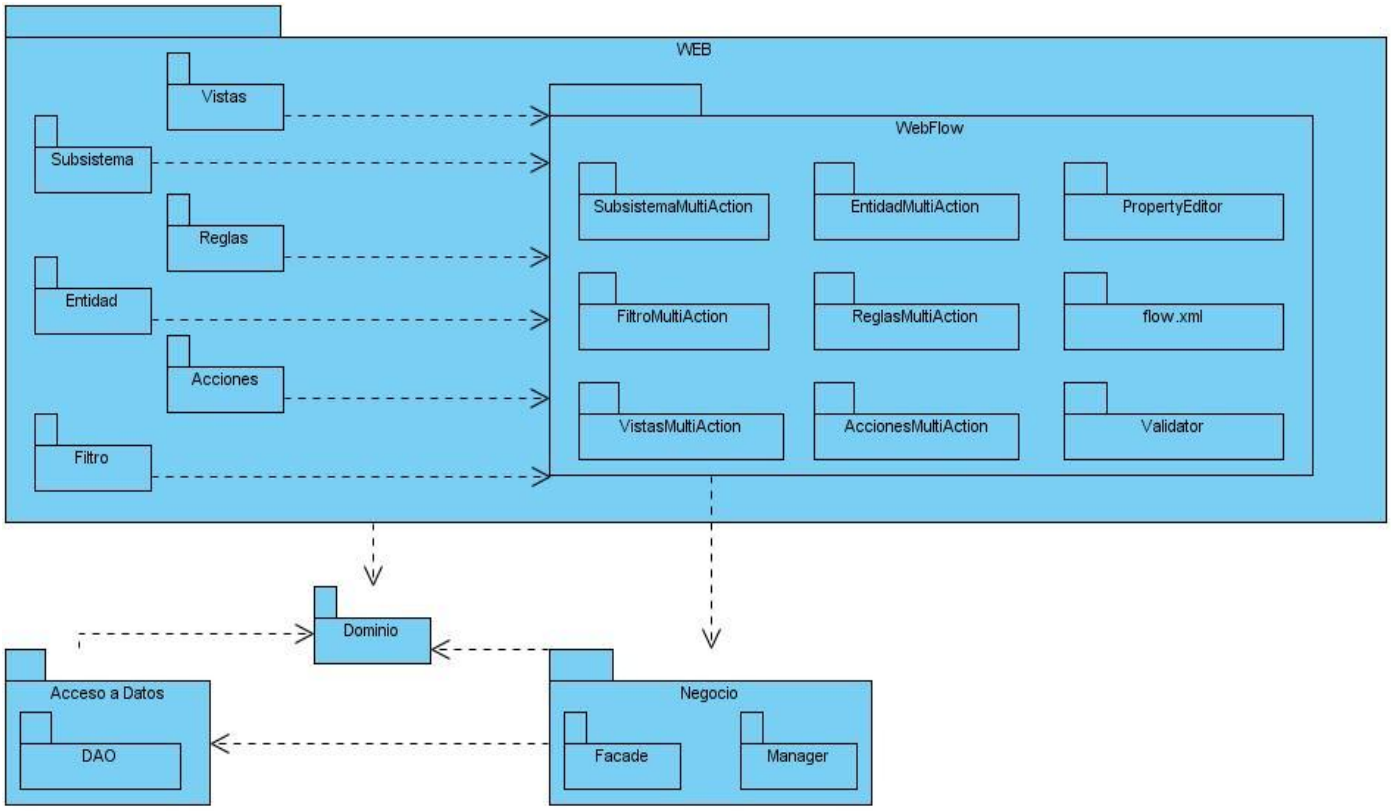


Figura 37: Diagrama de paquete.

3.6. Modelo de Datos

Un ***modelo de datos*** es un conjunto de conceptos, reglas y convenciones que describen y en ocasiones manipulan los datos que se desea almacenar en la base de datos. Este modelo está conformado por dos componentes: estática, relacionada con el lenguaje de definición de datos (LDD) y hace referencia a la estructura; dinámica, correspondida con el lenguaje de manipulación de datos (LMD) y operaciones que se pueden realizar sobre cada objeto.

El modelo de datos está normalizado porque cumple con las formas normales aplicadas a las tablas de la base de datos.

