Universidad Nacional de Loja

Área de la Energía, las Industrias y los Recursos Naturales no Renovables

Carrera de Ingeniería en Sistemas

Desarrollo de una herramienta de análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la generación de código fuente bajo la especificación del lenguaje de programación C Sharp (ECMA – 334)

Tesis previa a la obtención del
Título de Ingeniero en Sistemas

Autores:

Alexandra Elizabeth Maurad Córdova
Danny Emanuel Muñoz Flores

Director:

Ing. Edwin René Guamán Quinche

Loja – Ecuador

2010

CERTIFICACIÓN

Ing. Edwin René Guamán Quinche
DOCENTE DE LA CARRERA DE INGENIERÍA EN SISTEMAS DE LA UNIVERSIDAD NACIONAL DE LOJA, DIRECTOR DE TESIS

C E R T I F I C A :

Que los señores egresados Alexandra Elizabeth Maurad Córdova y Danny Emanuel Muñoz Flores, realizaron el trabajo de investigación titulado **"Desarrollo de una Herramienta de análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp (ECMA – 334)"** bajo mi dirección y asesoramiento, mismo que fue revisado, enmendado y corregido minuciosamente. En virtud que la Tesis reúne, a satisfacción, las cualidades de fondo y forma exigidas para un trabajo de este nivel, autorizo su presentación, sustentación y defensa ante el tribunal respectivo.

Loja, julio de 2010

…….
Ing. Edwin René Guamán Quinche
DIRECTOR DE TESIS

AUTORÍA

Los conceptos, ideas, análisis, descripciones, opiniones, conclusiones y recomendaciones vertidas en el desarrollo de la presente Tesis, son de absoluta responsabilidad de sus autores, excepto aquellas que se encuentran debidamente citadas.

…….	…….
Alexandra Elizabeth Maurad Córdova	**Danny Emanuel Muñoz Flores**

PENSAMIENTOS

"La verdad a medias descubierta por nosotros mismos, vale más que toda una verdad aprendida de otros".

S.R.

"Con la paciencia, con la bondad, la tranquilidad y la perseverancia se obtiene todo… y algo más".

Benjamín Franklín

"Nada que pueda conseguirse sin pena y sin trabajo, es verdaderamente valioso".

J. Addison

"El que trabaja con las manos es un artesano. El que emplea en su obra manos y cerebro, un artífice. Quien obra con las manos, cerebro y corazón, un artista".

L. Nizer

"De no ser por las ilusiones aún estaríamos en la edad media. Sin ellas, la mayoría de los logros de la humanidad nunca se hubiesen intentado".

A.F.O.

"El software libre es técnicamente viable, económicamente sostenible y socialmente justo. Existe un mundo lleno de posibilidades. Aprovecharlo depende de nosotros".

Anónimo

AGRADECIMIENTO

Al finalizar el presente trabajo investigativo queremos dejar constancia de nuestro más sincero agradecimiento a todos quienes participaron en el desarrollo del mismo:

Primeramente a Dios, por habernos permitido cumplir el ansiado objetivo de obtener nuestro título universitario y así finalizar una etapa más de nuestras vidas.

A la Universidad Nacional de Loja, al Área de la Energía, las Industrias y los Recursos Naturales no Renovables y a la Carrera de Ingeniería en Sistemas, en la persona de sus autoridades y demás administrativos, por proporcionarnos incesantemente los medios necesarios para nuestra educación no sólo académica sino también humana.

A nuestro Director de Tesis, Ing. René Guamán, quien durante todo el proceso investigativo y de desarrollo nos supo dirigir acertadamente brindándonos su tiempo, sus conocimientos y su valiosa experiencia.

A la planta docente de la CIS quienes a lo largo de nuestra formación universitaria nos prodigaron no solamente sus sabios conocimientos sino también su invaluable amistad, grabando en nosotros un imborrable recuerdo de aprecio, cariño y lealtad.

A estudiantes y egresados de la Carrera de Ingeniería en Sistemas y demás profesionales de la rama, quienes colaboraron generosa y desinteresadamente con nuestra investigación, puesto que sin ellos hubiese sido imposible culminar y entregar este pequeño aporte a la Universidad Nacional de Loja y a la sociedad en general.

A todos nuestros amigos y amigas con quienes compartimos momentos de felicidad y tristeza, pero más aún de solidaridad y superación. Por todo el apoyo y amistad que nos ofrecieron mientras compartimos arduas jornadas de estudio, trabajo y sacrificio.

A todos nuestros familiares por su apoyo total, por su confianza inquebrantable y por todo el amor incondicional que nos prodigan a cada instante. Gracias a ellos y a su constante sacrificio, encontramos las fuerzas necesarias para alcanzar una de nuestras más ambiciosas aspiraciones.

A todos ustedes infinitamente **¡GRACIAS!** ...

Los Autores

DEDICATORIA

Con toda mi gratitud y cariño quiero dedicar este trabajo a mi familia, de manera especial a mis padres Carmen y Jorge, a mis hermanos Gabriel e Isaac, y a mis abuelitos Victoria, María y José.

Alexandra

A todos mis familiares, amigos, maestros y demás, que formaron un sendero iluminado en el que he podido dar pasos firmes y llegar a cumplir una meta más en mi vida…

Muñoz, Danny

1. RESUMEN

El producto final obtenido como resultado de la presente investigación, denominado **AleDan** UML, se constituye en una herramienta para el análisis y diseño de sistemas, que fundamentalmente permite la creación de diagramas de clases, paquetes, componentes, despliegue, casos de uso y de secuencia bajo las especificaciones dadas por UML 2.0, y la generación de código fuente para el lenguaje de programación C# a partir de los diagramas de clases y secuencia.

La construcción de ésta herramienta ha implicado un arduo trabajo investigativo y de desarrollo, mismo que ha sido plasmado en éste documento de la manera expuesta a continuación:

En el acápite **Metodología** se han descrito los métodos, materiales y técnicas de trabajo empleados a lo largo del presente trabajo investigativo. En la sección correspondiente a la Metodología para el desarrollo de la Investigación, se hace una breve descripción de las actividades que se pudieron ejecutar al aplicar los métodos Inductivo y Deductivo. Por su parte la Metodología para el Desarrollo de la Herramienta hace hincapié en cada una de las fases que conforman a la Metodología Ágil para el Desarrollo de Software RUP.

En el punto referente a **Fundamentación Teórica**, se expone la historia, definiciones y las especificaciones y representaciones gráficas más importantes que UML 2.0 ofrece para la elaboración de diagramas. Así mismo se presentan las definiciones, características y especificaciones de mayor relevancia para el lenguaje de programación C#. Posteriormente se dan a conocer las definiciones, objetivos, características y componentes de las herramientas CASE, para continuar con un recuento de aquellas que en la actualidad son más utilizadas por los analistas de nuestro medio, y así, finalizar con una comparativa de las mismas.

Por su parte el apartado **Desarrollo de la propuesta Alternativa**, explica los pasos llevados a cabo para la construcción de la herramienta **AleDan UML**. En primera instancia se describe el problema identificado para poder delimitar el alcance de la aplicación. A continuación se muestra el prototipo de pantallas, la descripción de casos de uso y los diagramas en los que nos afianzamos para documentar el software.

Luego se da paso a la definición de la plataforma de desarrollo y de las políticas de programación. También se describen las actividades desarrolladas en las etapas de Verificación y de Validación del sistema para posteriormente exponer los resultados alcanzados.

Finalmente se hace el planteamiento de algunas **Conclusiones** elaboradas como producto de todo el proceso investigativo, y algunas **Recomendaciones** para los usuarios de la herramienta **AleDan UML** y para los desarrolladores de software en general.

2. SUMMARY

The final result obtained of the present research, which is denominated "**AleDan UML**", it is an important tool to the analysis and design of informatics systems, fundamental it permit the construction of classes, packages, components, deployment, use cases and sequence diagrams, based in the specification giving by UML 2.0, and source code generation for C# programming language, which result of the class and sequence diagrams.

The construction of this tool has implicated a hard investigative and development work. So this work has been focused in this document in the way that after we are going to detail.

About **Methodology,** has been described the methods, materials and work techniques, employed during the research work. In the methodology to the development of the research section, we are a brief description of the activities which we do applying the Deductive and Inductive methods. In other hand the methodology to the development of the tool take into account in each step which form the Agile Methodology for the Software Development RUP

In the point which refers to the **Theoretical reference**, we expose the history, definition and specifications and graphic representations more important than UML 2.0 offer to the elaboration of diagrams. In the same way we present the definitions, characteristics and specifications most relevant to the C# programming language. After, we present definitions, objectives, characteristics and components of the CASE tools, to follow with a remembering of the tools which actually are more used by the analyzers of our local city. So and end with a comparison of them.

In the same way the part of **Development of the alternative proposal,** it explain the steps with are necessary to the construction of AleDan UML tool. First we describe the identified problem to delimitate the reach application. After that we are going to show the screens prototype, the use cases description and the diagrams in which we support to document the software.

Then, we define the platform of development and the programming policies. Also we describe the developed activities in the verification and validation phases of the system for then expose the achieved results.

Finally we establish some **Conclusions** elaborated as a product of all the research process, and some **Recommendations** focused on to the users of the AleDan UML tool, and to the software developers in general.

3. ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

4. INTRODUCCIÓN

La Universidad Nacional de Loja en su afán de formar profesionales capaces de dar solución a las problemáticas presentadas en nuestra sociedad, impulsa en cada una de las carreras que oferta el desarrollo de investigaciones de carácter científico – técnico que coadyuven al desarrollo sustentable de nuestro entorno y contribuyan a su adelanto. Es así que en la carrera de Ingeniería en Sistemas, se capacita a los estudiantes de tal forma que a través de conceptos propios de sistemas, proporcionen alternativas válidas para dar solución a diversas dificultades que pueden llegar a presentarse en empresas tanto públicas como privadas, ya sea mediante la implantación de sistemas informáticos o mediante el planteamiento de propuestas para el mejoramiento de la situación problemáticas existentes.

Los constantes avances tecnológicos que se han dado en las últimas décadas han contribuido a que las empresas incrementen su productividad, mediante la adquisición de nuevos equipos (hardware) o bien, mediante la adquisición e implantación de tecnologías de información (software). Por su parte las empresas desarrolladoras de software incrementan su competitividad mediante la utilización de herramientas que permiten optimizar recursos y obtener productos de calidad. Las empresas más sobresalientes de desarrollo de software son las que utilizan buenas herramientas (herramientas CASE) que apoyen a una etapa específica o a todo el ciclo de desarrollo de un producto.

Ante la necesidad de mejorar el rendimiento en todos los sectores productivos, los seres humanos dependemos cada vez más del software en nuestras actividades cotidianas, razón por la que su fiabilidad es vital. Sin embargo, y pese a los recursos y esfuerzos invertidos, gran parte de los proyectos de desarrollo de software no llegan a cumplir los objetivos planteados, debido a errores de diseño detectados en etapas posteriores a éste. Una de las posibles alternativas de solución a éste inconveniente, es el correcto diseño de la arquitectura de la aplicación a desarrollar, para lo que se requiere el apoyo de una herramienta eficiente capaz de satisfacer, en éste caso específico, las necesidades de los desarrolladores que deban o quieran realizar su trabajo en el lenguaje de programación C Sharp.

La puesta en marcha de la investigación denominada **“Desarrollo de una Herramienta de análisis y diseño de sistemas que permita la documentación de**

software basada en UML 2.0 y la generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp (ECMA – 334[1])" dio paso a la creación de la herramienta **AleDan UML**, misma que estuvo encaminada a cumplir con los siguientes objetivos:

4.1. OBJETIVO GENERAL

- Desarrollar una Herramienta de análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp (ECMA – 334).

4.2. OBJETIVOS ESPECÍFICOS

- Analizar las herramientas de análisis y diseño de sistemas basadas en la especificación UML 2.0 más utilizadas por los desarrolladores de software.

- Desarrollar un módulo de diagramación basado en el Lenguaje de Programación C Sharp, y en la especificación UML 2.0 para modelar diagramas de clases, paquetes, componentes, despliegue, casos de uso y secuencia.

- Desarrollar un módulo de generación de código fuente para el Lenguaje de Programación C Sharp, a partir de los diagramas de clases y secuencia.

AleDan UML proporciona a los estudiantes que forman parte de la carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja y a los desarrolladores en general, los instrumentos necesarios para diseñar la arquitectura de una aplicación software y para generar automáticamente el código fuente del diseño realizado previamente en el lenguaje C Sharp.

Con el uso de esta herramienta, se espera minimizar el riesgo de que los desarrolladores adopten el mal hábito de la "ingeniería inversa", es decir, primero codificar y luego realizar el análisis y diseño del software. De igual manera, quienes utilicen ésta herramienta podrán realizar su trabajo en base a un diseño adecuado que

[1] European Computer Manufacturers Association – Organización internacional basada en membrecías de estándares para la comunicación y la información.

disminuya el peligro de retrasos y pérdida de tiempo, incrementando las posibilidades de obtener un producto exitoso con los recursos estimados en la planificación inicial.

La herramienta de software libre **AleDan UML** pretende evitar que analistas de sistemas tengan que trabajar con software comercial, muchas de las veces ilegal, gracias a que posee buenas características de funcionalidad acordes a las necesidades de nuestro entorno de trabajo.

Así mismo queda abierta la posibilidad de que a futuro estudiantes o egresados de la carrera puedan realizar modificaciones con la finalidad de mejorar la funcionalidad y operatividad de ésta herramienta, o bien, de integrar nuevos módulos al software para ampliar su alcance, buscando siempre optimizar las condiciones de trabajo de los desarrolladores de software no sólo de nuestra localidad, sino también, de la sociedad en general.

5. METODOLOGÍA

Durante el desarrollo del presente trabajo investigativo se hizo uso de las diferentes técnicas, métodos, herramientas y procedimientos que la Investigación Científica y el Desarrollo de Software ponen a nuestra disposición, para aplicar, sistematizar y descubrir nuevos conocimientos. Su finalidad, conducirnos a dar una alternativa de solución a la problemática identificada.

5.1. MÉTODOS

5.1.1. Metodología para el Desarrollo de la Investigación

Con el propósito de seguir un proceso ordenado y lógico durante el desarrollo de nuestro proyecto de Tesis se creyó conveniente apoyarnos esencialmente en los siguientes métodos lógicos:

- **Método Inductivo** (de lo particular a lo general)**.-** Permitió analizar las particularidades de la realidad actual sobre el uso de Herramientas CASE para la generación de código fuente a partir del diseño de la arquitectura del software. En base a dicho análisis, por una parte se pudo realizar el diagnóstico de la problemática identificada en nuestro entorno para extraer sus causas, características y actores principales, y por otra, se pudo realizar el planteamiento de hipótesis. Asimismo gracias a la generalización de los datos recogidos a partir de las encuestas aplicadas a desarrolladores de software se pudo dar paso a la construcción de la propuesta alternativa y a la elaboración de conclusiones y recomendaciones.

- **Método Deductivo** (de lo general a lo particular)**.-** Apoyándonos en el análisis de nuestra hipótesis general, éste método nos permitió deducir que el desarrollo de una Herramienta CASE que se adapte a las principales necesidades de los desarrolladores de software de nuestro medio, indiscutiblemente coadyuvaría a la solución de algunos de los problemas más imperantes no sólo en la Universidad Nacional de Loja sino de la sociedad en general. De igual manera obteniendo, clasificando y analizando la información recopilada en la investigación teórica, se logró enriquecer aún más la propuesta alternativa, las conclusiones y recomendaciones planteadas.

5.1.2. Metodología para el Desarrollo de la Herramienta

Para el diseño y desarrollo de la herramienta **AIeDan UML**, hemos creído conveniente emplear la **Metodología Ágil para el Desarrollo de Software RUP** (Rational Unified Process – Proceso Unificado de Modelado), debido a la simplicidad de sus reglas y prácticas, a su orientación a equipos de desarrollo de tamaño pequeño, a su flexibilidad ante los cambios, a su ideología de colaboración y a que define un ciclo de vida iterativo que prioriza el uso de lenguajes de modelado y casos de uso.

El proceso de ciclo de vida RUP se divide en cuatro fases bien definidas, que a su vez se subdividen en iteraciones, cada una de las cuales produce una pieza de software demostrable:

- **Inicio o Incepción.-** Dentro de ésta fase se establecieron los requerimientos generales del proyecto y su alcance. También se identificaron los casos de uso que orientan la funcionalidad de la herramienta para diseñar las arquitecturas preliminares y así estimar el cronograma[2] y presupuesto del proyecto.

- **Elaboración.-** Se analizó el dominio de la problemática identificada, y en base a ello se definieron en detalle las actividades a llevar a cabo por cada responsable[3]. Posteriormente se realizó el filtrado de requerimientos, la identificación de casos de uso definitivos y la documentación de la arquitectura de la herramienta (diagramas), con la finalidad de elaborar un prototipo base del sistema.

- **Construcción.-** De manera progresiva se desarrolló, integró y verificó los módulos que formarían parte de la herramienta, para lo cual se construyeron varias versiones preliminares de la aplicación, de tal manera que en cada una de ellas se pueda realizar la depuración de errores encontrados e incorporar paulatinamente cada una de las prestaciones establecidas inicialmente. Así mismo se dio paso a la elaboración de los manuales dirigidos tanto a usuarios como a desarrolladores.

- **Transición.-** Finalmente, dentro de ésta etapa, se realizó la depuración completa del producto final, es decir, se hicieron las últimas correcciones y se agregaron los detalles finales a la aplicación y a los manuales de usuario y de programador.

---

[2] Ver **Cronograma** en la página 294

[3] Ver **Matriz de operatividad de objetivos específicos** en la página 303

5.2. MATERIALES

- Fuentes bibliográficas e internet para recopilar información sobre la Fundamentación Teórica de la investigación, así como también para desarrollar el producto software planteado.

- Hardware y software que posibilitaron la documentación y el desarrollo de la aplicación:

  - ☞ Un ordenador Intel Pentium 4, CPU de 2.00GHz, 512 de RAM.
  - ☞ Un ordenador Intel Pentium 4, CPU de 3.00GHz, 512 de RAM.
  - ☞ Una impresora Canon iP1000.
  - ☞ Sistema Operativo: Windows XP Professional Versión 2002 compilación 6200, Ubuntu Utimate Edition 8.1.
  - ☞ Herramienta para la Planificación: Microsoft Office Project 2003.
  - ☞ Herramienta para la Documentación: OpenOffice 3.0.
  - ☞ Lenguaje de Programación: Java, C#.
  - ☞ Herramienta para codificación: NetBeans 6.7.1.
  - ☞ Herramienta para diseño de arquitectura: Enterprise Architect 7.1, NetBeans 6.7.1.
  - ☞ Herramienta para diseño gráfico: Gimp.