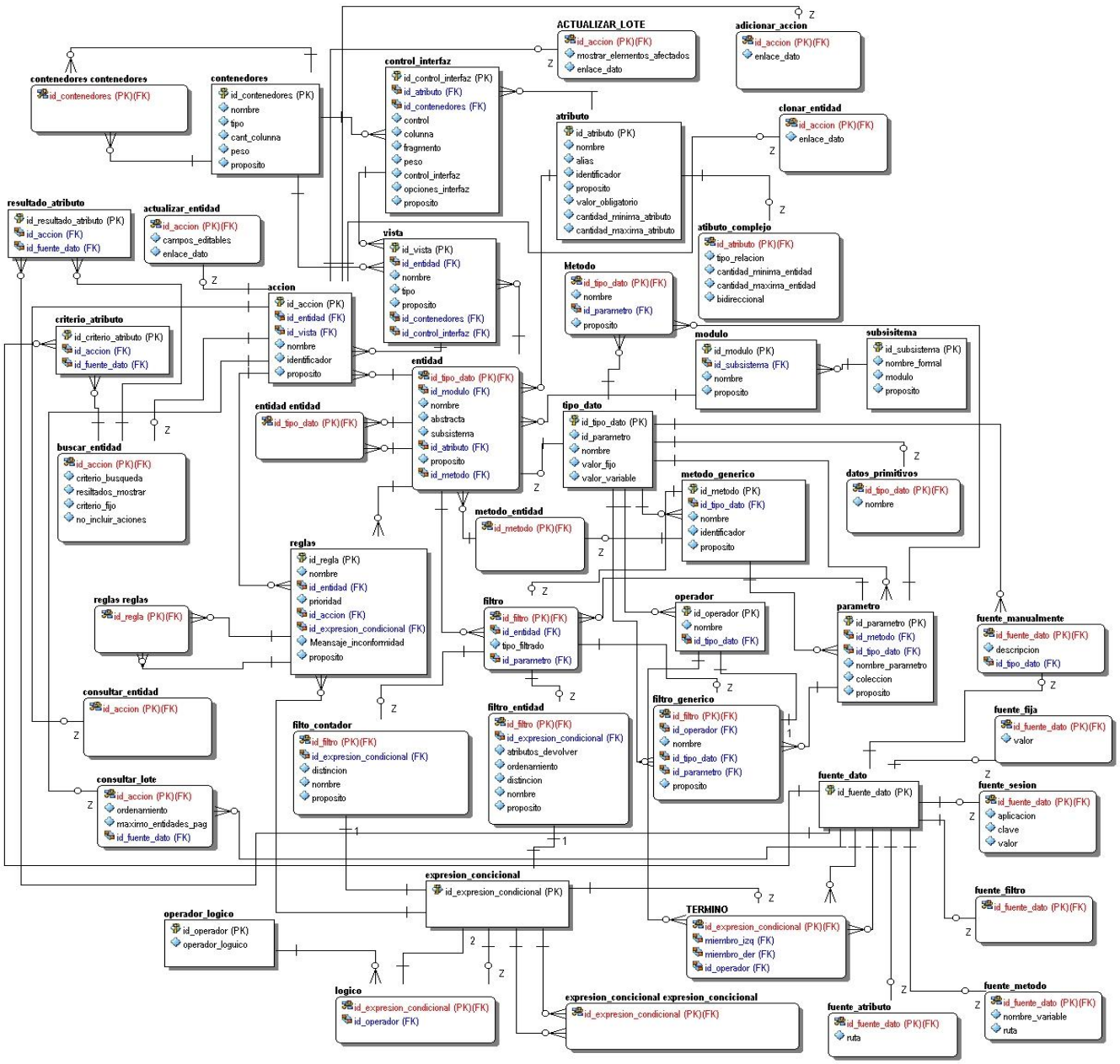


Figura 38: Modelo de datos.

3.7. Validación

La medición según Fenton:

La medición es el proceso de asignación de números o símbolos a los atributos de las entidades, de forma tal que las definan de acuerdo con las reglas claramente concretadas.

Es ineludible intentar «medir lo no medible» para mejorar la comprensión de entidades particulares.

Para validar la propuesta de diseño se utilizarán las métricas TC: haciendo uso del número total de operaciones para calcular los atributos responsabilidad, complejidad de implementación y reutilización. Además se aplicará APH: para verificar si el diseño es sencillo o complejo. Se escogieron estas porque se pueden aplicar fácilmente y con ellas se mide la calidad de la propuesta de diseño.

Tamaño Operacional de Clase (TOC): Hace referencia al número de métodos que contiene una clase. Se determina por los atributos: Responsabilidad, Complejidad y Reutilización, donde existe una relación directa con los dos primeros e inversa con el último respectivamente.

En la siguiente sección de tablas se muestran:
Resultados de la evaluación de la métrica Tamaño Operacional de Clase (TOC).

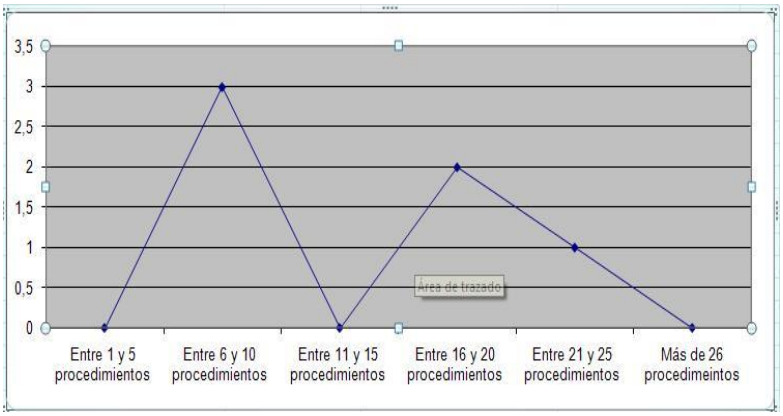


Figura 39: Gráfica de resultados obtenidos agrupados en los intervalos definidos.

Los resultados, en porciento, obtenidos agrupados en intervalos definidos.

Criterio	Cantidad de clases	Promedio
Entre 1 y 5 procedimientos	0	0
Entre 6 y 10 procedimientos	3	50
Entre 11 y 15 procedimientos	0	0
Entre 16 y 20 procedimientos	2	33,33333333
Entre 21 y 25 procedimientos	1	16,66666667
Más de 26 procedimeintos	0	0
Total	6	100

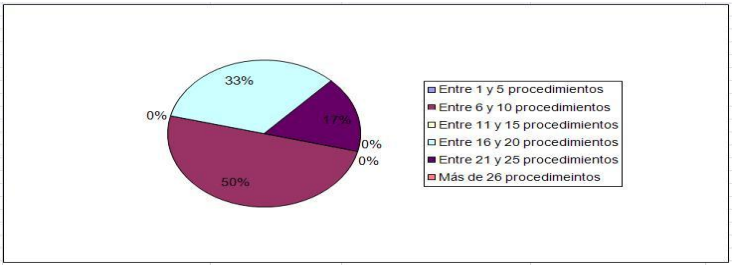


Figura 40: Gráfica en porciento de resultados obtenidos agrupados en los intervalos definidos.

Aquí se presentan los resultados de la medición de la métrica TOC para el atributo.

Responsabilidad	Cantidad de clases	Promedio
Baja	3	50
Media	3	50
Alta	0	0

Figura 41: Gráfica de incidencia de resultados de la evaluación de la métrica TOC en el atributo Responsabilidad.

Complejidad	Cantidad de clases	Promedio
Baja	3	50
Media	3	50
Alta	0	0

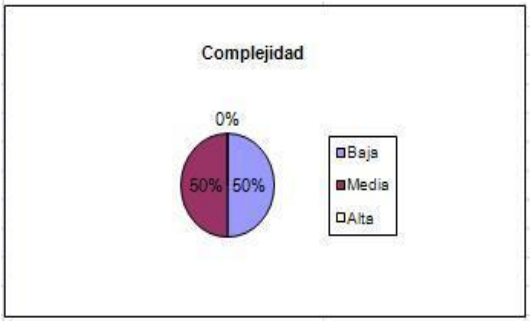


Figura 42: Gráfica de incidencia de resultados de la evaluación de la métrica TOC en el atributo Complejidad.

Reutilización	Cantidad de clases	Promedio
Alta	3	50
Media	3	50
Baja	0	0

Figura 43: Gráfica de incidencia de resultados de la evaluación de la métrica TOC en el atributo Reutilización.

Los atributos Responsabilidad y Complejidad se comportan de manera satisfactoria para un 100% de las clases, por otra parte la Reutilización se encuentra dentro de los límites aceptables.

Árbol de Profundidad de la Herencia: Esta métrica está dada por el nivel de profundidad de la herencia entre la clase hija y la base.

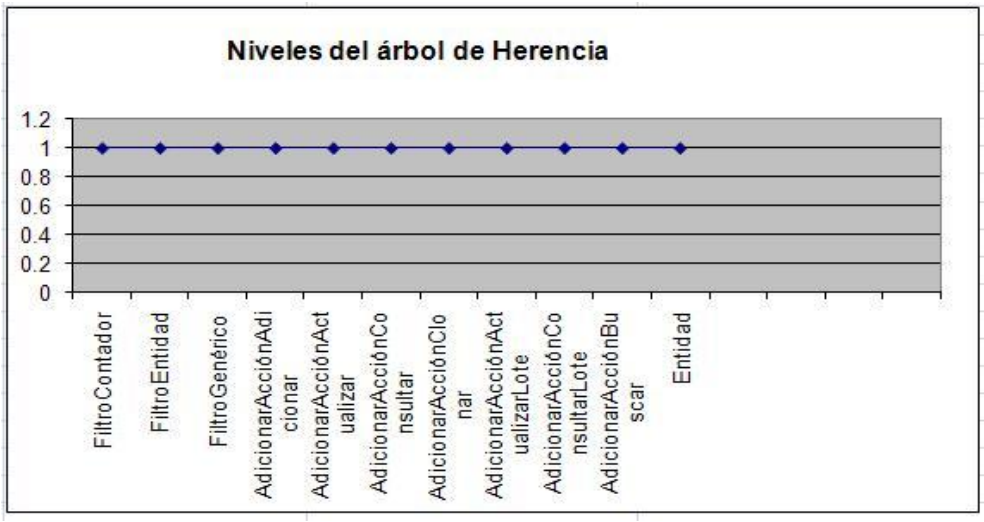


Figura 44: Gráfica de niveles del árbol de herencia.