5.3. TÉCNICAS DE TRABAJO

La realización del presente proyecto presupone la práctica de ciertas técnicas que contribuyan al proceso investigativo en la recolección de datos y que sirvan como instrumentos de medición y de prueba:

- **Observación Directa.-** Ésta técnica nos permitió conocer en parte la realidad de la Carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja, para determinar los problemas existentes en la misma, y seleccionar el objeto de nuestro proyecto de investigación: *"la falta de una Herramienta para el análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp".*

Esta técnica de trabajo fue de gran apoyo para vincularnos con dicho problema, y con sus fuentes de información teóricas y empíricas.

- **Encuestas.-** Ésta técnica fue dirigida en dos ocasiones a los desarrolladores de software de la Carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja y demás profesionales de la rama, con la finalidad de obtener información relacionada con el problema planteado.

  La primera de ellas fue encaminada a conocer cuáles son las experiencias y expectativas de los desarrolladores en el uso de herramientas UML. En base a ésta información, se realizó la recolección de material bibliográfico, la organización del material recogido, el procesamiento de los datos, su análisis y documentación, pero más importante aún, la determinación de requerimientos de la aplicación.

  La segunda encuesta por su parte tuvo como objetivo conocer el nivel de satisfacción del usuario con el manejo de la herramienta, y es por ello que fue realizada al culminar el proceso de codificación.

6. FUNDAMENTACIÓN TEÓRICA

CONTENIDO

Lenguaje Unificado de Modelado UML

Lenguaje de Programación C Sharp

Herramientas CASE

LENGUAJE UNIFICADO DE MODELADO UML

6.1.1. Definiciones

Un modelo es la simplificación de la realidad observada. En el campo del desarrollo de software, el objetivo que persigue el modelado de un sistema, es obtener las piezas más importantes del mismo. Para ello se procede a realizar una abstracción de esa realidad y se la plasma con una notación gráfica.

A ésta representación se la conoce como modelo visual y lo que permite es manejar la complejidad de los sistemas a analizar o diseñar. De ahí es que nace la necesidad de contar con una herramienta que permita visualizar, comprender y entender que será y cómo se hará un sistema, antes de ponerlo en marcha.

El **Lenguaje Unificado de Modelado UML (Unified Modeling Language)** nace cuando Rumbaugh, Booch y Jacobson unifican sus estudios basándose en una semántica y notación estándares, con el objetivo de lograr compatibilidad en el análisis y diseño orientado a objetos. De ésta manera se consigue que los proyectos sean realizados en un lenguaje de modelado maduro, dando a los desarrolladores la oportunidad de enfocarse en producir software de calidad.

El UML es un lenguaje gráfico estandarizado y orientado a objetos, que ayuda a escribir los “planos” del software, con la finalidad de visualizar, especificar, construir y documentar los artefactos de un sistema de información. "Incluye conceptos conceptuales tales como procesos de negocio y funciones del sistema, y algunos aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables”.[4]

UML está conformado por varios elementos, artefactos o figuras que tienen un significado propio. Mediante la agrupación de algunos de estos elementos (según sea necesario), se pueden obtener diagramas que modelan diferentes aspectos del sistema a desarrollar, desde sus vistas lógicas y físicas, hasta los aspectos dinámicos, estáticos y funcionales del mismo.

Esta notación gráfica muy expresiva, también permite representar en mayor o menor grado todas las fases que un proyecto informático debería seguir para ser exitoso: por

[4] WIKIPEDIA. Lenguaje Unificado de Modelado. http://es.wikipedia.org/wiki/UML, 28 de noviembre del 2008

ejemplo, apoyarnos en los casos de uso durante la etapa de análisis; en diagramas de clases, objetos, etc., en la fase de diseño; o definir la implementación y configuración del sistema mediante diagramas de despliegue.

UML es totalmente independiente del lenguaje de programación, es decir, los diseños realizados utilizando éste estándar pueden ser implementados en cualquier lenguaje, principalmente los orientados a objetos.

6.1.2. Historia

UML tuvo sus inicios en octubre de 1994, cuando dos investigadores de la metodología del software, Rumbaugh y Booch, decidieron unificar los métodos que cada uno de ellos había desarrollado por separado: el método OMT (Object Modelling Tool) y el método Booch.

En octubre de 1995 suceden dos acontecimientos importantes: por una parte se presenta el primer borrador del lenguaje UML, y por otra, Jacobson, otro entusiasta del área de la metodología del software, se une al equipo para aportar algunas de sus ideas al lenguaje en desarrollo. Con el avance del proyecto, se solicitó la colaboración de varias empresas para que aportaran ideas, recomendaciones o sugerencias, con el propósito de enriquecer UML.

Con todas estas colaboraciones de por medio, se pudo concretar la primera versión del lenguaje gráfico para el modelado UML, la misma que fue entregada a un grupo de trabajo del OMG (Object Management Group). Este grupo formuló algunas modificaciones para que sean incrementadas en una nueva versión de UML (la 1.1), la misma que sería finalmente adoptada por el OMG como estándar en noviembre de 1997. La versión 2.0 de UML fue lanzada en octubre de 2004, aunque desde el año 2007 se cuenta ya con la versión 2.1.

6.1.3. Funciones de UML

- Permite que los desarrolladores representen gráficamente un sistema, de tal manera que otro u otros lo puedan leer.
- Permite especificar las características de un sistema antes de su construcción.
- Permite documentar el sistema desarrollado a través de sus elementos gráficos.

- A partir de los modelos realizados en éste lenguaje, se pueden construir los sistemas diseñados.

6.1.4. Estructura de un Modelo

Un modelo está compuesto por tres diferentes bloques de construcción:

- **Clasificadores,** que son abstracciones de los objetos o entidades que se desea representar.
- **Relaciones,** que permiten relacionar entre sí a los objetos o entidades representados.
- **Diagramas,** que son un conjunto de clasificadores con sus respectivas relaciones.

6.1.4.1. Clasificadores

Figura 1: Artefacto utilizado para representar un Clasificador

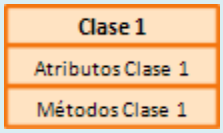
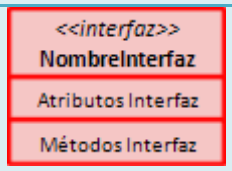
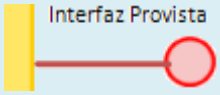
Un clasificador es un mecanismo de UML que describe las características estructurales y de comportamiento de objetos o entidades. Como puede observarse en la Figura 1, la notación predefinida para un clasificador es la siguiente:

- Se representa como un rectángulo que contiene el nombre del clasificador, opcionalmente con compartimientos adicionales para describir sus características (atributos y métodos).
- El nombre de un clasificador, atributo o método debe describir de manera clara y sencilla lo que representa. Si está conformado por varia palabras, el nombre se forma encadenando dichas palabras, utilizando mayúsculas para la primera letra de cada palabra, excepto el caso de métodos y atributos cuyo nombre siempre debe empezar con una letra minúscula.
- Adicionalmente, el nombre de un clasificador deberá ser escrito en negrita, centrado y con la primera letra en mayúscula.
- El tipo específico del clasificador puede mostrarse sobre el nombre del mismo como un estereotipo.

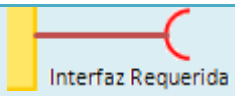
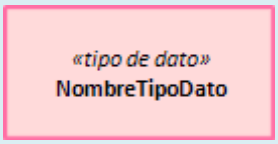
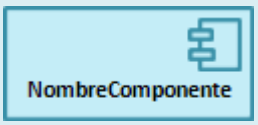
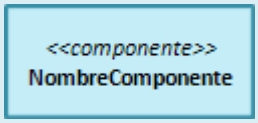
- Los estereotipos deberán estar centrados sobre el nombre del clasificador.
- En caso de que se trate de un clasificador abstracto, su nombre se muestra en cursiva, o bien, usando el estereotipo[5] «abstract» después de, o sobre su nombre.

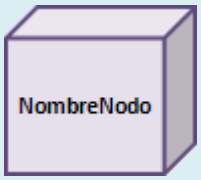
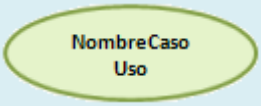
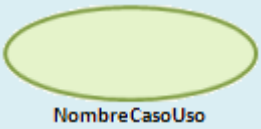
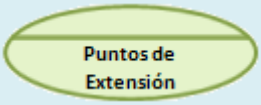
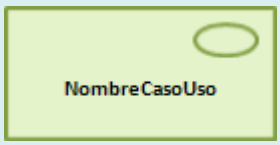
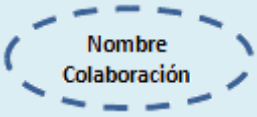
A través de la Tabla 1 se muestran las especializaciones más importantes que puede adoptar un clasificador:

Tabla 1: Tipos de Clasificador

Artefacto	Descripción
	**Clase.-** Describe un conjunto de objetos que comparten las mismas especificaciones de comportamiento, restricciones y semántica. Representa cualquier objeto o entidad de la realidad que se desea modelar. Gráficamente se muestra como un clasificador. A menudo se muestra con tres compartimientos en los que se deberá colocar el nombre de la clase, los atributos que posea y los métodos que deberá realizar.
 Notación expandida  Notación icónica  Notación Lollipop	**Interfaz.-** Es una colección de operaciones usada para especificar un servicio de una clase o de un componente. En caso de que posea atributos, éstos deberán ser constantes. Todas las operaciones de la interfaz son públicas y abstractas. Una interfaz puede ser realizada por muchas clases o componentes, y una clase o componente pueden ser realizados por muchas interfaces. Gráficamente una interfaz se representa con notación expandida, como un clasificador con el estereotipo «interfaz»; con notación icónica, mediante un círculo etiquetado con el nombre de la interfaz; con notación lollipop (única para interfaces provistas); o bien, con notación socket (única para interfaces requeridas). UML define dos tipos de interfaz: **1.** Interfaz provista.- Es aquella implementada por un clasificador. Si la realización de una interfaz (clasificador → interfaz) se representa con notación lollipop, se muestra a la interfaz como un círculo etiquetado con su nombre unida al clasificador que la realiza por una línea sólida. Si la interfaz es representada en forma expandida o icónica, la relación a utilizar será una realización.

[5] Revisar la descripción de **estereotipo** en la Tabla 3: Mecanismos Comunes de la página 32

 Notación Socket	**2.** Interfaz requerida.- Es aquella que un clasificador necesita para realizar su cometido. Si el uso de una interfaz (clasificador → interfaz) se representa con notación socket, se muestra a la interfaz como un semicírculo etiquetado con su nombre, conectada al clasificador que la requiere por una línea sólida. Si la interfaz es representada en forma expandida o icónica, la relación a utilizar será una dependencia.
	**Tipo de Dato.-** Es un tipo especial de clasificador similar a una clase, con la diferencia de que sus instancias son identificadas únicamente por su valor. Puede contener atributos para mapear tipos de datos estructurados. Comúnmente es usado para representar tipos primitivos en los lenguajes de programación (integer, string, etc.). Un tipo de dato es denotado como un rectángulo con el estereotipo «dataType», o bien, mediante una cadena de texto con el nombre del tipo de dato (cuando es el referenciado por un atributo por ejemplo).
 	**Componente.-** Es una unidad independiente que encapsula el estado y comportamiento de varios clasificadores. Especifica un contrato formal de los servicios que provee a sus clientes y de los que requiere de otros componentes o servicios del sistema (a través de sus interfaces provistas y requeridas). También puede ser definido como una parte física y reemplazable de un sistema, que representa los elementos participantes en su ejecución (librerías, tablas, archivos, clases, paquetes, etc.) y cómo se encuentran organizados dentro de dicho sistema. Un componente puede poseer puertos para formalizar sus puntos de interacción con interfaces provistas y requeridas (un componente puede tener varias interfaces provistas y requeridas). Gráficamente un componente se muestra como un clasificador con el estereotipo «component». Opcionalmente, en la esquina superior derecha se puede presentar el icono de componente utilizado en versiones anteriores de UML. También se pueden listar sus interfaces y/o propiedades en compartimientos adicionales dentro del símbolo del componente.
	Nodo.- Es un elemento físico que existe en tiempo de ejecución. Representa un recurso computacional que tiene algo de memoria y capacidad de procesamiento, sobre el que se pueden desplegar

	y ejecutar los componentes del sistema. Los nodos son usados para modelar la topología del hardware sobre la que se ejecuta un sistema software. Gráficamente un nodo se muestra como la vista tridimensional de un cubo. Si se desea incluir los servicios que ofrece el entorno de ejecución, se pueden listar en el interior del símbolo del nodo.
   	**Caso de Uso.-** Es la interacción con el sistema a desarrollar, donde se representan los requisitos funcionales del mismo. Es una narración hablada o escrita del comportamiento que tiene o que va a tener el escenario. Describe lo que hace un sistema (subsistema, clase o interfaz) pero no especifica cómo lo hace. Obligatoriamente deben dejar algo de valor para el sistema. La conducta de un caso de uso se puede especificar describiendo un flujo de eventos en un texto bastante claro. Cuando se describe este flujo de eventos, se debe incluir cómo y cuándo un caso de uso empieza y termina, cuándo el caso de uso interactúa con los actores y qué objetos son intercambiados, y finalmente, el curso normal y los cursos alternos de comportamiento. Gráficamente este clasificador se muestra como una elipse, cuyo nombre podrá ir dentro del artefacto, o bien, bajo él. Un caso de uso también puede usar la notación de rectángulo propia de un clasificador, con un icono de elipse en la esquina superior derecha. De una u otra manera se pueden agregar compartimientos adicionales para listar sus propiedades o sus puntos de extensión. En el último caso, el compartimiento deberá llevar como título **Puntos de Extensión** y su correspondiente descripción de forma clara y entendible.
	**Colaboración.-** Describe una estructura de objetos que cooperan para conseguir una determinada funcionalidad. Generalmente son usadas para especificar la realización de casos de uso y operaciones, y para modelar arquitectónicamente los mecanismos más significativos de un sistema.

6.1.4.2. Relaciones

Una relación es una conexión entre objetos que muestra la manera como interactúa cada uno de ellos. Gráficamente se representa como un camino trazado con diferentes

tipos de líneas que puede incluir ciertos adornos para aclarar el tipo de interacción existente entre los elementos conectados. Tales adornos son los siguientes:

- **Nombre,** descripción clara y precisa de la relación que existe entre clasificadores.
- **Navegabilidad,** grado de conocimiento que tiene un objeto respecto a otro en la relación. Puede ser unidireccional o bidireccional. Se representa mediante una punta de flecha dirigida.
- **Rol,** papel específico que cumple un elemento en una relación. Este deberá ser colocado en uno de los extremos de la relación.
- **Multiplicidad,** permite identificar cuántos objetos intervienen en una relación. Puede ser de tipo:
  - 0 … 1 → 0 a exactamente 1
  - 1 → exactamente 1
  - 0 … * → 0 a muchos
  - 1 … * → 1 a muchos
  - n → exactamente n objetos, donde n > 0
  - 0 … n → 0 a n, donde n > 0
  - 1 … n → 1 a exactamente n, donde n > 1
  - 1 … 0, 1 → 1 a ninguno o a uno
  - 1 … 4..6 → 1 a cuatro hasta seis

UML reconoce cuatro tipos de relaciones: Asociación, Generalización, Dependencia y Realización.

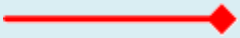
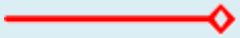
6.1.4.2.1. Asociación

En el UML una Asociación es definida como una relación semántica entre dos o más clasificadores que involucra conexiones entre sus instancias. En su representación gráfica se pueden incluir cualquiera de los adornos permitidos dentro de una relación[6].

Una Asociación puede especializarse en algunos subtipos tal y como se muestra en la Tabla 2:

[6] Los adornos que puede tener una Asociación se describen en el apartado "6.1.4.2 Relaciones" de la página 28

Tabla 2: Tipos de Asociación

Artefacto	Descripción
	Asociación Simple.- Describe una conexión entre objetos.
	**Composición.-** Cada componente pertenece solamente a un todo. Un objeto puede pertenecer sólo a una Composición.
	**Agregación.-** Representa una relación entre un todo y sus partes. Se lee “es una parte de”.
	**Asociación Reflexiva.-** Se da cuando un clasificador se asocia consigo mismo.

6.1.4.2.2. Generalización

Figura 2: Artefacto utilizado para representar una Generalización

Conocida también como herencia, la generalización es una relación en la que el elemento especializado (el hijo) comparte la estructura y la conducta del elemento generalizado (el padre). Gráficamente una generalización se representa como una línea sólida con una punta de flecha vacía que apunta al padre.

6.1.4.2.3. Dependencia

Figura 3: Artefacto utilizado para representar una Dependencia

Es una relación semántica entre dos objetos, en donde un cambio a un objeto (el objeto independiente) puede afectar la semántica del otro objeto (el objeto dependiente). Su representación gráfica es una línea discontinua en la que se puede colocar cualquiera de los adornos permitidos dentro de una relación[7].

6.1.4.2.4. Realización

Figura 4: Artefacto utilizado para representar una Realización

[7] Los adornos que puede tener una Dependencia se describen en el apartado “6.1.4.2 Relaciones” de la página 28

Es una relación semántica entre clasificadores, en donde un clasificador especifica un convenio que otro clasificador garantiza llevar a cabo. Una realización se podrá utilizar en dos circunstancias: entre interfaces y clases o componentes que las contienen, y entre casos de uso y colaboraciones. Gráficamente se representa con una línea discontinua con una punta de flecha vacía.

6.1.4.3. Diagramas

Debido a la amplitud del tema, los diagramas serán abordados en el punto 1.6 expuesto a continuación.

6.1.5. Diagramas

Un diagrama es la representación gráfica de un sistema o subsistema, que permite comprender de mejor manera cómo interactúan sus elementos en la realidad (sistema actual) y cómo se desea que interactúen en el sistema a desarrollar.

Gráficamente un diagrama es un conjunto de vértices y arcos. Sin embargo, lo realmente valioso de un diagrama es su significado y no su imagen gráfica.

Los diagramas son usados para visualizar un sistema desde diferentes perspectivas, puesto que ningún sistema complejo puede entenderse totalmente desde un único punto de vista. Si están bien elaborados, los diagramas pueden lograr que el sistema en desarrollo sea entendible y accesible.