Figura 45: Gráfica de profundidad de la herencia.

Al aplicar la métrica Árbol de Profundidad de Herencia se muestra el comportamiento de las clases. Tienen un valor simple (pequeño, en este caso 1); por lo que el diseño es sencillo aunque esto implique que no hay muchos métodos que se puedan reutilizar.

3.8. Conclusiones Parciales

En el capítulo se generaron los artefactos pertenecientes al flujo de trabajo de Diseño (diagramas de clases de diseño, de secuencia, de paquete y modelo de datos). Para la realización del diseño fue necesaria la utilización de patrones y de la arquitectura propuesta para lograr buenas prácticas a la hora del desempeño del mismo. Por lo que se concluye esta fase de manera satisfactoria.

Se han aplicado las métricas para la evaluación del diseño obtenido, se puede observar que no presenta una complejidad alta y permite la integración de módulos con facilidad; también tiene un bajo acoplamiento pues la profundidad de la herencia está acorde con los umbrales definidos. Facilita el posterior desarrollo del sistema en etapas superiores al diseño; de esta manera se procura el cumplimiento del objetivo planteado.

Conclusiones

Con la culminación del trabajo, todas las tareas fueron desarrolladas a fin de cumplir con los resultados esperados. Esto fue posible por el conocimiento adquirido durante la investigación del tema.

Específicamente para ello fue necesario:

El estudio y descripción de las herramientas generadoras de código automático de forma que se identificaran las ventajas y desventajas.

Se definieron las herramientas, metodologías, y tecnologías a usar para el desarrollo de la solución.

La realización del análisis y diseño de un sistema automático para la gestión de entidades del negocio.

El estudio y aplicación de métricas usadas para validar el diseño.

Los resultados que este documento describe son los artefactos generados en el flujo de análisis y diseño. Además de las validaciones donde se aplicaron métricas de medición del diseño de software por lo que se puede decir que el trabajo concluido muestra un correcto valor técnico.

Recomendaciones

Realizar la implementación del diseño propuesto para incrementar la productividad de las empresas desarrolladoras de software.

Realizar una segunda iteración del análisis y diseño, incluyendo el resto de los casos de uso que no fueron abordados en el presente trabajo porque no se encontraban dentro del alcance especificado para el trabajo.

.

Bibliografía

Bibliografía Consultada

2002. Architecture patterns. *Commonly used architectural patterns in Java applications.* [En línea] 21 de 10 de 2002. http://articles.techrepublic.com.com/5100-10878_11-1049864.html.

2006. .NET Application Architecture: the Data Access Layer. [En línea] 11 de 06 de 2006. [Citado el: 25 de 02 de 2010.] http://www.simple-talk.com/dotnet/.net-framework/.net-application-architecture-the-data-access-layer/.

2001-2009. Ajax Applications, Web Reporting, Web Development Tool & Code Generator for PHP, C# y VB.NET. *CodeCharge Studio 4.3.* [En línea] 2001-2009. http://www.yessoftware.com/products/product_detail.php?product_id=1.

2010. AndroMDA. *Components quickly with AndriMDA.* [En línea] 18 de 04 de 2010. [Citado el: 02 de 05 de 2010.] http://www.andromda.org/docs/.

Aneiros, Andrés Vegas. 2000. Enterprise Java Beans. [En línea] mayo de 2000. [Citado el: 20 de 02 de 2010.] http://jungla.dit.upm.es/~santiago/externos/docencia/doctorado/drci/trabajos99-00/avegas/sld002.htm.

Capítulo 2: Ingeniería de software, Análisis y Diseño. [En línea] [Citado el: 17 de 02 de 2010.] http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/fuentes_k_jf/capitulo2.pdf.

Douglas C. Schmidt. 2006. Model-Driven Engineering. [En línea] 02 de 2006. [Citado el: 11 de 02 de 2010.] http://www.cs.wustl.edu/~schmidt/PDF/GEI.pdf.

Generación de código en función del modelo. [En línea] [Citado el: 11 de 02 de 2010.] http://librosweb.es/symfony_1_0/capitulo14/generacion_de_codigo_en_funcion_del_modelo.html.

2007. Guía de Usuario de Enterprise Architect 7.0. *Modelado de Datos.* [En línea] 2007. [Citado el: 21 de 03 de 2010.] http://www.sparxsystems.com.ar/download/ayuda/index.html.

Herrington, Jack. Code Generation in Action. *Kickstartnews.com.* [En línea] Manning Publications. [Citado el: 9 de 02 de 2010.] http://www.kickstartnews.com/reviews/books/code_generation_action.html. ISBN 1-930110-97-9.

Hibernate. *Relational Persistence for Java and .NET.* [En línea] [Citado el: 24 de 02 de 2010.] http://www.hibernate.org/.

Jaramillo, Luz Mato. *Capítulo IV LuzMato.*

Java en Castellano. [En línea] Copyright © 1999-2010 Programación en castellano. [Citado el: 07 de 02 de 2010.] http://www.programacion.com/java/tutorial/swing/1/.