Con el propósito de realizar una representación correcta de un sistema, la versión 2.0 de UML ofrece trece diferentes tipos de diagramas organizados jerárquicamente, tal como se muestra en la Figura 5:

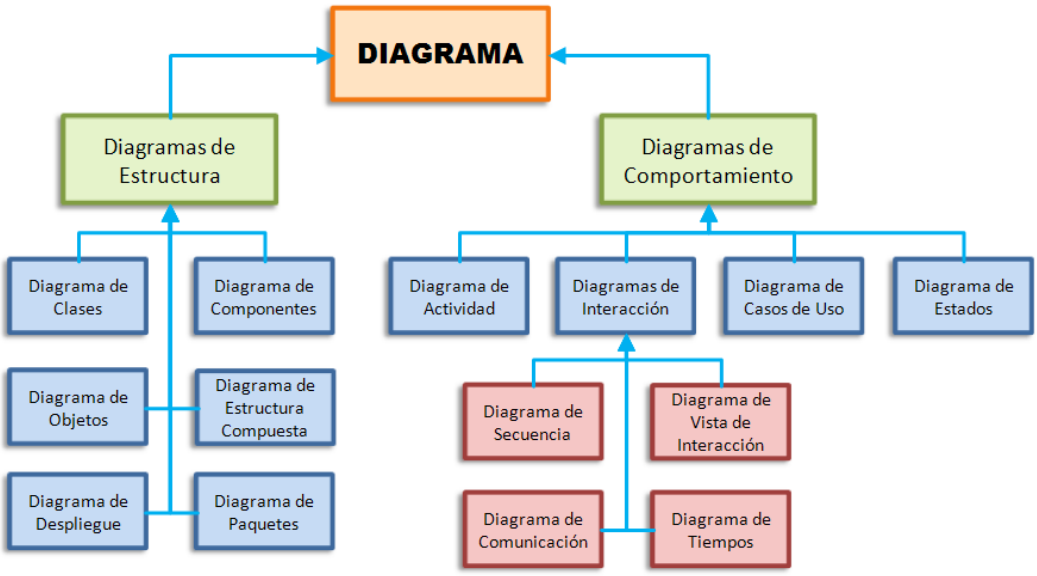
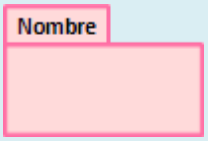
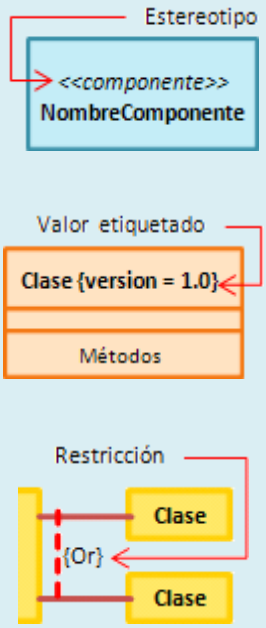


Figura 5: Tipos de Diagrama UML 2.0

Además de los artefactos propios de cada tipo de diagrama, existen algunos elementos que son comunes para todos ellos. Estos se describen en la Tabla 3:

Tabla 3: Mecanismos Comunes

Artefacto	Descripción
	**Marco.-** En UML 2.0 los diagramas se enmarcan dentro de un recuadro que posee una etiqueta para indicar el tipo de diagrama al que pertenece: **1.** pkg.- Diagrama de Paquetes. **2.** cmp.- Diagrama de Componentes. **3.** uc.- Diagrama de Casos de Uso. **4.** act.- Diagrama de Actividades. **5.** stm.- Diagrama de Estados. **6.** sd.- Diagrama de Secuencia.
	**Paquete.-** Ver descripción en la "Tabla 9: Elementos de un Diagrama de Paquetes" de la página 44. El paquete es el único elemento de agrupamiento permitido en cualquiera de los diagramas UML. Es un mecanismo de propósito general que permite modelar un sistema en grupos organizados de elementos, de tal manera que se los pueda entender más fácilmente.
	**Nota.-** Es un símbolo gráfico para mostrar restricciones o comentarios a un elemento o una colección de elementos. Para adjuntar una nota a un elemento se utiliza una línea discontinua.

	También se puede asociar una nota a un atributo específico. Una nota puede contener imágenes o texto.
	**Mecanismos de Extensibilidad.-** Por un lado permiten a los usuarios adaptar las especificaciones dadas por UML a las necesidades de un proyecto, y por otro, que UML se ajuste fácilmente a la nueva tecnología de software, y con ello, a la gran diversidad de lenguajes de programación. Estos mecanismos pueden ser: **1.** Estereotipo.- Es una extensión del vocabulario del UML, que permite crear nuevos tipos de bloques similares a los ya existentes pero que se especializan para un problema específico. Permite tomar elementos propios de UML y convertirlos en otros. Un estereotipo se muestra dentro de los símbolos "« »", utilizando palabras que lo definan de forma clara y sencilla (por ejemplo, «interfaz»). **2.** Valor etiquetado.- Es una extensión de las propiedades de un elemento UML, que permite crear nueva información en la especificación de ese elemento. **3.** Restricción.- Es una extensión de la semántica de un elemento UML, que permite agregar nuevas reglas o modificar las ya existentes. Una restricción especifica condiciones que deben cumplirse en su totalidad antes de continuar con el proceso.

6.1.5.1. Diagramas de Estructura

Ayudan a visualizar, especificar, construir y documentar los aspectos estáticos de un sistema. Se dice que se trata de un modelado estático porque no se indican los cambios que van a tener los objetos durante su ejecución. Un Diagrama de Estructura se centra en enfatizar los elementos que deben existir en el sistema modelado tales como clases, interfaces, colaboraciones, componentes y nodos.

6.1.5.1.1. Diagrama de Clases

Muestra un conjunto de clases, interfaces, colaboraciones y sus relaciones, para lo cual, el analista deberá definir las características de cada uno de dichos elementos. Un diagrama de éste tipo, puede verse como la Figura 6:

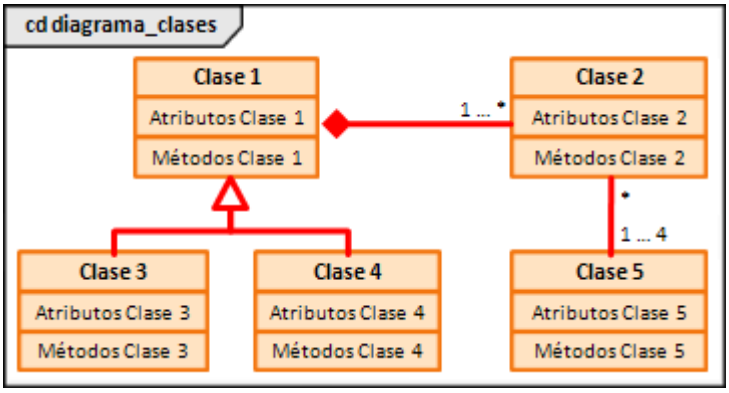
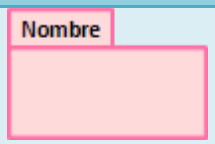
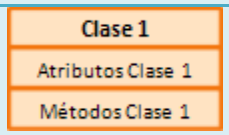
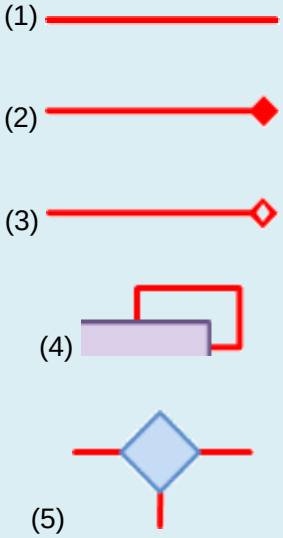


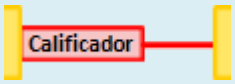
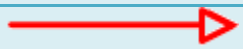
Figura 6: Diagrama de Clases

Para tener un concepto más claro de los artefactos que participan en un Diagrama de Clases, a continuación se presenta la Tabla 4:

Tabla 4: Elementos de un Diagrama de Clases

Artefacto	Descripción
	**Paquete.-** Ver descripción en la "Tabla 9: Elementos de un Diagrama de Paquetes" de la página 44.
	**Clase.-** Ver descripción en la "Tabla 1: Tipos de Clasificador" de la página 26.
	**Interfaz.-** Ver descripción en la "Tabla 1: Tipos de Clasificador" de la página 26.
	**Asociación[8].-** Pueden aparecer dos asociaciones entre clases en el mismo diagrama, o bien, pueden asociarse diversas clases con una en particular. En un Diagrama de Clases se puede utilizar: 1. Asociación simple 2. Composición 3. Agregación.- Una clase (el todo) está formada por otras (sus partes). Cuando una clase tiene muchos atributos, conviene dividirla en clases auxiliares y usar agregaciones para reunirlas. 4. Asociación Reflexiva 5. Asociación Tripartita.- Se establece entre más de dos clases

[8] Ver descripción de Asociación Simple, Composición, Agregación y Asociación Reflexiva en la "Tabla 2: Tipos de Asociación" de la página 30.

(6) 	cuando una misma clase puede desempeñar distintos roles. 6. Asociación Calificada.- Los objetos en el rol de "muchos" se pueden identificar de forma no ambigua mediante un atributo denominado calificador (un calificador actúa como un selector entre los objetos reduciendo la multiplicidad efectiva de "muchos" a "uno").
	**Generalización.-** Una clase puede tener cero o un padre.
	**Dependencia.-** Una clase utiliza a otra. Opcionalmente puede tener cualquiera de los adornos permitidos en una relación.[9]
	**Realización.-** Semánticamente es una combinación entre la dependencia y generalización. Se usa para especificar la relación entre una interfaz y una clase que proporciona una operación o servicio para ella.

6.1.5.1.2. Diagrama de Componentes

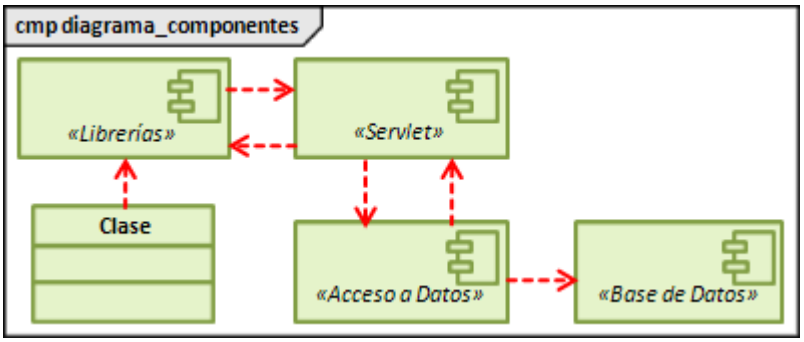


Figura 7: Diagrama de Componentes

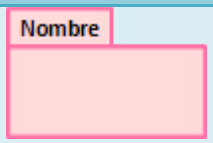
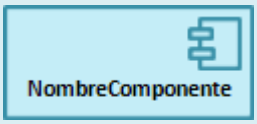
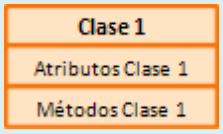
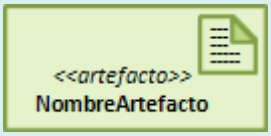
Tal como se puede observar en la Figura 7, éste tipo de diagrama muestra un conjunto de componentes y sus relaciones, permitiendo a los analistas realizar el diseño de cómo quedará un sistema en su parte física.

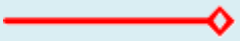
Normalmente los Diagramas de Componentes se utilizan para modelar código fuente, código ejecutable y bases de datos físicas. Estos se relacionan con los Diagramas de Clase en donde un componente típico mapea a una o a más clases, interfaces o colaboraciones.

[9] Los adornos que puede tener una Dependencia se describen en el apartado "6.1.4.2 Relaciones" de la página 28.

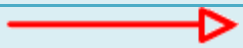
Los elementos que se pueden utilizar en éste tipo de diagrama se enuncian en la Tabla 5:

Tabla 5: Elementos de un Diagrama de Componentes

Artefacto	Descripción
	**Paquete.-** Ver descripción en la "Tabla 9: Elementos de un Diagrama de Paquetes" de la página 44.
	**Componente.-** Ver descripción en la "Tabla 1: Tipos de Clasificador" de la página 26. Las operaciones de un componente sólo pueden ejecutarse a través de su interfaz. Los clasificadores internos que realizan el comportamiento de un componente pueden desplegarse por medio de dependencias generales, pueden anidarse dentro de la forma del componente, o bien, desplegarse anidados dentro del componente.
	**Clase.-** Ver descripción en la "Tabla 1: Tipos de Clasificador" de la página 26. La representación gráfica de una clase se puede omitir nombrando su nombre en el interior del componente correspondiente.
	**Interfaz.-** Ver descripción en la "Tabla 1: Tipos de Clasificador" de la página 26. Una interfaz se conecta al componente que la implementa a través de una realización, y al componente que utiliza sus servicios con una dependencia. Las interfaces también se pueden mostrar como una lista dentro del símbolo de componente, separando las interfaces ofrecidas o provistas y las requeridas. **1.** Interfaz provista.- Interfaz realizada o implementada por un componente, por uno de sus clasificadores o por uno de sus puertos. **2.** Interfaz requerida.- Interfaz usada por un componente a través de una relación de dependencia desde el componente, desde uno de sus clasificadores realizados, o desde uno de sus puertos.
	**Artefacto.-** Un artefacto es la especificación de una pieza física de información usada o producida por un proceso de desarrollo de software, o por despliegue y operación de un sistema. Son utilizados para representar cualquier elemento empaquetable (por

	ejemplo un jar, modelo de archivos, archivos fuente, scripts y archivos ejecutables binarios, una tabla en un sistema de base de datos, un documento de procesamiento de textos, un mensaje de correo, etc.). Los artefactos que manifiestan a un componente se pueden listar dentro del símbolo del componente. Los artefactos implementados por un componente pueden conectarse a él por contención física (anidamiento) o por una Asociación estereotipada «implement» entre el Componente y Artefacto.
	**Puerto.-** Es una propiedad de un clasificador que especifica un punto de interacción entre el clasificador y su ambiente o entre el clasificador y sus partes internas. Un puerto puede especificar los servicios que un clasificador provee (ofrece) a su ambiente así como los servicios que un clasificador espera (requiere) de su ambiente. Gráficamente se muestra como un cuadrado pequeño, con un nombre colocado cerca de él, seguido de su multiplicidad (entre corchetes "[ ]"). Si no se necesita, puede omitirse el nombre y la multiplicidad. El tipo de un puerto puede mostrarse seguido de su nombre, separado por dos puntos (":"). Si el puerto se grafica sobre la línea límite del componente, este puerto es expuesto, es decir, su visibilidad es pública. Si el puerto se muestra dentro del símbolo del componente, entonces el puerto está oculto y su visibilidad es como se especifique (protegido por defecto).
(1) (2)  (3) 	**Asociación[10].-** Indica que un componente se refiere a los servicios ofrecidos por otro componente. Opcionalmente, esta relación puede contar con un nombre y estereotipo que la identifique. En un Diagrama de Componentes se puede utilizar: **1.** Asociación Simple **2.** Agregación **3.** Conector de ensamble.- Es un conector (tipo de Asociación) entre dos componentes que establece que uno de ellos provee los servicios que el otro requiere. Está definido desde una interfaz requerida o de un puerto a una interfaz provista o a un puerto. **4.** Conector de delegación.- Es un conector que une el contrato externo de un componente a la realización interna del

[10] Ver descripción de Asociación Simple y Agregación en la "Tabla 2: Tipos de Asociación" de la página 30.

	comportamiento de las partes de ese componente. Al usar un conector de delegación se conectan los trabajos internos del sistema con el mundo exterior, por una delegación de las conexiones de las interfaces externas. Si el elemento realiza la interfaz, la flecha apunta desde el puerto al elemento. Si el elemento usa la interfaz, la flecha apunta desde el elemento hasta el puerto. Si se representan las interfaces de los elementos internos, los conectores de delegación se unen a la interfaz.
	**Generalización**
	**Dependencia.-** Se usa cuando: **1.** Un componente necesita los servicios ofrecidos por otro componente para implementar su funcionamiento. **2.** Conectar las clases de un componente a ese componente. **3.** Un artefacto usa a otro.
(1)  (2)	**Realización.-** Cuando se trate de una interfaz implementada, la realización se dibujará: **1.** Como una línea discontinua con punta de flecha vacía (si la interfaz usa notación expandida[11]). **2.** Como una línea sólida (si la interfaz usa notación icónica, lollipop o socket).

6.1.5.1.3. Diagrama de Objetos

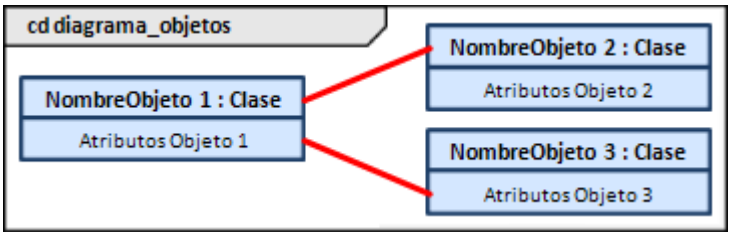


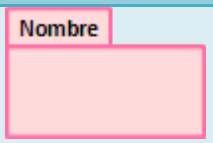
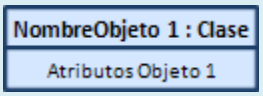
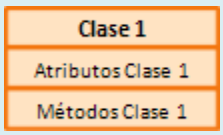
Figura 8: Diagrama de Objetos

Muestra un conjunto de objetos y la relación existente entre ellos en un momento determinado de la aplicación (ver Figura 8). Representan una instantánea estática de los objetos que forman parte de un Diagrama de Clases, motivo por el cual, el analista deberá decidir previamente cuál es la situación del sistema que se desea representar.

[11] Ver los tipos de notación utilizados para representar una Interfaz en la "Tabla 1: Tipos de Clasificador" de la página 26.