2010. Layered architecture. *A practical introduction to layered architecture.* [En línea] 2010. [Citado el: 01 de 03 de 2010.] http://fewagainstmany.com/blog/introduction-to-layered-architecture-part-one.

Lic. Espinosa Robles. 2009. Slideshare. *Clase Flujo de Análisis.* [En línea] 2009. [Citado el: 20 de 02 de 2010.] http://74.125.113.132/search?q=cache:O90u9TFeU0AJ:www.slideshare.net/juliopari/13-clase-flujo-de-analisis+artefactos+del+flujo+analisis&cd=1&hl=es&ct=clnk&gl=cu.

205. metodologias-de-desarrollo-del-software.html. *Metodologías de desarrollo del software .* [En línea] 205. [Citado el: 04 de 02 de 2010.]

Mi biblioteca. *Ingeniería de software.* [En línea] Editorial UOC . http://books.google.com.cu/books?id=_tKTpr4Ah88C&pg=PA144&lpg=PA144&dq=paquetes+de+analisis+del+software&source=bl&ots=RtITp2BHxO&sig=Uj9mEQ6pSNeZG8IO-biuhgO-PQI&hl=es&ei=0qmOS5znBMn-8QaQzaz6DQ&sa=X&oi=book_result&ct=result&resnum=2&ved=0CAsQ6AEwAQ#v=onep.

2002. Model-View-Controller. [En línea] 2002. http://java.sun.com/blueprints/patterns/MVC.html.

OpenXava. *Aprende OpenXava con ejemplos.* [En línea] [Citado el: 13 de 02 de 2010.] http://www.openxava.org/web/guest/doc.

2005. Osmosis Latina. *Definición y usos.* [En línea] 07 de 09 de 2005. [Citado el: 04 de 03 de 2010.] http://www.osmosislatina.com/lenguajes/uml/casos.htm.

Raúl González Duque. Mundo Geek. *Modelo de Datos .* [En línea] http://mundogeek.net/archivos/2004/08/26/modelo-de-datos/.

Reglas para la generación Automática de código. [En línea] Universidad Nacional de Colombia. [Citado el: 11 de 02 de 2010.] http://dyna.unalmed.edu.co/ver_resumen.php?id_articulo=AM150207.

Ricardo Rocha Amorim, Manuel Lama Penín, Eduardo M. Sánchez Vila. 01/02/2008. *proyecto suma.* 01/02/2008.

Sarduy Sánchez, Ismel y Cid Escalona, Lilian. mayo, 2009. *Análisis y Diseño del subsistema Multimoneda del ERP-Cuba.* mayo, 2009.

Scribd. *RUP etapa de diseño.* [En línea] [Citado el: 20 de 03 de 2010.] http://www.scribd.com/doc/395783/RUP-etapa-diseno.

2000-2007. Sparx. *Modelo Físico.* [En línea] 2000-2007. [Citado el: 22 de 03 de 2010.] http://www.sparxsystems.com.ar/resources/tutorial/physical_models.html.

2010. YesSoftware lanza la nueva versión de CodeCharge Studio. [En línea] PR Newswire Europe Limite, 2010. [Citado el: 11 de 02 de 2010.] http://www.prnewswire.co.uk/cgi/news/release?id=86097.

Zapata, María Antonia. 2006. *UML Introducción.* 2006.

Bibliografía Referenciada

2007 Solus S.A.. *Diagramas de componentes.* [En línea] 2007 Solus S.A. [Citado el: 21 de 03 de 2010.] http://www.sparxsystems.com.ar/download/ayuda/index.html?componentdiagram.htm.

Pérez Betancourt, Yadian Guillermo y González Polanco, Liset. 2009. *Análisis y Diseño del subsistema Préstamos del proyecto Modernización del Sistema Bancario Cubano.* La Habana, CUba : s.n., 2009.

2002-2007. ALS. *OptimalJ de Compuware.* [En línea] 2002-2007. [Citado el: 19 de 02 de 2010.] http://www.als-es.com/home.php?location=herramientas/entorno-desarrollo/optimalj.

2010. AndroMDA. *Welcome to AndroMDA!* [En línea] 17 de 02 de 2010. [Citado el: 26 de 02 de 2010.] http://www.andromda.org/index.php.

Aplicaciones en capas. *Capítulo 3. Justificación .* [En línea] [Citado el: 21 de 02 de 2010.] http://oness.sourceforge.net/proyecto/html/ch03s02.html.

Clikear.com. *Diagramas de Interacción.* [En línea] [Citado el: 05 de 03 de 2010.] http://www.clikear.com/manuales/uml/diagramasinteraccion.aspx.

CodeCharge Studio. *Construya webs dinámicas sin apenas programar.* [En línea] [Citado el: 20 de 02 de 2010.] http://www.cmp.es/index.php?option=com_content&view=article&id=18&Itemid=40.