Los elementos gráficos que se deben tomar en consideración para elaborar un Diagrama de Objetos son los que se muestran a continuación en la Tabla 6:

Tabla 6: Elementos de un Diagrama de Objetos

Artefacto	Descripción
	**Paquete.-** Ver descripción en la "Tabla 9: Elementos de un Diagrama de Paquetes" de la página 44.
	**Objeto.-** Representa las instancias creadas en un momento determinado de la ejecución. En el casillero superior se coloca el comportamiento del objeto en la forma [Nombre de objeto: Nombre de clase], mientras que en el inferior se colocan los atributos de dicho objeto.
	**Clase.-** Ver descripción en la "Tabla 1: Tipos de Clasificador" de la página 26. Este artefacto se utiliza opcionalmente cuando se desee mostrar de manera explícita la clase a partir de la cual se extiende uno o varios objetos.
	Vínculo (Link).- Es una instancia de Asociación que especifica un camino a lo largo del cuál un objeto puede despachar un mensaje a otro (o al mismo) objeto. Si es necesario puede tener cualquiera de los adornos permitidos en una relación.[12]

6.1.5.1.4. Diagrama de Estructura Compuesta

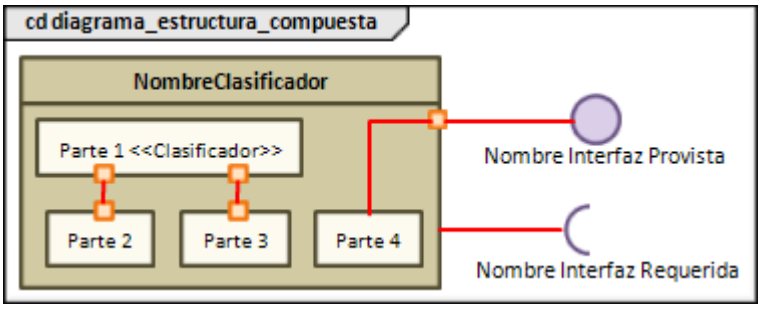


Figura 9: Diagrama de Estructura Compuesta

El diagrama descrito en la Figura 9 refleja la colaboración interna de clases, interfaces y componentes para describir una funcionalidad. Muestra la estructura interna de un

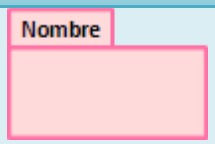
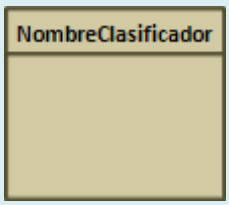
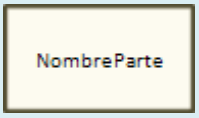
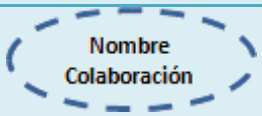
[12] Los adornos que puede tener un Vínculo se describen en el apartado "6.1.4.2 Relaciones" de la página 28.

clasificador, incluyendo sus puntos de interacción a otras partes del sistema. Este diagrama permite que todos los clasificadores puedan tener una estructura compuesta.

El comportamiento de las instancias de otros clasificadores contenidos (estructura interna) en un clasificador determinado, puede especificarse como una colaboración.

Los artefactos utilizados en un diagrama de éste tipo se describen en la Tabla 7:

Tabla 7: Elementos de un Diagrama de Estructura Compuesta

Artefacto	Descripción
	**Paquete.-** Ver descripción en la "Tabla 9: Elementos de un Diagrama de Paquetes" de la página 44.
	**Clasificador.-** Ver descripción en la "Tabla 1: Tipos de Clasificador" de la página 26. Representa a cualquier clasificador, generalmente una clase, cuyo comportamiento puede ser completa o parcialmente descrito mediante interacciones entre partes.
	**Parte.-** Representa el papel que cumple un objeto en un momento dado. Puede incluir un factor de multiplicidad[13]. Cuando el clasificador contenedor se destruye, todas sus partes se destruyen con él. Se muestra como un rectángulo e indica los objetos que conforman al objeto principal.
	**Puerto.-** Ver descripción en la "Tabla 5: Elementos de un Diagrama de Componentes" de la página 36.
	**Interfaz.-** Ver descripción en la "Tabla 1: Tipos de Clasificador" de la página 26.
	**Colaboración.-** Ver descripción en la "Tabla 1: Tipos de Clasificador" de la página 26.
	Conector.- Es una instancia de Asociación que especifica un enlace (link) entre partes. Indica la relación entre las partes internas del clasificador que se analiza.

[13] Revisar la descripción de **multiplicidad** en el apartado "6.1.4.2 Relaciones" de la página 28.

6.1.5.1.5. Diagrama de Despliegue

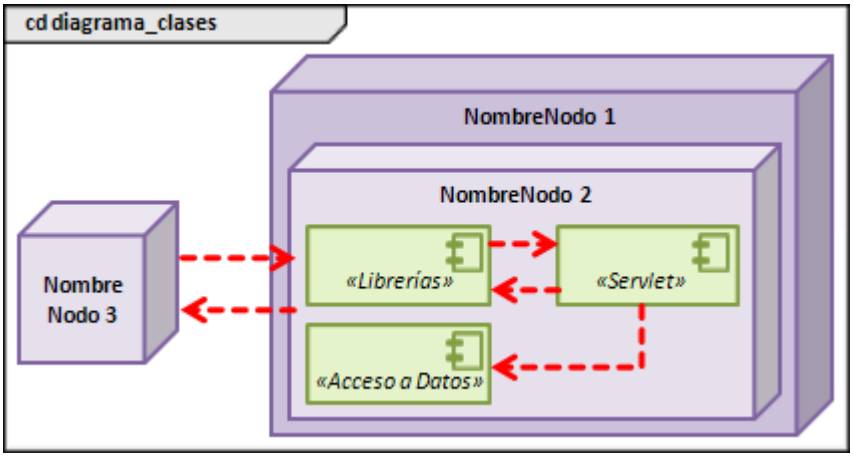


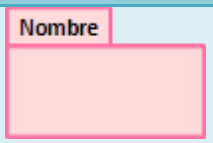
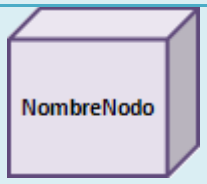
Figura 10: Diagrama de Despliegue

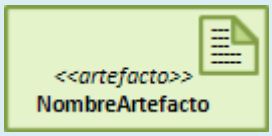
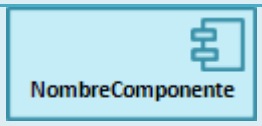
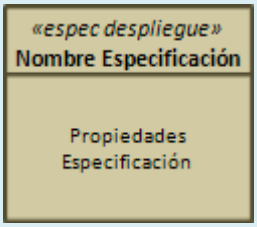
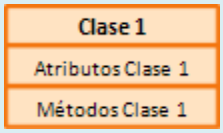
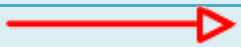
Ayudan a visualizar la configuración física del sistema en tiempo de ejecución, a través de nodos que muestran la disposición tanto en hardware como en software de la aplicación (ver Figura 10).

Estos diagramas se relacionan directamente con los Diagramas de Componentes, pues un nodo agrupa uno o más componentes. Por esta razón, antes de realizar un Diagrama de Despliegue, el analista deberá previamente haber realizado el Diagrama de Componentes respectivo.

Un Diagrama de Despliegue está conformado por los elementos mostrados en la Tabla 8:

Tabla 8: Elementos de un Diagrama de Despliegue

Artefacto	Descripción
	**Paquete.-** Ver descripción en la “Tabla 9: Elementos de un Diagrama de Paquetes” de la página 44.
	**Nodo.-** Ver descripción en la “Tabla 1: Tipos de Clasificador” de la página 26.
	Artefacto.- Ver descripción en la “Tabla 5: Elementos de un Diagrama de Componentes” de la página 36. Son utilizados para

	representar la versión compilada de un componente. Cuando se desea especificar que un artefacto se despliega dentro de un nodo, se lo puede hacer de cualquiera de las siguientes maneras: **1.** Dibujando el símbolo del artefacto dentro del nodo. **2.** Con una flecha de dependencia desde el artefacto hasta el nodo, con el estereotipo «deploy». **3.** Listando el nombre del artefacto dentro del nodo.
	**Componente.-** Ver descripción en la "Tabla 1: Tipos de Clasificador" de la página 26.
	**Especificación de Despliegue.-** Es un subtipo de artefacto que especifica un conjunto de propiedades que determinan los parámetros de ejecución de un artefacto que es desplegado en un nodo. Un artefacto que implementa las propiedades de una especificación de despliegue es un descriptor de despliegue. Se puede conectar a la línea de dependencia que tiene un artefacto o hacia un nodo. Se representa mediante un rectángulo con el estereotipo «deployment spec». Las propiedades se listan dentro del símbolo como atributos, usando la notación nombre: tipo.
	**Clase.-** Ver descripción en la "Tabla 1: Tipos de Clasificador" de la página 26. Se puede omitir la representación gráfica de una clase, nombrándola en el interior del componente o nodo correspondiente.
	**Interfaz.-** Ver descripción en la "Tabla 1: Tipos de Clasificador" de la página 26.
(1) (2)	**Conector/Conexión[14].-** Es una Asociación que representa una conexión física entre nodos (una conexión Ethernet por ejemplo). Se pueden usar también, para modelar conexiones indirectas como un link satelital entre procesadores distantes. En un Diagrama de Despliegue se puede utilizar: **1.** Conector/Conexión (Asociación Simple)[15] **2.** Agregación
	**Generalización**

[14] Los adornos que puede tener una Conexión se describen en el apartado "6.1.4.2 Relaciones" de la página 28.

[15] Revisar la descripción de Asociación Simple y Agregación en la "Tabla 2: Tipos de Asociación" de la página 30.

	**Dependencia.-** Significa que un nodo (y sus elementos) requieren los servicios de otro, es decir, depende de su proveedor. Puede representarse gráficamente, o bien, listando los nodos desplegados dentro del nodo. En un Diagrama de Despliegue, la dependencia sirve para modelar: **1.** El vínculo entre un nodo y un componente. **2.** La relación entre un artefacto y su especificación de despliegue (desde la especificación hasta el artefacto, dibujando a ambos dentro del nodo). **3.** La relación entre un artefacto y un componente. **4.** La relación entre artefactos, donde uno de ellos usa al segundo. **5.** La capacidad que tiene un nodo para desplegar un componente o artefacto (estereotipada «deploy»). **6.** Cómo se organizan los componentes software sobre el hardware del sistema (estereotipada «manifest»). Si un artefacto representa la versión compilada de un componente o de un paquete, en UML se dice que el artefacto manifiesta a ese componente, paquete o clase.

6.1.5.1.6. Diagrama de Paquetes

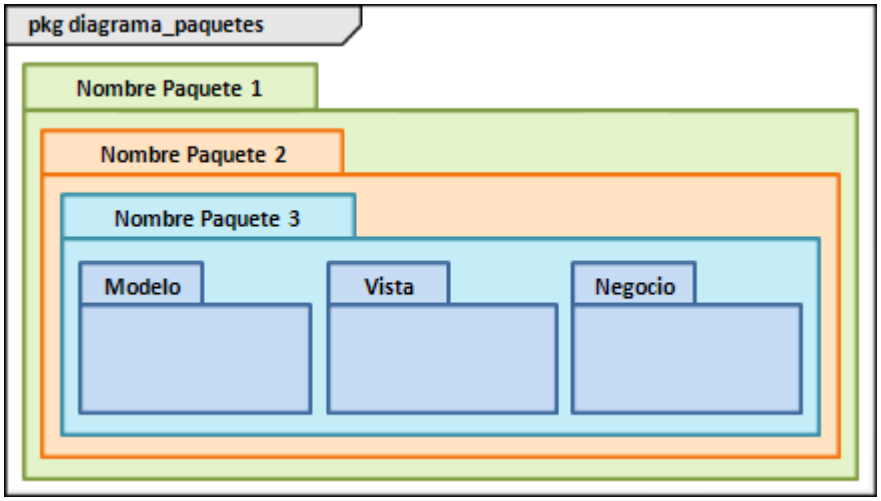


Figura 11: Diagrama de Paquetes

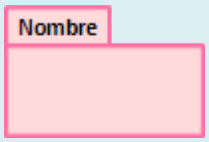
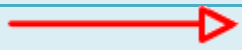
Es un tipo de diagrama que indica gráficamente la organización de los paquetes y de sus clases. Un Diagrama de Paquetes muestra cómo un sistema está dividido en agrupaciones lógicas y las dependencias entre esas agrupaciones.

Como se puede apreciar en la Figura 11, dentro de todo un conjunto de paquetes, se encuentran tres principales: modelo (agrupa las clases relacionadas con el dominio del sistema real), vista (contiene la interfaz gráfica de usuario) y negocio (contiene las reglas del negocio).

Al igual que otros artefactos, los paquetes pueden utilizar ciertos estereotipos, dentro de los que se destacan los siguientes: «facade», «framework», «stub», «subsystem» y «system».

En la Tabla 9 se proporciona la descripción de los componentes gráficos permitidos:

Tabla 9: Elementos de un Diagrama de Paquetes

Artefacto	Descripción
	**Paquete.-** Es un mecanismo de propósito general que permite modelar un sistema en grupos organizados de elementos, de tal manera que se los pueda entender más fácilmente. Suministra un namespace para los elementos agrupados. Un paquete puede poseer sus propios elementos (clases, interfaces, componentes, nodos, colaboraciones, casos de uso, diagramas e incluso otros paquetes), lo que quiere decir que el elemento es declarado en el paquete, y si éste es destruido, el elemento será destruido también. Generalmente el nombre del paquete deberá ir ubicado en la pestaña de la carpeta, mientras que su contenido, deberá ser ubicado dentro del artefacto.
	**Dependencia.-** Significa que al menos un elemento del paquete cliente utiliza los servicios ofrecidos por al menos un elemento del paquete proveedor. Se especializa en: **1.** Importación.- Cuando se importa un paquete, sólo son accesibles sus elementos públicos. Puede ser: 1.1. Pública «import». Los elementos importados tienen visibilidad pública en el paquete cliente. 1.2. Privada «access». Los elementos importados tienen visibilidad privada en el paquete cliente y no pueden usarse fuera de él, incluyendo cualquier otro paquete que los importe.
	**Generalización.-** Usada para especificar familias de paquetes.

Pese a que varios autores incluyen dentro de éste diagrama las clases y sus relaciones, hemos creído conveniente que únicamente los paquetes sean los nodos permitidos.

6.1.5.2. Diagramas de Comportamiento

Los **Diagramas de Comportamiento**, describen lo que debe suceder en el sistema modelado. Se usan para visualizar, especificar, construir y documentar los aspectos dinámicos de un sistema.

Se puede pensar en los aspectos dinámicos de un sistema como la representación de sus partes cambiantes en un determinado periodo de tiempo. Dentro de ésta categoría constan los siguientes diagramas:

6.1.5.2.1. Diagrama de Actividades

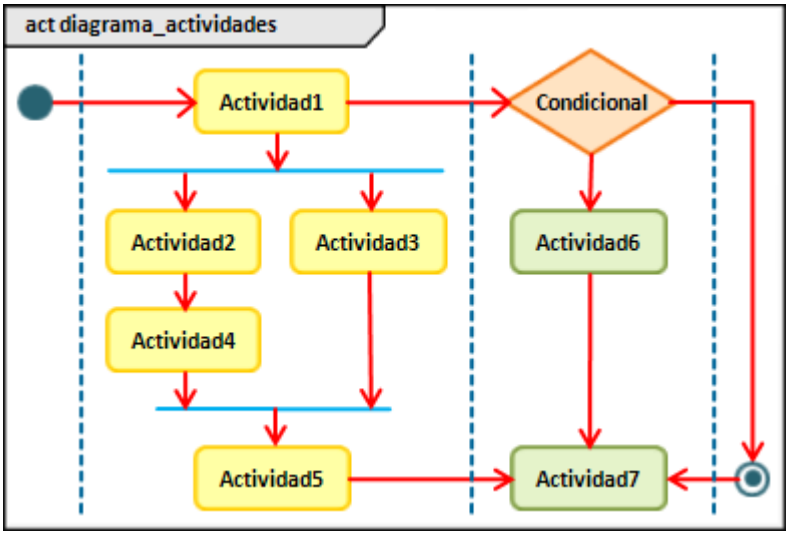
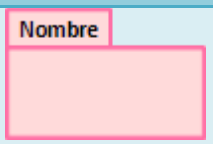
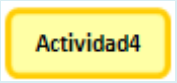
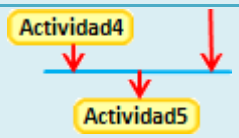
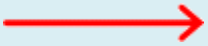


Figura 12: Diagrama de Actividades

Como se puede apreciar en la Figura 12, éste tipo de diagrama muestra el orden en el que se van a ejecutar las actividades dentro de un evento determinado. Permite modelar un sistema, un escenario o un proceso.

Los elementos utilizados son los que se muestran en la Tabla 10:

Tabla 10: Elementos de un Diagrama de Actividades

Artefacto	Descripción
	**Paquete.-** Ver descripción en la "Tabla 9: Elementos de un Diagrama de Paquetes" de la página 44.
	**Estado Inicial.-** Señala el estado inicial de la actividad que se va a representar.
	**Actividad.-** Representa la operación que se debe ejecutar. Pueden llegar a formar jerarquías, es decir, puede estar formada a su vez por varias actividades.
	**Condicional.-** Se da cuando existe la posibilidad de que ocurra más de una transición al terminar la ejecución de una actividad. Especifica caminos alternativos que pueden ser tomados basándose en una expresión booleana.
	**División.-** Representa una división de un flujo de control simple en dos o más flujos de control paralelos. Puede tener una transición de entrada y dos o más transiciones de salida, cada una de las que representan un flujo de control independiente.
	**Unión.-** Representa la fusión de dos o más flujos de control. Puede tener dos o más transiciones de entrada y una transición de salida.
	**Pasarela o Calle.-** Agrupa actividades de acuerdo a su responsabilidad, en regiones distintas. Permiten visualizar cómo cada grupo se separa de sus vecinos por una línea vertical sólida.
	**Transición.-** Indica el cambio de una actividad a otra. Especifica que cuando se completa una actividad, el flujo de control pasa inmediatamente a la siguiente.
	**Estado Terminal.-** Representa el final del diagrama de actividades.

6.1.5.2.2. Diagrama de Casos de Uso

Como se puede apreciar en la Figura 13, éste diagrama muestra un conjunto casos de uso, actores y sus relaciones. Ayuda a documentar el comportamiento que tiene el sistema actual (análisis) o el comportamiento que va a tener el sistema a desarrollar

(diseño), a través de la identificación de los actores y escenarios que se van a presentar.