CodeSmith Tools. [En línea] [Citado el: 05 de 02 de 2010.] http://www.codesmithtools.com/.

2009. Diagramas de Caso de Uso. [En línea] 2009. [Citado el: 27 de 02 de 2010.] http://www.slideshare.net/javi2401/diagramas-de-casos-de-uso-presentation.

El Proceso Unificado de Desarrollo de Software. **Rumbaugh, Ivar Jacobson Grady Booch James. 2000.** Madrid : Pearson Educación, S.A., 2000.

2010. Erwin. *CA ERwin Data Modeler.* [En línea] 2010. [Citado el: 01 de 03 de 2010.] http://erwin.com/products/detail/ca_erwin_data_modeler/.

Garcia, Walter y Ospina, Gustavo. Patron de Diseño: Command. *Command.* [En línea] [Citado el: 02 de 04 de 2010.] http://agamenon.uniandes.edu.co/~pfiguero/soo/PatronesDiseno/Command/Command.htm.

Genexus. [En línea] [Citado el: 21 de 02 de 2010.] http://www.genexus.com/portal/hgxpp001.aspx?2,61,1006,O,S,0,MNU;E;248;1;MNU;,.

Herramienta CASE. [En línea] [Citado el: 25 de 02 de 2010.] http://www.cyta.com.ar/biblioteca/bddoc/bdlibros/proyectoinformatico/libro/c5/c5.htm.

2004. IBM. *An introduction to Model Driven Architecture.* [En línea] 17 de 02 de 2004. [Citado el: 20 de 02 de 2010.] http://www.ibm.com/developerworks/rational/library/3100.html.

Introducción a Spring Web Flow. *Qué es spring web flow.* [En línea] [Citado el: 15 de 02 de 2010.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=springWebFlow.

Modelo del Dominio. [En línea] [Citado el: 01 de 03 de 2010.] http://lsi.ugr.es/~ig1/isoo/larman/Modelo%20del%20dominio.pdf.

OpenXava. [En línea] http://www.openxava.org/web/guest/home.

Pabo Figueroa. Conceptos básicos en un Diagrama de Colaboración. [En línea] [Citado el: 05 de 03 de 2010.] http://webdocs.cs.ualberta.ca/~pfiguero/soo/uml/colaboracion01.html.

Pressman, Roger S. *Ingeniería del Software Enfoque Práctico.*

Rubiano, Ing. Sandra Merchán. Ocubre 2005. *El Lenguaje Unificado de Modelado.* s.l. : Fundación Universitaria Manuela Beltrán, Ocubre 2005.

Seam city. *Comparativa de Frameworks: Spring.* [En línea] [Citado el: 15 de 02 de 2010.] http://seamcity.madeinxpain.com/archives/comparativa-spring.

2000-2007. Spark Systems . *Diagrama de Paquete UML 2.* [En línea] 2000-2007. [Citado el: 21 de 03 de 2010.] http://www.sparxsystems.com.ar/resources/tutorial/uml2_packagediagram.html.

2010. Spring Source. *Spring Roo.* [En línea] 2010. [Citado el: 19 de 02 de 2010.] http://www.springsource.org/roo.

2009. The Enterprise Architect Building An Agile Enterprise. *MDE - Model Driven Engineering - reference guide.* [En línea] 15 de 01 de 2009. [Citado el: 11 de 02 de 2010.] http://www.theenterprisearchitect.eu/archive/2009/01/15/mde---model-driven-engineering----reference-guide.

Visual Paradigm for UML - UML tool for software application development. [En línea] [Citado el: 16 de 02 de 2010.] http://www.visual-paradigm.com/product/vpuml/.

2010. "The Architecture of Choice for a Changing World" . *OMG Model Driven Architecture .* [En línea] 06 de 02 de 2010. [Citado el: 28 de 02 de 2010.] http://www.omg.org/mda/.

Anexos

Especificación de Caso de Uso

Adicionar Subsistema

Gestión de Entidades

SADSG

SADSG

Módulo Gestionar Entidad

Modelos de Casos de Uso del sistema

Precondiciones

El usuario debe estar autenticado en el sistema.

Pos condiciones

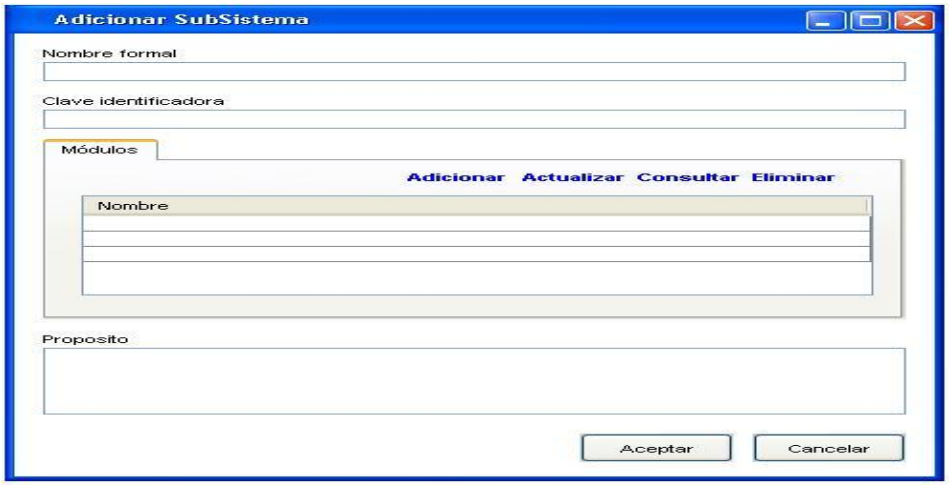
Queda registrado el subsistema en el sistema.