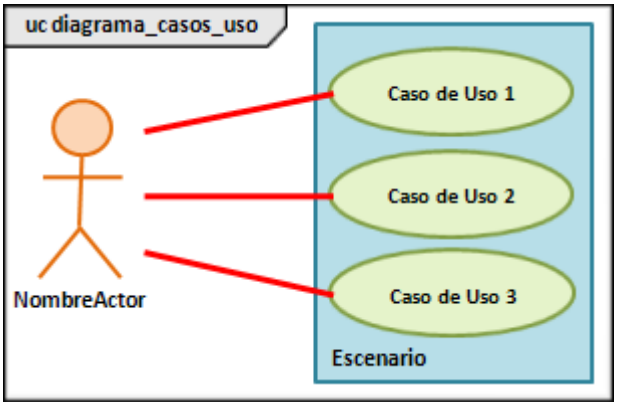
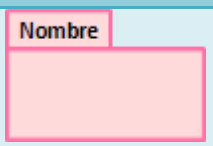
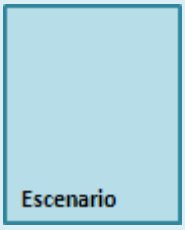
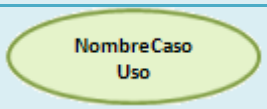
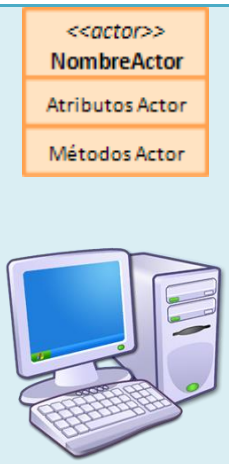
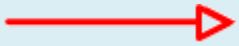


Figura 13: Diagrama de Casos de Uso

El Diagrama de Casos de Uso es especialmente importante para el modelado y la organización del comportamiento de un sistema. Los artefactos usados en el diseño de un Diagrama de Casos de Uso son:

Tabla 11: Elementos de un Diagrama de Casos de Uso

Artefacto	Descripción
	**Paquete.-** Ver descripción en la "Tabla 9: Elementos de un Diagrama de Paquetes" de la página 44.
	**Escenario.-** Es una secuencia específica de acciones de uno o varios casos de uso. Define claramente el ámbito del sistema, es decir, delimita hasta dónde va a llegar. Un escenario es básicamente una instancia de un caso de uso. El escenario de un caso de uso puede ser un sistema físico o cualquier otro elemento que pueda tener comportamiento como un componente, subsistema, o clase.
	**Caso de Uso.-** Ver descripción en la "Tabla 1: Tipos de Clasificador" de la página 26.
	**Actor.-** Representa cualquier entidad (humano, hardware o software) que puede interactuar con los casos de uso. Se puede representar mediante un muñeco (stick), como una clase con el estereotipo «actor» o con otras imágenes que expresen el tipo de actor que se quiere representar. El nombre del artefacto deberá

	identificar claramente el rol que el actor desempeña dentro del sistema. Puede ser de tres diferentes tipos: **1.** Primario o Principal.- Inician casos de uso. Actúa directamente sobre el sistema y tiene metas completamente definidas sobre él a través de los servicios. Éste maneja completamente el caso de uso. **2.** Secundario o De soporte.- Provee información externa que tiene relación con el sistema. Responden a una necesidad del sistema que el software no puede resolver. **3.** Externo o Silencioso.- Tiene cierto interés por el comportamiento del caso de uso.
	Asociación.- Es la manera cómo interactúa el actor con los casos de uso identificados en el sistema. Los actores pueden ser conectados a los casos de uso solamente por una Asociación.
	**Generalización.-** Establece una jerarquía de herencia, donde el elemento derivado adquiere toda la especificación del elemento base. Esta relación podrá darse exclusivamente entre actores o entre casos de uso.
	**Dependencia.-** Esta relación puede darse únicamente entre casos de uso de dos maneras diferentes: **1.** Inclusión.- Significa que el caso de uso base usa explícitamente el comportamiento de otro caso de uso en un momento específico, es decir, utiliza los pasos de un caso de uso como parte de la secuencia de su comportamiento. **2.** Extensión.- Se define como la agregación de pasos a la secuencia del caso de uso base. Significa que el caso de uso base implícitamente incorpora el comportamiento de otro caso de uso, únicamente en ciertos momentos llamados “puntos de extensión”. Esta relación sirve para modelar la parte de un caso de uso que el usuario puede ver como la conducta opcional o alterna del sistema.
	**Realización.-** Se puede especificar explícitamente la realización de un caso de uso por una colaboración. Sin embargo, un caso de uso dado es realizado por exactamente una colaboración, así que no se necesitará modelar esta relación explícitamente.

6.1.5.2.3. Diagrama de Estados

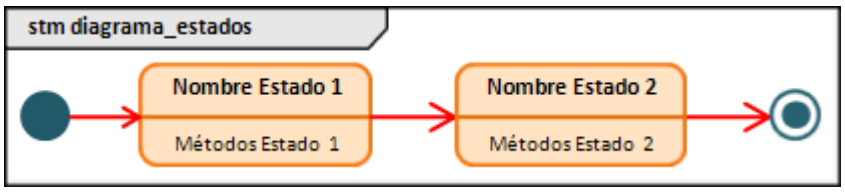
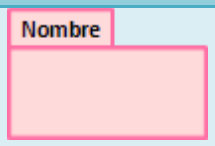
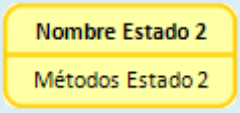
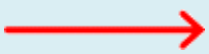


Figura 14: Diagrama de Estados

Capturan el ciclo de vida de uno o de más objetos y expresa los estados que va a tener el o los objetos durante su tiempo de ejecución. Estos diagramas (ver Figura 14) son especialmente importantes en el modelado del comportamiento de una interfaz, clase o colaboración.

Los elementos utilizados para elaborar un Diagrama de Estados, son los descritos dentro de la Tabla 12:

Tabla 12: Elementos de un Diagrama de Estados

Artefacto	Descripción
	**Paquete.-** Ver descripción en la "Tabla 9: Elementos de un Diagrama de Paquetes" de la página 44.
	**Estado Inicial.-** Indica en dónde inicia el diagrama, es decir, el comienzo del ciclo de vida del objeto representado.
	**Estado.-** Representa el estado del objeto en un momento determinado de su ejecución. Es una condición durante el ciclo vida de un objeto durante el que satisface alguna condición, realiza alguna actividad o espera por algún evento (estímulo que puede activar un estado de transición).
	**Actividades.-** Los tipos de actividades o eventos que pueden presentarse son: **1.** Inicio – Entrada (Enter).- Acciones cuando entra al estado. **2.** Hacer (Do).- Acciones mientras está en el estado. **3.** Salida (Exit).- Acciones cuando sale del estado.
	**Transición.-** Indica que un objeto en el primer estado realizará ciertas acciones y entrará en el segundo estado cuando un evento especificado ocurre y condiciones dadas son satisfechas. En

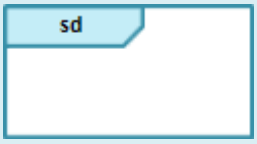
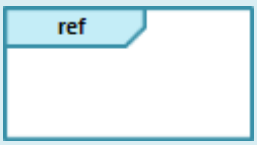
	pocas palabras, representa el cambio de un estado a otro.
	**Estado Final.-** Representa el final del Diagrama de Estados.

6.1.5.2.4. Diagramas de Interacción

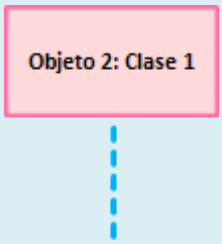
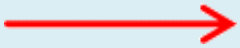
Los **Diagramas de Interacción** se encargan de mostrar el flujo de control y de datos entre los elementos del sistema modelado. Consisten en un conjunto de objetos y sus relaciones, incluyendo los mensajes que pueden enviarse entre ellos.

Un Diagrama de Interacción puede incluir también un conjunto de atributos locales con la misma sintaxis que la utilizada para una clase y pueden ser colocados en la parte superior del marco del diagrama o dentro del símbolo de nota en cualquier lugar del diagrama.

Tabla 13: Elementos comunes de un Diagrama de Interacción

Artefacto	Descripción
	**Interacción.-** Una interacción es una unidad de comportamiento que se enfoca en el intercambio de información con mensajes entre los elementos conectados de un clasificador. Se representa mediante un rectángulo sólido con una etiqueta en forma de pentágono en la esquina superior izquierda con la palabra clave **sd**, el nombre de la interacción y los parámetros. Una interacción puede ser dividida en varias partes, en cuyo caso adquiere el nombre de Fragmento de Interacción.
	**Uso de interacción.-** Hace referencia a una Interacción. Es una taquigrafía para copiar el contenido de la interacción a la que hace referencia el uso de interacción. Un Uso de interacción permite a varias interacciones hacer referencia a una interacción particular que representa una porción común de su especificación. Se muestra como un Fragmento de Interacción[16], donde el operador es llamado **ref**.
	Línea de Vida.- Representa un participante individual en una interacción. Su símbolo se divide en dos partes importantes:

[16] Ver la descripción de Fragmento de Interacción en la "Tabla 14: Elementos de un Diagrama de Secuencia" de la página 52.

	**1.** Objeto, simboliza a las instancias que intervienen en el caso de uso que se está representando. Gráficamente es similar al símbolo de clasificador, pues a menudo eso es lo que representa. Su información se puede desplegar en la forma "nombre_objeto: nombre_clase", donde el nombre del objeto no es obligatorio, mientras que el de la clase a partir del cual fue instanciado sí lo es. **2.** Línea de vida, simboliza el tiempo de vida del participante. Se muestra como una línea vertical entrecortada a continuación del objeto.
	**Mensaje.-** Indica la comunicación existente entre dos líneas de vida. Sobre éste artefacto se puede describir la acción que será realizada, los valores necesarios para ello, y los valores de retorno que se obtendrán al finalizar dicha tarea.

Aunque un Actor no es un artefacto propio de un Diagrama de Interacción, se lo suele incluir para aclarar un poco más las interacciones representadas.

6.1.5.2.4.1. Diagrama de Secuencia

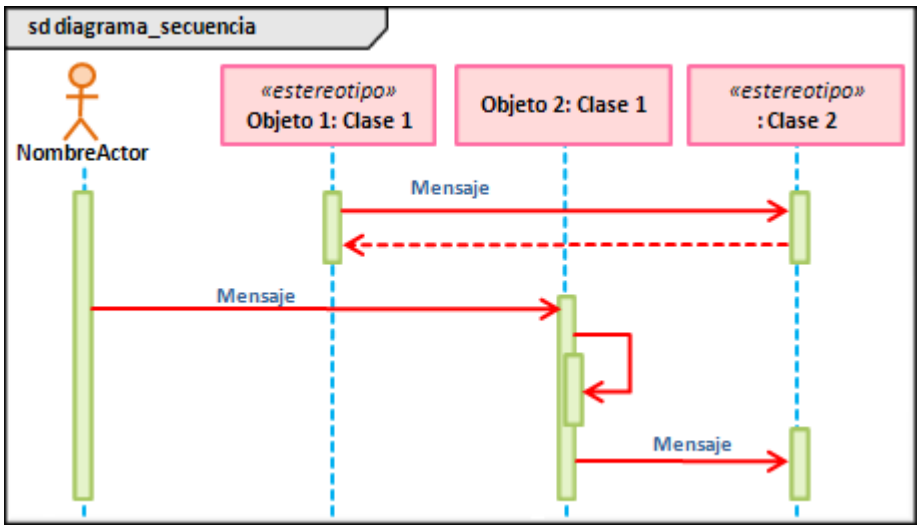


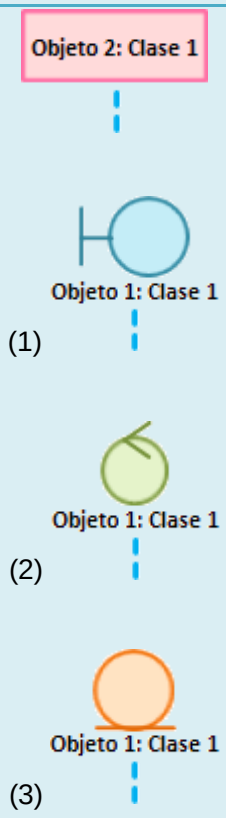
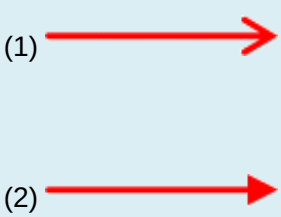
Figura 15: Diagrama de Secuencia

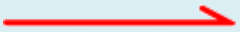
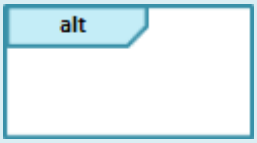
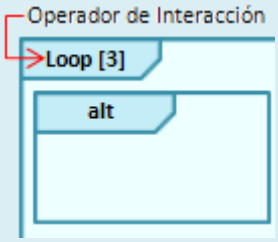
Como puede apreciarse en la Figura 15, este tipo de diagrama representa la secuencia ordenada y lógica que se da entre los objetos de un caso de uso dentro de un escenario específico. Se enfoca en ordenar los mensajes de acuerdo a su tiempo de envío.

Los Diagramas de Secuencia constituyen el mayor trabajo que debe realizarse durante la etapa de diseño, puesto que marcan el comportamiento que tendrá el sistema en tiempo de ejecución. Para lograrlo, se deberá incluir con gran detalle, cómo los objetos van a realizar todos y cada uno de los procesos.

Los elementos identificados en éste diagrama UML son los que se presentan en la Tabla 14:

Tabla 14: Elementos de un Diagrama de Secuencia

Artefacto	Descripción
	**Actor.-** Ver descripción en la "Tabla 11: Elementos de un Diagrama de Casos de Uso" de la página 47. Aunque no es un artefacto propio de un Diagrama de Secuencia, se lo suele incluir.
	**Línea de vida.-** Ver descripción en la "Tabla 13: Elementos comunes de un Diagrama de Interacción" de la página 50. De manera opcional el analista puede incluir tres artefactos específicos para mostrar: **1.** Objetos de frontera «boundary».- Son aquellos con los que puede interactuar el actor a través de la interfaz de usuario. **2.** Objetos de entidad «entity».- Generalmente son objetos del modelo de dominio. **3.** Objetos de control «control».- Son intermediarios entre los de frontera y entidad. En un Diagrama de Secuencia todos los objetos deben estar alineados en la cima del mismo, con sus líneas de vida dibujadas hasta el fondo. Las líneas de vida de objetos creados durante la interacción empiezan cuando se recepta el mensaje «create», mientras que la línea de vida de objetos destruidos durante ella, acaban cuando se recepta el mensaje «destroy» (se da la señal visual de una X grande para marcar el fin de sus vidas).
	**Mensaje.-** Ver descripción en la "Tabla 13: Elementos comunes de un Diagrama de Interacción" de la página 50. Pueden ser colocados entre las líneas de vida de los objetos o entre métodos. Un mensaje puede ser de tres tipos: **1.** Simple.- Transferencia del control de un objeto a otro.

(3) 	**2.** Síncrono.- El objeto que lo envía esperará la respuesta a tal mensaje antes de continuar con su trabajo. **3.** Asíncrono.- El objeto que lo envía no espera una respuesta antes de continuar.
	**Método/Activación.-** Muestra el periodo de tiempo durante el cual un objeto ejecuta una acción directa o a través de un procedimiento subordinado. Se encuentra unido al objeto a través de una línea de vida. El inicio del artefacto se alinea con el inicio de la acción, mientras que el final se alinea con su realización. Al igual que otros artefactos, permite anidación (causada por recursión).
	**Fragmento combinado.-** Describe una unidad de interacción reutilizable. Es definido por un operador de interacción y sus operandos de interacción. Ayudan a graficar el diagrama en una manera compacta y concisa.
	**Operador de interacción.-** Es una enumeración que designa diferentes tipos de operadores para un fragmento combinado. Los valores literales de esta enumeración son: **1.** Alternativa **alt**.- El fragmento combinado representa una elección de comportamiento, en donde únicamente uno de los operandos contenidos será escogido. El operando debe tener una expresión que evalúe el valor de verdad en ese punto de interacción (si no tiene ningún control el valor se considera verdadero). Puede incluir un operando **else**. **2.** Opcional **opt**.- El fragmento combinado representa una elección de comportamiento que puede darse o no. Contiene un único operando. **3.** Quiebre **break**.- El fragmento combinado representa un quiebre de secuencia. Un operador break con una expresión de control es escogido si la expresión es verdadera (el resto del fragmento se ignora), de lo contrario el operando es ignorado (el resto del fragmento se ejecuta). Este operador debe cubrir todas las líneas de vida del fragmento combinado y debe tener un único operando. **4.** Paralelo **par**.- El fragmento combinado Incluye hilos de ejecución paralelos. **5.** Bucle **loop**.- El fragmento combinado representa un bucle que

	será repetido un cierto número de veces. La expresión de control puede incluir un número mínimo y uno máximo de repeticiones para el bucle (como una expresión booleana). El bucle iterará el número mínimo de veces (dado por la expresión booleana) y a lo mucho el número máximo de veces. Luego de que el número mínimo se ha ejecutado y la expresión booleana es falsa el bucle termina. Su notación es de la forma "loop['('<miniN> [',' <maxN>]')'], en donde <minN> es un número natural no negativo y <maxN> es un número natural no negativo mayor o igual a <minN>. Si únicamente se da <minN> significa que es igual a <maxN>. Si únicamente se da **loop** significa que el bucle tendrá infinito como máximo y 0 como mínimo. Este operador debe tener un único operando. **6.** Región crítica **critical**.- Representa una región crítica que será tratada atómicamente por el fragmento combinado cuando determina secuencias válidas. No pueden intercalarse otros eventos mientras se ejecuta. **7.** **neg**.- El fragmento combinado representa secuencias inválidas. Todas las interacciones diferentes de **neg** son considerados positivos, es decir, describen secuencias válidas que deberían ser posibles. Este operador debe tener un único operando. **8.** Aseveración **assert**.- El fragmento combinado representa una aseveración. Las secuencias del operando **assert** son las únicas continuaciones válidas. **9.** **strict**.- El fragmento combinado representa una secuencia estricta del comportamiento de los operandos. Su semántica define el orden estricto de los operandos en primer nivel dentro del fragmento combinado. Por lo tanto los eventos dentro del fragmento combinado no serán comparados directamente con otros eventos. **10.** **seq**.- El fragmento combinado representa una secuencia débil del comportamiento de los operandos.
	Operando de interacción.- Define el tipo de operador de un fragmento combinado.

6.1.5.2.4.2. Diagrama de Comunicación

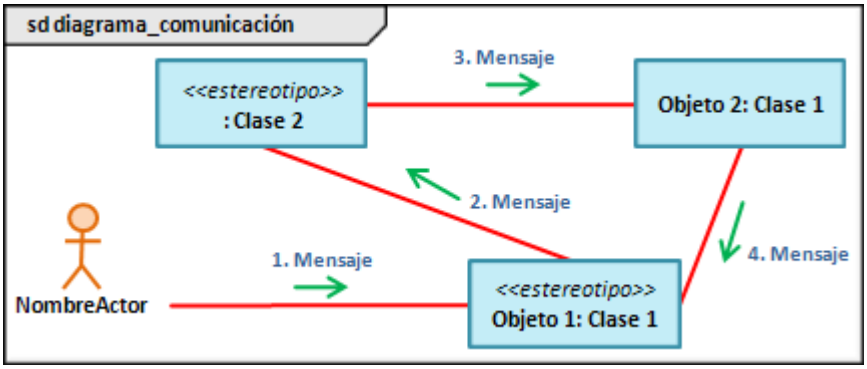


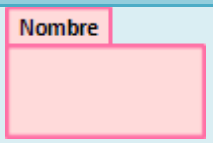
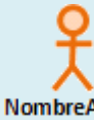
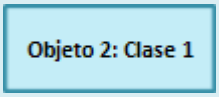
Figura 16: Diagrama de Comunicación

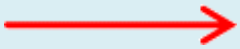
Antiguamente conocido como Diagrama de Colaboración, es el encargado de explicar la distribución que los objetos van a tener en el sistema (ver Figura 16).

Guardan una estrecha relación con los diagramas de secuencia, puesto que los dos muestran cómo interactúan los objetos entre sí, a través del intercambio de mensajes. Sin embargo, la diferencia radica en que el diagrama de secuencia hace énfasis en seguir ordenadamente la narrativa del flujo de un caso de uso en el tiempo, mientras que un diagrama de comunicación hace énfasis en la organización estructural de los objetos, mostrando cómo colaboran en un caso de uso.

Un diagrama de comunicación está conformado por los elementos de la Tabla 15:

Tabla 15: Elementos de un Diagrama de Comunicación

Artefacto	Descripción
	**Paquete.-** Ver descripción en la "Tabla 9: Elementos de un Diagrama de Paquetes" de la página 44.
	**Actor.-** Ver descripción en la "Tabla 11: Elementos de un Diagrama de Casos de Uso" de la página 47. Aunque no es un artefacto propio de un Diagrama de Comunicación, se lo suele incluir.
	**Objeto/Línea de Vida.-** Ver descripción en la "Tabla 13: Elementos comunes de un Diagrama de Interacción" de la página 50. En un Diagrama de Comunicación las líneas de vida se

	representan únicamente mediante el objeto.
	**Mensaje.-** Ver descripción en la “Tabla 13: Elementos comunes de un Diagrama de Interacción” de la página 50. Puede ir acompañado de un número que indica el orden del mensaje dentro de la secuencia.

6.1.5.2.4.3. Diagrama de Tiempos

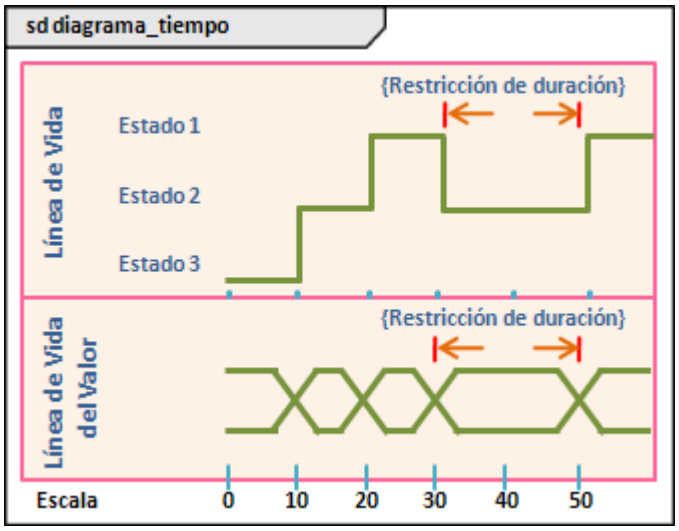
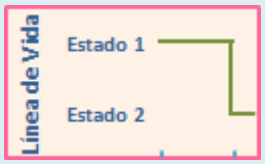


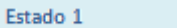
Figura 17: Diagrama de Tiempos

El diagrama mostrado en la Figura 17, se constituye en un nuevo tipo incorporado en la versión 2.0 de UML. Se usan para mostrar el cambio en el estado o valor de uno o más clasificadores en el tiempo, en respuesta a eventos ocurridos. También puede mostrar la interacción entre los eventos de tiempos, las restricciones de tiempos y la duración que los gobiernan. El uso más común es mostrar el cambio de estado de un objeto a lo largo del tiempo, en respuesta a los eventos o estímulos aceptados.

Los artefactos utilizados en éste diagrama son detallados en la Tabla 16.

Tabla 16: Elementos de un Diagrama de Tiempos

Artefacto	Descripción
	**Línea de vida.-** Representa un participante individual en una interacción. Muestra el cambio de estado de un participante en el tiempo. El eje X muestra el tiempo trascurrido en cualquier unidad que se elija, mientras que el eje Y se nombra con una lista de

	estados que la entidad puede tener.
	**Línea de vida del valor.-** Muestra el cambio del valor de un objeto en el tiempo. El eje X muestra el tiempo transcurrido en cualquier unidad que se elija. El valor se muestra entre el par de líneas horizontales que se cruzan en cada cambio del valor. El valor se denota explícitamente como texto. Cruzada refleja el evento donde el valor cambió.
	**Estado.-** Es el estado del clasificador o atributo dado como un valor enumerable discreto o continuo. Se representa como una cadena de texto.

6.1.5.2.4.4. Diagrama de Vista de Interacción

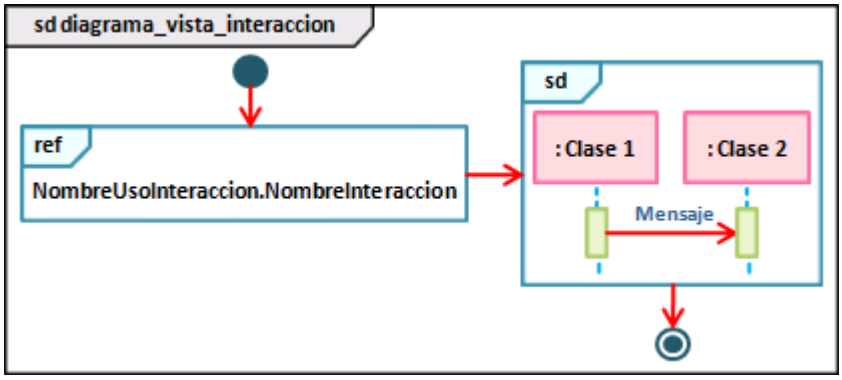


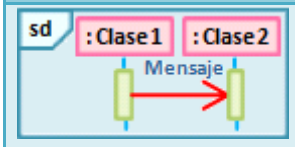
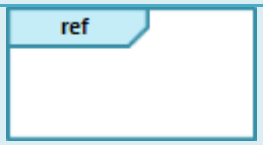
Figura 18: Diagrama de Vista de Interacción

Como puede apreciarse en la Figura 18, este tipo de diagrama muestra cierta vista de los aspectos dinámicos del sistema que se debe modelar. Son una especialización de los Diagramas de Actividad centrados en el flujo de control, donde los nodos son interacciones o usos de interacciones.

Cuando se tenga entre manos sistemas complejos, lo más aconsejable es subdividir en partes más pequeñas las interacciones, de tal manera que se pueda realizar un mejor trabajo en el diseño del sistema.

Los artefactos utilizados para elaborar un Diagrama de Vista de Interacción, se especifican en la Tabla 17 mostrada a continuación:

Tabla 17: Elementos de un Diagrama de Vista de Interacción

Artefacto	Descripción
	**Interacción.-** Ver descripción en la “Tabla 13: Elementos comunes de un Diagrama de Interacción” de la página 50.
	**Uso de Interacción.-** Ver descripción en la “Tabla 13: Elementos comunes de un Diagrama de Interacción” de la página 50.

LENGUAJE DE PROGRAMACIÓN C SHARP

6.2.1. Definiciones

C# o C Sharp es un lenguaje de programación que fue diseñado específicamente para utilizarse en la plataforma .NET, desarrollado por Microsoft (principalmente por Scott Wiltamuth y Anders Hejlsberg), tomando como base las mejores características de C, C++, Delphi, Java y Visual Basic, con la idea no sólo de crear un lenguaje orientado a objetos, sino también, con el objetivo de contar con el primer lenguaje orientado a componentes.

6.2.2. Características

- No soporta herencia múltiple: únicamente se maneja el runtime para la herencia múltiple en la forma de interfaces.
- No maneja apuntadores: usa los llamados delegados (delegates) para proveer del modelo de eventos.
- Gestión automática de memoria, puesto que proporciona un colector de basura para la administración de memoria en los programas C#.
- Es completamente orientado a objetos (encapsulamiento, herencia y polimorfismo).
- A los modificadores private, public y protected, se incrementa el internal.
- Es un lenguaje orientado a componentes.
- Posee un sistema de tipos unificado, puesto que todos los tipos de datos se derivan de una clase base común llamada System.Object.
- Se puede insertar en código fuente C#, fragmentos de código escrito en cualquiera de los lenguajes soportados.

6.2.3. Tipos de Datos

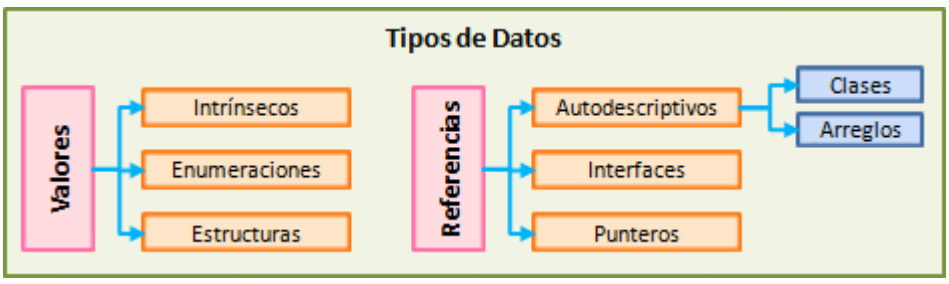


Figura 19: Tipos de Datos[17]

[17] CHARTE OJEDA, Francisco. 2003. Programación C# .NET. España. ANAYA. 104 p.

C# cuenta con un sistema de tipos unificado que establece que todos los tipos de datos parten de una raíz común denominada System.Object (ver Figura 20), misma que se subdivide en dos grandes grupos (ver Figura 19): por una parte los que almacenan valores (estáticos), y por otro, los tipos que almacenan referencias (dinámicos).

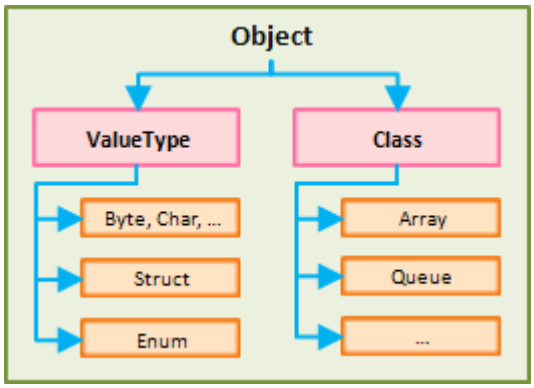


Figura 20: Sistema de tipos unificado[18]

6.2.3.1. Tipos de Datos Nativos en C#

Como Tipos Intrínsecos se conoce a los tipos de datos fundamentales para cualquier lenguaje: números, cadenas de caracteres, lógicos, etc.

El Sistema Común de Tipos (CTS) define como tipos de datos para C#, a los que se muestran a continuación, en la Tabla 18:

Tabla 18: Tipos de Datos Intrínsecos[19]

Tipo	Dato que puede contener
byte	Número entero de 8 bits sin signo.
sbyte	Número entero de 8 bits con signo.
short	Número entero de 16 bits sin signo.
ushort	Número entero de 16 bits con signo.
int	Número entero de 32 bits sin signo.
uint	Número entero de 32 bits con signo.
long	Número entero de 64 bits sin signo.
ulong	Número entero de 64 bits con signo.

[18] CHARTE OJEDA, Francisco. 2003. Programación C# .NET. España. ANAYA. 105 p.
[19] CHARTE OJEDA, Francisco. 2003. Programación C# .NET. España. ANAYA. 107 – 108 pp.

float	Número con parte decimal de precisión simple.
double	Número con parte decimal de precisión doble.
decimal	Número con parte decimal fija para cálculos financieros.
bool	Valor de verdad verdadero (true) o falso (false).
char	Un carácter del conjunto Unicode.
string	Cadena de caracteres.
object	Referencia a cualquier objeto.

6.2.4. Declaración de Variables

En el lenguaje de programación C# se define a una variable como un identificador utilizado para almacenar datos generados durante el proceso de ejecución de un programa.

La declaración de una variable (en cuanto a su nombre) debe seguir ciertas normas establecidas:

- Su nombre debe ser claro y referente al problema.
- No deben contener espacios en blanco ni símbolos extraños entre ellas.
- No deben ser palabras de C#.

En lo referente a su visibilidad se deberá tomar en consideración los siguientes modificadores:

- Public, si se desea que la variable sea visible en todo el namespace.
- Private, en cuyo caso la variable será visible únicamente en la clase dentro de la que fuera declarada.
- Protected, cuando la variable deba ser accedida desde una clase determinada, y desde todas las clases derivadas a partir de ésta.
- Internal, en cuyo caso la variable podrá ser accedida po. el código que es parte del mismo componente (ensamblado), ya sea una aplicación o una biblioteca.
- Internal protected, que permite el acceso ya sea de forma internal o protected.

Ahora bien, la sintaxis para la declaración de una variable es la mostrada en la Figura 21:

Sintaxis

```
<modificador_acceso> <tipo_dato> <nombre_variable> [= <inicialización>]
```

Figura 21: Sintaxis para la declaración de una variable

6.2.5. Programación Orientada a Objetos en C#

6.2.5.1. Ámbitos con nombre

Un espacio o ámbito con nombre (conocido como namespace) "es un ámbito delimitado explícitamente al que se ha asignado un identificador"[20]. Dentro de éstos, es posible incluir definiciones de tipos de datos como también nuevos ámbitos con nombre, dando paso a una jerarquía de namespaces.

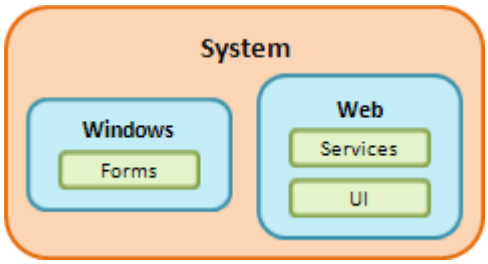


Figura 22: Anidamiento de diferentes Ámbitos con nombre[21]

Como se puede observar en la Figura 22, el ámbito SYSTEM contiene en su interior a dos namespaces: WINDOWS y WEB. Por su parte el ámbito WEB contiene a su vez, a los ámbitos SERVICES y UI. De ésta manera se puede comprobar es posible que se presente un anidamiento con todos los ámbitos que formen parte de un proyecto de software determinado.

La sintaxis de un namespace o ámbito con nombre es la mostrada en la Figura 23:

Sintaxis

```
namespace <nombre_ambito> {
        //líneas de código
}
```

Figura 23: Sintaxis para la declaración de un Ámbito con Nombre (Namespace)

[20] CHARTE OJEDA, Francisco. 2003. Programación C# .NET. España. ANAYA. 174 p.
[21] CHARTE OJEDA, Francisco. 2003. Programación C# .NET. España. ANAYA. 174 p.

6.2.5.2. Clases

Una clase es una plantilla en la que se describen las características y el comportamiento de todos los objetos que se creen a partir de ésta. Las clases son utilizadas para modelar las aplicaciones como si estuviesen conformadas por un conjunto de objetos.

En C# las clases representan un novedoso mecanismo para crear nuevos tipos de datos, a partir de las que se crearían los objetos para acceder a sus miembros. Lo que las diferencia de cualquier otro tipo de dato, es que las clases pueden derivarse unas de otras.

La palabra clave que las representa es **class**, misma que deberá ser precedida por un identificador; las instrucciones que la clase incluya serán delimitadas por llaves ({ }) de inicio y de cierre. Adicionalmente, se deberá tomar en consideración que cualquier clase deberá ser declarada dentro de un ámbito con nombre (namespace).

En lo referente a su visibilidad tenemos los siguientes modificadores:

- Public, si se desea que la clase sea visible en todos los ámbitos.
- Private, que es el ámbito más reducido de los existentes. En éste caso, la clase podrá ser utilizada únicamente en el interior del ámbito en donde haya sido declarada.
- Internal, en cuyo caso la clase será visible desde fuera de su ámbito inmediato, siempre y cuando pertenezca al mismo ensamblado.
- Abstract, en cuyo caso una clase no podrá ser instanciada, es decir, no se podrán crear objetos a partir de ella.
- Sealed, si se desea que la clase no pueda ser heredada.

Es importante indicar que las clases cuentan con constructores y destructores propios, los cuales son métodos específicos para la construcción y destrucción de los objetos instanciados a partir de una clase determinada.

Una misma clase puede contar con varios constructores siempre y cuando posean diferentes listas de parámetros, pero destructores, uno sólo. Finalmente en la Figura 24 se hace la descripción de la sintaxis utilizada para la declaración de una clase:

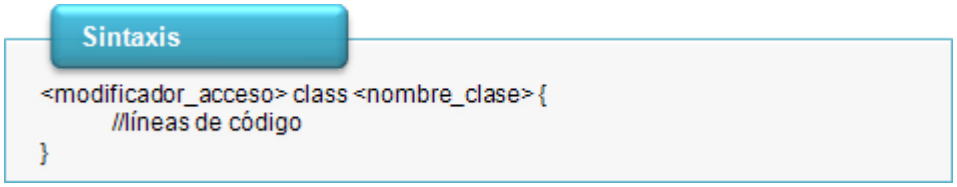


Figura 24: Sintaxis para la declaración de una Clase

6.2.5.3. Miembros de una Clase

Los miembros que forman parte de una clase son los mostrados en la Tabla 19:

Tabla 19: Miembros de una Clase

Miembro	Descripción
Variables	Identificador utilizado para almacenar datos generados durante la ejecución de un programa.
Métodos	Describen el comportamiento y la funcionalidad que va a tener una clase u objeto. Son parte del tipo (clase), pero no son parte del objeto.
Propiedades	Permiten personalizar a los objetos o bien, obtener información a partir de ellos.
Eventos	Son señales que permiten a los objetos interactuar con la aplicación en la que son utilizados. Una clase puede usar un evento para notificar a otra que algo ocurrió.
Miembros compartidos (variables o métodos)	Están asociados con la clase, no con el objeto. Son compartidos por todos los objetos de la clase y se diferencian por contar con el modificador de visibilidad static. Son conocidos como miembros estáticos.
Miembros sobrecargados (métodos)	Permite que en una misma clase existan varios métodos con el mismo nombre, siempre y cuando la lista de parámetros varíe.
Miembros redefinidos	Facilitan una nueva implementación del miembro.