Reglas del negocio

El subsistema debe contener al menos un módulo.

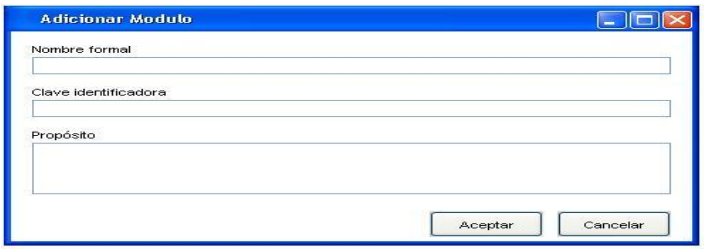
Descripción extendida

Caso de Uso:	**Adicionar Subsistema**
Actores:	**Usuario**
Flujo Normal de Eventos	
Sección "Adicionar Subsistema"	
Acción del Actor	**Respuesta del Sistema**
1. El usuario selecciona la opción "Adicionar".	2. El sistema muestra una interfaz con los campos necesarios para registrar el subsistema.

<table>
<tr><td colspan="2">Prototipo de interfaz
 
</td></tr>
<tr><td>3. El usuario introduce los datos y selecciona la opción “Aceptar”.

En el caso de seleccionar la opción “Adicionar”. Ver sección “Adicionar Módulo”.
En el caso de seleccionar la opción “Actualizar”. Ver sección “Actualizar Módulo”.
En el caso de seleccionar la opción “Consultar”. Ver sección “Consultar Módulo”.
En el caso de seleccionar la opción “Eliminar”. Ver sección “Eliminar Módulo”.</td><td>4. El sistema valida los datos de entrada.

En caso de datos incorrectos. Ver sección “Datos Incorrectos”.</td></tr>
<tr><td></td><td>5. El sistema registra el subsistema y finaliza el caso de uso.</td></tr>
<tr><td colspan="2">Flujos Alternos</td></tr>
<tr><td colspan="2">Sección “Datos Incorrectos”</td></tr>
<tr><td>Acción del Actor</td><td>Respuesta del Sistema</td></tr>
<tr><td></td><td>1. El sistema señala los datos incorrectos y muestra el mensaje “Entrada de datos no válidos”.</td></tr>
</table>

2. El usuario corrige los datos de entrada.	3. El sistema pasa a la acción 4 del flujo normal de eventos.
Sección “Adicionar Módulo”	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra una interfaz con los campos necesarios para registrar el módulo.
Prototipo de interfaz	
	
2. El usuario introduce los datos y selecciona la opción “Aceptar”.	3. El sistema valida los datos de entrada y pasa a la acción 4 del flujo normal de eventos. a. En caso de datos incorrectos. Ver sección “Datos Incorrectos”.
Sección “Actualizar Módulo”	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra una interfaz con los campos necesarios para actualizar el módulo.
2. El usuario introduce los datos y selecciona la opción “Aceptar”.	3. El sistema valida los datos de entrada y pasa a la acción 4 del flujo normal de eventos. a. En caso de datos incorrectos. Ver sección “Datos Incorrectos”.
Sección “Consultar Módulo”	

Acción del Actor	Respuesta del Sistema
	1. El sistema muestra una interfaz con los campos del módulo.
2. El usuario selecciona la opción “Aceptar”.	3. El sistema pasa a la acción 4 del flujo normal de eventos.
Sección “Eliminar Módulo”	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema chequea el módulo. a. En caso de entidades en el módulo. Ver sección “Entidades en módulo”. b. El sistema elimina el módulo y pasa a la acción 4 del flujo normal de eventos.
Sección “Entidades en módulo”	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra un mensaje de error “El módulo contiene entidades”. 2. El sistema pasa a la acción 4 del flujo normal de eventos.

Validación de campos.

Nombre	Ob.	Longitud	Validación	Acción
Nombre Formal	X		No tiene	
Clave Identificadora	X		Solo puede contener letras en minúsculas.	Muestra el mensaje “Solo letras en minúsculas son permitidas”.

Módulos	X		Al menos debe haber un módulo en el subsistema.	
Propósito			No tiene	

Anexo 1. Tabla: Descripción del Módulo Gestionar Subsistema / Caso de Uso: Adicionar Subsistema.