Independientemente del miembro de clase al que se desee hacer referencia, éstos pueden utilizar los modificadores de visibilidad private, protected, public, internal y protected internal (ver sección 6.2.4, Declaración de Variables).

6.2.5.4. Interfaz

Una interfaz es una clase en la que se enumeran los métodos que la componen, indicando tipos de retornos y listas de parámetros, y cuyos miembros no podrán ser implementados. Se la puede considerar como una especificación a la que podrán ajustarse las clases que así lo requieran.

La sintaxis utilizada es la que se muestra a continuación en la Figura 25:

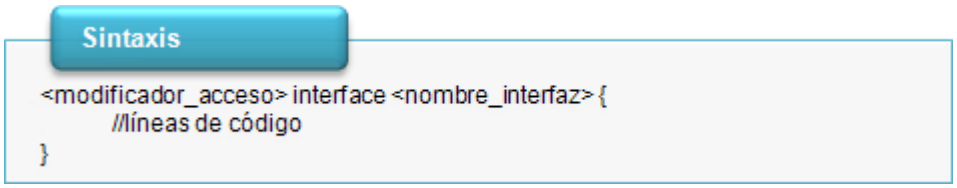


Figura 25: Sintaxis para la declaración de una Interfaz

6.2.6. Colecciones de Datos

El lenguaje de Programación C# proporciona a los desarrolladores de software una gama variada de listas especializadas o colecciones de datos, las mismas serán utilizadas de acuerdo al tipo de información con el que se esté trabajando.

En la Tabla 20 mostramos un listado de las Colecciones de Datos más importantes:

Tabla 20: Colecciones de Datos

Colección	Descripción
ArrayList	Almacena una lista de objetos.
SortedList	Almacena una lista de objetos, pero de forma ordenada.
Queue	Almacena una lista de objetos, pero con la restricción de que el primer objeto en entrar es el primero en salir (Tratamiento similar al de una cola).
Stack	Similar a la anterior, con la diferencia de que el último objeto en entrar será el primero en salir.
HashSet	Almacena una lista de objetos, con la restricción de que no se permiten objetos duplicados.
SortedList	Almacena una colección de pares de valores, una clave y un valor, de manera ordenada.

SortedDictionary	Almacena una colección de pares (clave, valor) que se ordenan en la clave.
ArraySegment	Delimita una sección de una matriz unidimensional.
LinkedList	Representa una lista doblemente vinculada.

6.2.6.1. Arreglos Unidimensionales

Como su nombre lo indica, se trata de arreglos de una sola dimensión, en donde el número de elementos que contendrá será dado al momento de su creación. Es por ello que pueden ser representadas gráficamente como una lista de valores.

Para declarar un arreglo se deberá especificar el ámbito, identificador, tipo, símbolo ([ ]) y nombre, tal y como se muestra en la Figura 26:

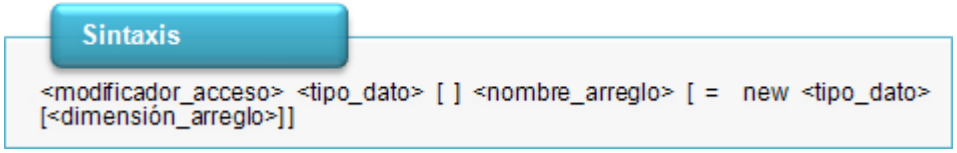


Figura 26: Sintaxis para la declaración de un Arreglo Unidimensional

6.2.6.2. Arreglos Multidimensionales

Son arreglos que tienen varias dimensiones (60 como máximo), cada una de las cuales será manejada con un rango diferente. Cada uno de éstos rangos podrá contar con tantos elementos como valores posibles hay en el tipo long. En la Figura 27 se muestra la sintaxis utilizada para la declaración de un arreglo multidimensional:

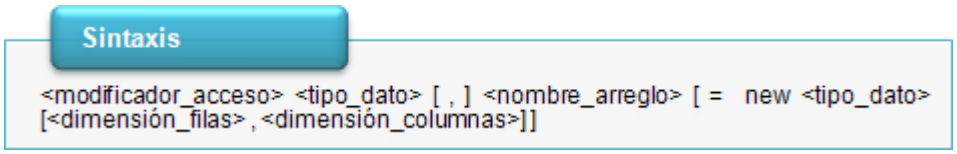


Figura 27: Sintaxis para la declaración de un Arreglo Multidimensional

En éste caso se trata de un arreglo bidimensional, pero de acuerdo a las necesidades del programador, el número de dimensiones variará. Como se puede observar, el número total de elementos es 10 (5 elementos en la primera dimensión, por 2 elementos de la segunda dimensión).

6.2.6.3. Colecciones Genéricas

Las colecciones genéricas nos permiten definir el tipo de datos que se desea almacenar, cuando éste no sea un objeto.

En la Tabla 21 se muestra un listado de las mismas:

Tabla 21: Colecciones Genéricas

Colección	Colección Genérica
ArrayList	List.(Representa una lista de objetos con establecimiento inflexible de tipos a la que se puede tener acceso a través de un índice. Proporciona métodos para buscar, ordenar y manipular listas)
SortedList	SortedList
Queue	Queue
Stack	Stack
HashSet	HashSet
SortedDictionary	Dictionary (Representa una colección de claves y valores)
ArraySegment	ArraySegment
LinkedList	LinkedList

HERRAMIENTAS CASE

6.3.1. Definiciones

Las herramientas CASE (Ingeniería de Software Asistida por Computador, por sus siglas en inglés: Computer Aided Software Engineering), se definen como un conjunto de programas, métodos, utilidades y técnicas que apoyan a los analistas, ingenieros en sistemas y desarrolladores en general, durante todo el proceso de desarrollo de software.

Estas herramientas nos permiten automatizar dicho proceso, completa o parcialmente, para así aumentar la productividad de las aplicaciones y reducir su coste de desarrollo. Es probable que las dos utilidades más importantes que presta una Herramienta CASE a un desarrollador de software sean la Ingeniería Directa y la Ingeniería Inversa.

6.3.1.1. Ingeniería Directa

Es el proceso de trasformar un modelo en código fuente mapeándolo en un lenguaje de programación. Dicho proceso genera cierta pérdida de información ya que los modelos escritos en UML son semánticamente más ricos que cualquier lenguaje de programación orientado a objetos. Esta es precisamente la razón por la que se necesitan modelos además de código: los rasgos estructurales, como colaboraciones y características de comportamiento, pueden ser visualizadas claramente en UML, pero no tan claramente en el código fuente.

6.3.1.2. Ingeniería Inversa

Es el proceso de transformar código fuente en un modelo mapeándolo desde un lenguaje de programación específico. La ingeniería inversa produce un gran cúmulo de información que podría estar en un nivel más bajo de detalle, del que se necesita para construir modelos útiles. No se podrá recrear completamente un modelo desde el código a menos que las herramientas usadas codifiquen la información de los comentarios, los cuales van más allá de la semántica del lenguaje de programación.

6.3.2. Objetivos

Entre los principales objetivos que persigue una Herramienta CASE se pueden anotar:

- Optimizar la productividad, efectividad y eficiencia en el desarrollo de software.

- Reducir el coste asociado con el desarrollo de software, a través de la disminución en la inversión de recursos, principalmente tiempo y dinero.
- Mejorar la planificación de recursos que se deberán invertir en un proyecto.
- Automatizar el proceso de desarrollo y de mantenimiento del software.
- Ayudar en la reutilización, portabilidad y documentación de proyecto de software.
- Gestión global en todas las fases de desarrollo de software con una misma herramienta.

6.3.3. Componentes

6.3.3.1. Repositorio (Directorio)

Almacena los elementos definidos o creados por la herramienta CASE, y su gestión se realiza gracias a un Sistema de Gestión de Base de Datos (SGBD) o a un sistema de gestión de ficheros.

6.3.3.2. Meta Modelo

Marco para la definición de técnicas y metodologías que serán soportadas por la herramienta.

6.3.3.3. Carga o Descarga de Datos

Permiten cargar el repertorio de la herramienta con los datos que provienen desde otros sistemas, o bien generar información que podrá ser utilizada en ellos.

6.3.3.4. Comprobación de Errores

Es una de las capacidades más importantes de las herramientas CASE, puesto que analiza la exactitud, integridad y consistencia de los esquemas generados por dicha herramienta, incrementando así la productividad y la calidad del producto. Existen cinco tipos básicos de comprobaciones que una herramienta CASE debe ejercer sobre los diferentes diagramas:

- Comprobación de sintaxis y tipo
- Comprobación de integridad y consistencia

- Comprobación de descomposición funcional
- Comprobación cruzada de consistencia a través de todos los niveles y vistas
- Comprobación de rastreo de requerimientos

6.3.3.5. Módulo de Diagramación y Modelación

Está constituido por editores de texto y herramientas de diseño gráfico que facilitan el trabajo realizado por los analistas o desarrolladores en general. Permite a los usuarios elaborar los diferentes diagramas para el modelado de sistemas con la aplicación de un modelo determinado, ya sea de manera automática o semiautomática.

Gracias a este módulo, cuando se realiza cualquier cambio en el sistema se modifican los diagramas, más no las aplicaciones: el código se genera nuevamente, y la documentación e información estará siempre actualizada. Más allá de la automatización del proceso de creación de diagramas debemos tomar en consideración tres aspectos fundamentales:

- Capturando el significado de un objeto dibujado en la pantalla del ordenador, podemos comprobar su corrección y complejidad.
- Almacenando los objetos de un diagrama determinado, podemos compartir objetos entre diferentes diagramas.
- Recogiendo el significado de una eficaz corrección de errores realizada de manera automática sobre los diagramas dibujados, podemos generar código automáticamente.

6.3.3.6. Generador de Código

Es el encargado de transformar el diseño realizado, en código fuente de la aplicación que se va a desarrollar.

6.3.3.7. Generador de Documentación

Se alimenta del repositorio de la herramienta para transcribir las especificaciones contenidas en él.

6.3.4. Clasificación

No existe una clasificación estándar de herramientas CASE, por lo que varios autores utilizan múltiples criterios para hacerlo. De acuerdo a su funcionalidad estas herramientas pueden ser clasificadas tal como se muestra en la Tabla 22:

Tabla 22: Clasificación de Herramientas CASE según su funcionalidad

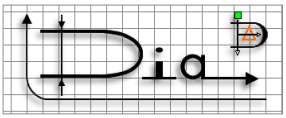
Nombre	Descripción
Herramientas integradas I – CASE	Integrated CASE o CASE integrado, abarcan todo el ciclo de vida del desarrollo de software. También son conocidas como CASE workbench.
Herramientas de alto nivel U – CASE	Upper CASE o CASE superior, son herramientas orientadas a automatizar y dar soporte a las actividades que se desarrollan durante el análisis y el diseño.
Herramientas de bajo nivel L – CASE	Lower CASE o CASE inferior, son herramientas que están orientadas a automatizar las fases de construcción e implantación.

6.3.5. Herramientas CASE para el diseño de la arquitectura de un producto software

Según el tipo de licencia que posean, estas herramientas pueden dividirse en dos grandes grupos: por una parte las privativas o de uso comercial, y por otra parte, las no privativas o de uso libre. Una breve lista de estas herramientas se muestra en la Tabla 23:

Tabla 23: Herramientas CASE para el diseño de la arquitectura de un producto software

Uso Libre	
Logotipo	**Descripción**
BOUML	**BOUML.-** Permite definir y generar código en C++, Java, IDL y PHP y es capaz de generar documentación en varios formatos, como por ejemplo HTML, XMI, etc. Su principal ventaja, es que a pesar de estar trabajando con varios diagramas a la vez, consume muy poca memoria. Es compatible con los sistemas operativos Windows / Unix / Linux / Solaris / MacOS X.

	**StarUML.-** Permite modelar diagramas UML y traducir ese modelo a código fuente C++, Java y C#. Se puede comenzar a dibujar los gráficos manualmente o seleccionar las plantillas que contiene el archivo de instalación para modificarlas. Funciona sobre los sistemas operativos Windows 98 SE/Me/2000/NT/XP.
	**Umbrello UML Modeller.-** Herramienta para el desarrollo de diagramas UML, que también permite la generación de código fuente para los lenguajes ActionScript, Ada, C++, Corba IDL, Java, JavaScript, PHP, Perl, Python, SQL y XMLSchema. Funciona sobre sistemas operativos Windows/Linux/BSD/UNIX.
	**Dia.-** Además de permitir el diseño de diagramas UML, permite el diseño de redes de computadores, diseño de circuitos electrónicos a nivel de puertas y dispositivos, etc. Funciona en sistemas operativos Windows 98 / NT / 2000 / ME / XP / Linux.
	**ArgoUML.-** Herramienta para el modelado de software que ofrece la posibilidad de generar código automáticamente para los lenguajes de programación Java, C++, C# y PHP. Funciona en sistemas operativos Windows 98 / NT / 2000 / ME / XP / Linux.
	**EclipseUML.-** Herramienta multiplataforma para el modelado visual de objetos orientada al lenguaje Java que se integra directamente en el entorno de desarrollo Eclipse. Genera código Java a partir de los diagramas UML. Permite realizar no sólo los diagramas UML de una aplicación, sino que también permite modelar y gestionar bases de datos visualmente.
	**UMLet.-** Herramienta Open Source en Java que permite dibujar los diferentes diagramas UML (clases, paquetes, estados, casos de uso, actividad, secuencia y colaboración). Puede ser ejecutado desde un archivo jar, o como un pluggin de Eclipse.
	**NetBeans.-** Éste IDE dispone de un pluggin para el modelado diagramas de clases, actividades, colaboración, componentes, despliegue, paquetes, secuencia, estado y casos de uso. A partir de dichos diagramas genera automáticamente el código fuente en el lenguaje de programación Java. Funciona en sistemas operativos Windows / Linux.
	**JDeveloper.-** Ésta IDE incluye herramientas de modelado UML, herramientas para la creación y orquestación de servicios web y para la gestión de flujos de negocio. Además incluye

	herramientas destinadas al desarrollo visual de interfaces de usuario. A pesar de que en sus inicios era de carácter comercial, actualmente se distribuye de manera gratuita.
poseidon for uml	**Poseidon for UML.-** Herramienta con excelente interfaz de usuario que permite la elaboración de diagramas de clases, paquetes, casos de uso, estado, componentes, actividad y secuencia. Además permite la generación de código fuente para el lenguaje de programación Java.
Uso Comercial	
Logotipo	**Descripción**
TogetherSoft	**Together.-** Está totalmente adaptada a Java y su característica más importante es la generación de código sincronizada con los diagramas. Soporta los lenguajes Java, C++ e IDL y sus próximas versiones incluirán Visual Basic y ASP.
Rational Rose 98 The world's leading visual modeling tool.	**Rational.-** Soporta la generación de código para Ada, Ansi C++, C++, Java, Visual Basic y CORBA. Puede trabajar en diseños de bases de datos con capacidad de soportar la integración de los datos y los requerimientos de aplicación a través de diseños lógicos y físicos.
ENTERPRISE ARCHITECT	**Enterprise Architect.-** Permite el desarrollo de todos los diagramas UML y la generación de código a los lenguajes ActionScript 2.0, Java, C#, C++, VB.NET, Delphi, Visual Basic, Python y PHP. Funciona sobre los sistemas operativos Win98 / WinME/ Windows2000 / WinXP / Windows2003.
magicdraw Architecture Made Simple	**MagicDraw UML.-** Soporta UML 2.0 y permite la generación de código fuente para los lenguajes de programación Java, EJB, C#, C, CORBA IDL, DDL, WSDL y XML Schema. Puede ser ejecutado en los sistemas operativos Windows 98/ME/NT/2000/XP/Vista, Solaris, OS / 2, Linux, HP-UX, AIX, MacOS y en cualquier lugar donde Java 5 o 6 sea compatible.

6.3.6. Comparativa de las Herramientas U – CASE más usadas

Para conocer algunas de las condiciones de trabajo en las que se enmarcan los desarrolladores de software en lo referente al análisis y diseño de sistemas, el día lunes 4 de mayo de 2009 se logró encuestar a un total de 58 personas entre los que constan estudiantes, egresados y docentes de la Carrera de Ingeniería en Sistemas de

la Universidad Nacional de Loja, egresados de la Carrera de Tecnología en Sistemas de Automatización del Instituto Superior Sudamericano y desarrolladores de software que laboran en diferentes empresas de la localidad.

En base a la información recopilada a partir de la Encuesta 01[22] se pudo establecer cuáles son las herramientas más usadas en nuestro medio. Posteriormente se procedió a realizar un contraste de las prestaciones brindadas por cada una de dichas herramientas y de sus fortalezas y debilidades. El resumen de dicho análisis se muestra a continuación en la Tabla 24.

[22] El modelo utilizado para la Encuesta 01 se encuentra en el Anexo 2 de la página 193.

Tabla 24: Comparativa de las Herramientas CASE más utilizadas en la ciudad de Loja

Herramienta	Enterprise	NetBeans	Omondo	Poseidon	Rational Rose	Magic Draw
Licencia	Comercial	Libre	Libre / Comercial	Libre / Comercial	Comercial	Comercial
Diagramas que soporta	Paquetes, Clases, Objetos, Estructura Compuesta, Componentes, Despliegue, Casos de Uso, Actividad, Estado, Comunicación, Secuencia, Tiempos y Vista de Interacción	Clases, Componentes, Despliegue, Casos de Uso, Actividad, Estado, Comunicación y Secuencia	Clases, Objetos, Componentes, Despliegue, Casos de Uso, Actividad, Estado, Comunicación, Secuencia y Robustez	Clases, Componentes, Despliegue, Casos de Uso, Actividad, Estado, Comunicación y Secuencia	Clases, Componentes, Despliegue, Casos de Uso, Actividad, Estado, Comunicación y Secuencia	Clases, Estructura Compuesta, Componentes, Despliegue, Casos de Uso, Actividad, Estado, Comunicación, Secuencia, Tiempos, Vista de Interacción, Robustez y más
Generación de código fuente	ActionScript, C, C#, C++, Delphi, Java, PHP, Python, Visual Basic Net y Visual Basic	Java	Java	Java	ANSI C++, Ada 83, Ada 95, CORBA, Java, Visual C++, Visual Basic	C#, C++, Java, CORBA IDL, DDL, EJB 2.0, XML Schema y WSDL

Generación de documentación	RTF y HTML	HTML	No	HTML y DOC	No	HTML
Ingeniería inversa	Si	Si	Si	Si	Si	Si
Navegación por el modelo	Si	Si	Si	Si	Si	Si
Exportación de diagramas	*.bmp, *.png, *.jpg, *.tga, *.gif, *.wmf y *.emf	*.jpg, *.svg y *.png	*.svg, *.gif, *.wmf y *.jpg	Todos los formatos gráficos estándares	No	*.eps, *.png, *.jpg, *.svg, *.tif, *.tiff, *.wmf y *.emf
Importación XMI	Si	No	No	No	No	Si
Exportación XMI	Si	No	Si	Si	No	Si
Distribución automática de componentes	Si	Si	No	Si	No	Si
Alineación de componentes	Si	No	Si	Si	No	Si
Visualización de diagramas a escala	No	No	Si	Si	No	Si

7. EVALUACIÓN DEL OBJETO DE INVESTIGACIÓN

El presente trabajo investigativo denominado **"Desarrollo de una Herramienta de análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp (ECMA – 334)"** dio como resultado final la construcción de la Herramienta **AleDan UML**, misma que fue concebida desde sus inicios, como una aplicación de uso libre, multiplataforma y basada en las especificaciones tanto de UML 2.0 como del Lenguaje de Programación C Sharp.