Especificación de Caso de Uso

Buscar Subsistema

Gestión de Entidades

SADSG

SADSG

Módulo Gestionar Entidad

Modelos de Casos de Uso del sistema

Especificación del caso de uso.

Precondiciones.

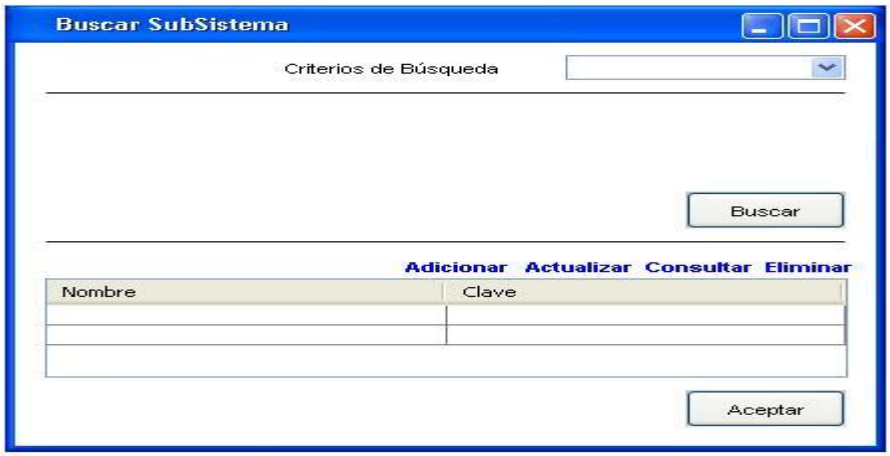
El usuario debe estar autenticado en el sistema.

Poscondiciones.

Reglas del negocio.

Descripción extendida.

Caso de Uso:	**Buscar Subsistema**
Actores:	**Usuario**
Flujo Normal de Eventos	
Sección "Buscar Subsistema"	
Acción del Actor	**Respuesta del Sistema**
1. El usuario introduce los criterios de búsqueda y	2. El sistema realiza la búsqueda y muestra el

selecciona la opción “Buscar”.	resultado.
Prototipo de interfaz	
	
3. El usuario selecciona una opción. a. En el caso de seleccionar la opción “Adicionar”. Ver “DCU Adicionar Subsistema”. b. En el caso de seleccionar la opción “Actualizar”. Ver “DCU Actualizar Subsistema”. c. En el caso de seleccionar la opción “Consultar”. Ver “DCU Consultar Subsistema”. d. En el caso de seleccionar la opción “Eliminar”. Ver “DCU Eliminar Subsistema”.	4. Finaliza el caso de uso.
Flujos Alternos	

Anexo 2. Tabla: Descripción del Módulo Gestionar Subsistema / Caso de Uso: Buscar Subsistema.

Artefactos generados para Módulo: Subsistema en el capítulo 2 y capítulo 3

Glosario de términos

Modelo: Representación simplificada de la realidad que ayuda a entender un sistema grande y complejo que no puede ser comprendido fácilmente en su totalidad.

XMI: XMI es un estándar de OMG para el intercambio de información y metainformación entre herramientas, repositorios y aplicaciones, que proporciona un formato de intercambio para entornos distribuidos. Estándar para el intercambio de metamodelos usando XML.

MOF: La Estructura de operaciones de Microsoft (MOF, Microsoft Operations Framework) es una colección de recomendaciones, principios y modelos. Proporciona una guía técnica completa para lograr confiabilidad, disponibilidad y capacidad de soporte técnico y de administración del sistema de producción crítico con productos y tecnologías de Microsoft.

WSDL: Web Services Description Language, formato XML para describir interfaz publica a los servicios web.

J2EE: Java Platform, Enterprise Edition o Java EE es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en lenguaje JAVA con arquitectura de N niveles distribuidos, basándose en componentes de software modulares se ejecuta sobre servidores aplicaciones.

Casos de Uso: Los casos de uso (CU) no son parte del diseño (cómo), sino parte del análisis (qué). Por lo tanto ayudan a describir lo que el sistema debe hacer.

IU: Interfaz de Usuario.

EJB: Define un modelo para el desarrollo y distribución de componentes reutilizables del lado servidor (Java). Define un conjunto de interfaces estándar que dan respuesta a las necesidades típicas de los componentes del servidor.

JSF: (Java Server Faces) framework de desarrollo basado en el patrón MVC (Modelo Vista Controlador), JSF usa JavaServer Pages (JSP) como la tecnología que permite hacer el despliegue de las páginas.

Struts: herramienta de soporte para desarrollar aplicaciones web, basadas en el MVC.

MVC: describe la forma de organizar el código de una aplicación web, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Tapestry: marco de trabajo de código abierto para la creación de páginas web basadas en componentes.

Mapear: La finalidad de mapear es simplemente comprender de mejor manera las interacciones necesarias en un proceso organizadas de tal manera que sea posible apreciar desviaciones no deseadas o analizadas los puntos críticos para evitar los posibles fallos.