El objetivo general así como cada uno de los objetivos específicos lograron ser abarcados en su totalidad, gracias a la aplicación ordenada de metodologías orientadas al desarrollo de la investigación y al desarrollo de software:

- Analizar las herramientas de análisis y diseño de sistemas basadas en la especificación UML 2.0 más utilizadas por los desarrolladores de software.

  La realización de encuestas a desarrolladores de software en nuestra localidad, así como la constante investigación acerca de la temática abordada, permitió llevar a cabo el análisis de las herramientas UML más usadas en nuestro medio, y en base a ello, determinar los requerimientos que AleDan UML debería satisfacer.

- Desarrollar un módulo de diagramación basado en el Lenguaje de Programación C Sharp, y en la especificación UML 2.0 para modelar diagramas de clases, paquetes, componentes, despliegue, casos de uso y secuencia.

  Este objetivo se pudo ejecutar exitosamente puesto que toda la información recopilada en la fase anterior, paulatinamente se fue plasmando en el diseño y posterior desarrollo de los diferentes módulos que hacen parte de la herramienta, dándole cada vez, más y mejores prestaciones para el usuario.

  El uso de una apropiada plataforma de desarrollo, que incluye sistema operativo, lenguaje de programación, entorno de desarrollo y control de versiones, aportó en gran medida en el proceso de codificación llevado a cabo.

- Desarrollar un módulo de generación de código fuente para el Lenguaje de Programación C Sharp, a partir de los diagramas de clases y secuencia.

  Al igual que en el caso anterior, los conocimientos adquiridos a lo largo del continuo proceso investigativo y el empleo de una adecuada plataforma de desarrollo, proporcionaron los elementos necesarios para la puesta de marcha del proceso de codificación de este nuevo producto software.

  El estudio del lenguaje de programación C# y del lenguaje de modelado UML 2.0, nos permitió conocer la estructura básica que se debería contemplar en la elaboración de diagramas y en la generación de código.

Habiéndose logrado cumplir satisfactoriamente con cada uno de los objetivos planteados en el presente trabajo de investigación, es que ahora podemos contar con una aplicación que está en capacidad de:

- Detectar errores de diagramación de acuerdo a las especificaciones de UML 2.0.
- Permitir la creación de los seis tipos de diagramas UML 2.0 más utilizados por los desarrolladores de software en nuestro medio: clases, paquetes, componentes, despliegue, casos de uso y secuencia.
- Permitir la generación de código fuente para el Lenguaje de Programación C Sharp a partir del diseño previo de diagramas.
- Reducir la necesidad de trabajar con aplicaciones comerciales con o sin licencia, sin dejar de lado las mejores características de funcionalidad.

Es de suma importancia impulsar el desarrollo de software libre no sólo por sus mínimos o nulos costos de licencia, sino más aún, por la posibilidad que brinda a los desarrolladores de conocer la manera de cómo fue construida una aplicación, incrementando así nuevos conocimientos e ideas innovadoras.

Esperamos que **AleDan UML**, a más de constituirse en una herramienta que ayude en el proceso de enseñanza – aprendizaje, se convierta en un punto de partida no sólo para el mejoramiento de sus funcionalidades, sino también, para el desarrollo de nuevos y mejores proyectos de investigación.

8. DESARROLLO DE LA PROPUESTA ALTERNATIVA

8.1. INICIO O INCEPCIÓN

8.1.1. Población y muestra para la captura de requerimientos

El universo poblacional tomado en cuenta para ésta etapa de la investigación lo conforman los desarrolladores de software de la ciudad de Loja y dado que no se conoce con precisión el volumen de la población a encuestar, se procedió a determinar el tamaño de la muestra de acuerdo a la siguiente fórmula[23]:

$$n = \frac{Z^2 pq}{E^2}$$

Donde:

- n es el tamaño de la muestra = ?
- Z es el nivel de confianza = 90% = 1,65
- p es la variabilidad positiva = 0,5
- q es la variabilidad negativa = 0,5
- E es el margen de error = 10% = 0,1

En base a ello se pudo determinar lo siguiente:

$$n = \frac{(1{,}65)^2(0{,}7)(0{,}3)}{(0{,}1)^2} = \frac{0{,}571725}{0.01} = 57{,}1725 \sim 58$$

Por lo tanto, el tamaño de la muestra es de 58 desarrolladores de software.

8.1.2. Requerimientos generales del proyecto

Basándonos en los resultados obtenidos tanto en el análisis de la Encuesta 01[24] aplicada a 58 desarrolladores de software, como en el análisis de las herramientas

[23] Revisar Tamaño de muestras en el Anexo 1 de la página 191.
[24] El modelo utilizado para la Encuesta 01 se encuentra en el Anexo 2 de la página 193.

UML más usadas por los mismos[25], se han identificado los requerimientos que la presente aplicación deberá cumplir.

Se ha creído conveniente asignar a cada uno de ellos un nivel de prioridad (**P**) que comprende un rango de 01 a 22, mismo que será estipulado de acuerdo a la necesidad de incluirlos en la herramienta y tomando en consideración si están o no dentro del alcance delimitado. Se debe destacar que aquellos requerimientos con mayor prioridad tienen un valor numérico más bajo.

8.1.2.1. Requerimientos Funcionales

El sistema permitirá:

Tabla 25: Requerimientos funcionales preliminares

Requerimiento	P
Administrar un proyecto (abrir, crear, modificar, eliminar)	01
Administrar Diagrama de Clases (crear, modificar, eliminar)	03
Administrar Diagrama de Paquetes (crear, modificar, eliminar)	04
Administrar Diagrama de Componentes (crear, modificar, eliminar)	05
Administrar Diagrama de Despliegue (crear, modificar, eliminar)	06
Administrar Diagrama de Casos de Uso (crear, modificar, eliminar)	07
Administrar Diagrama de Secuencia (crear, modificar, eliminar)	08
Administrar artefactos (crear, modificar, eliminar)	02
Generar código fuente para el Lenguaje de Programación C# a partir de los diagramas realizados previamente (clases y secuencia)	09
Generar código fuente para el Lenguaje de Programación Java a partir de los diagramas realizados previamente	21
Exportar los diagramas en un formato PNG	13
Presentar un cuadro de propiedades para la configuración de nodos y relaciones	10
Distribución preestablecida (layout) de los artefactos UML	17
Alineación de los artefactos UML	18
Permitir la edición de las propiedades gráficas de los artefactos UML	12
El sistema deberá contar con las propiedades de Zoom In y Zoom Out	15
Permitir la visualización de diagramas a escala	16

[25] El análisis de las herramientas basadas en UML 2.0 se encuentra en el Anexo 5 de la página 211. El cuadro comparativo de dichas herramientas se encuentra en la página 77.

El sistema deberá contar con las propiedades de Hacer y Deshacer acciones	14
Importación XMI	20
Exportación XMI	19
Visualizar los errores detectados a través de una consola	11

8.1.3. Alcance

El sistema como mínimo debe permitir la administración de los siguientes tipos de diagramas: clases, paquetes, componentes, despliegue, casos de uso y secuencia; la exportación de un diagrama a un archivo de imagen; la impresión de diagramas y la generación de código fuente para el lenguaje de programación C Sharp.

8.1.4. Modelo inicial de Casos de Uso

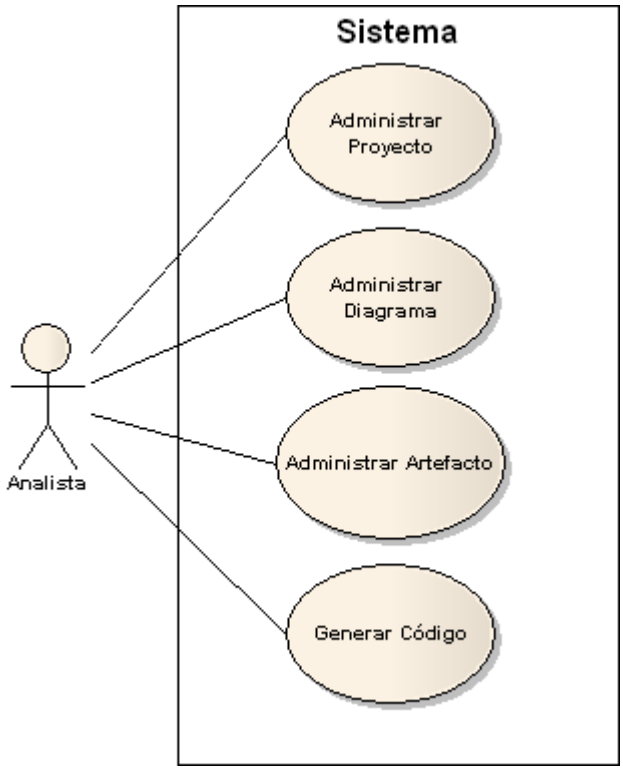


Figura 28: Modelo inicial de Casos de Uso

8.1.5. Arquitectura base del sistema

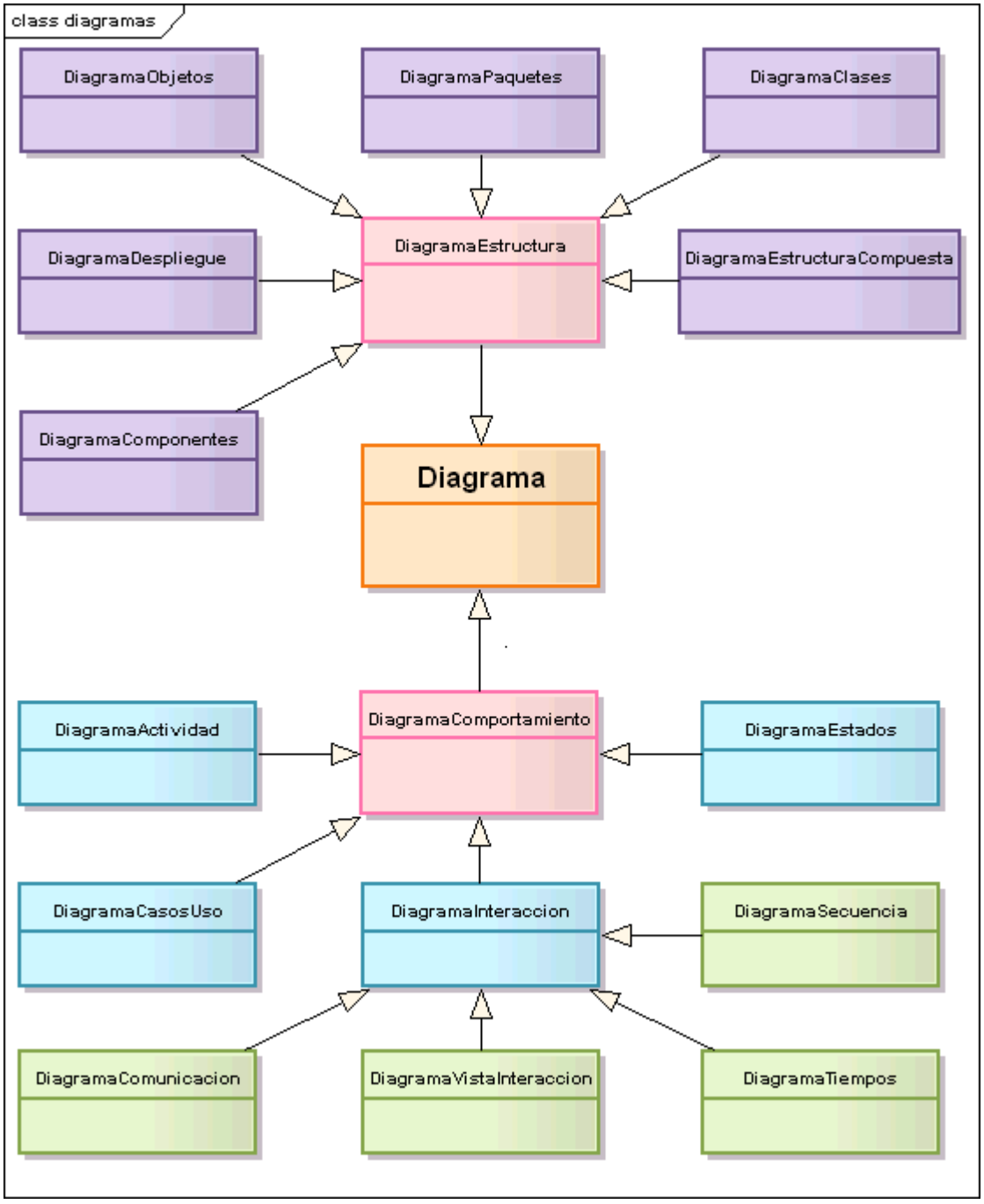


Figura 29: Clasificación de Diagrama

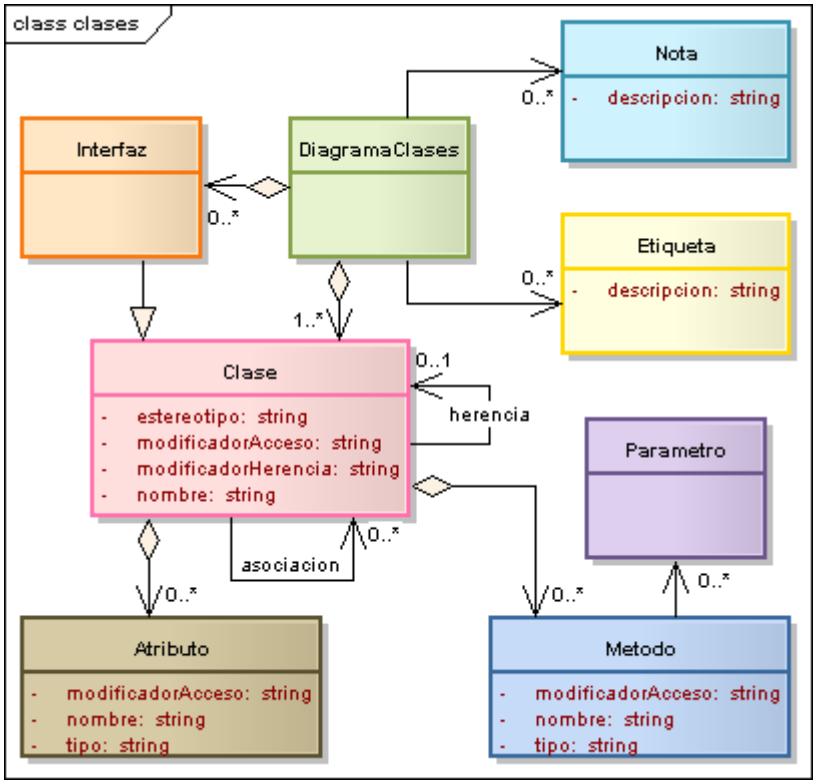


Figura 30: Arquitectura de Diagrama de Clases

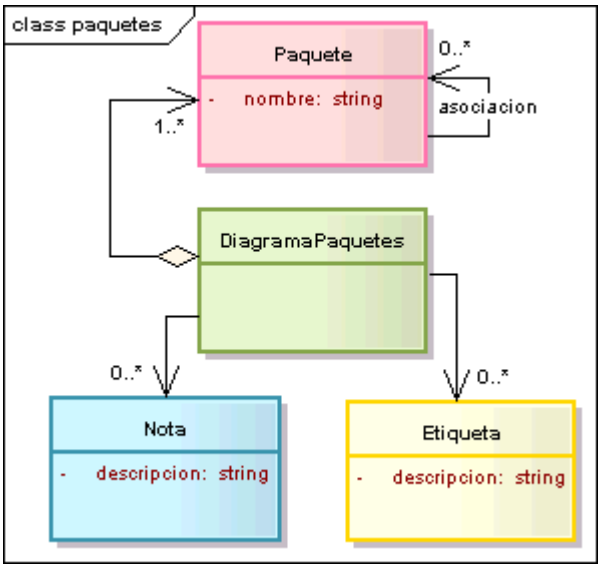


Figura 31: Arquitectura de Diagrama de Paquetes

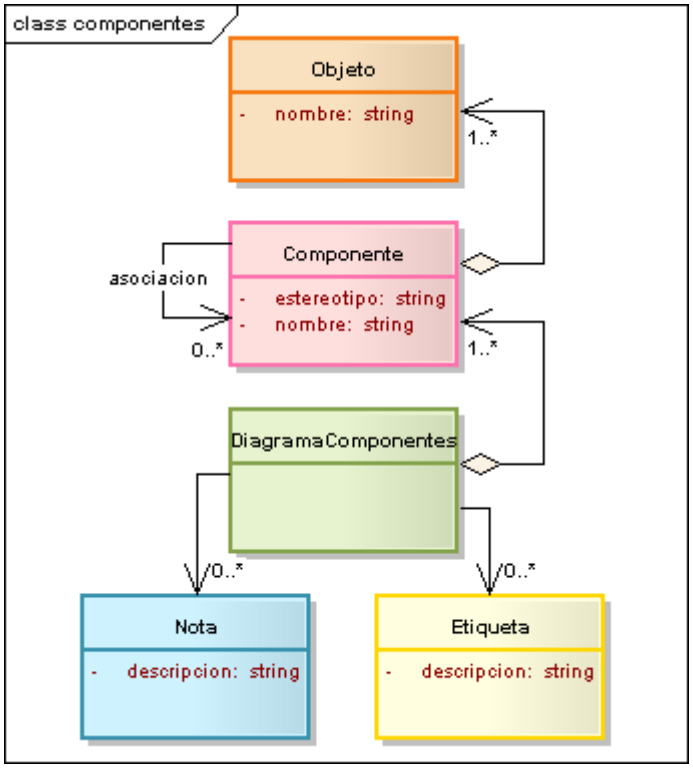


Figura 32: Arquitectura de Diagrama de Componentes

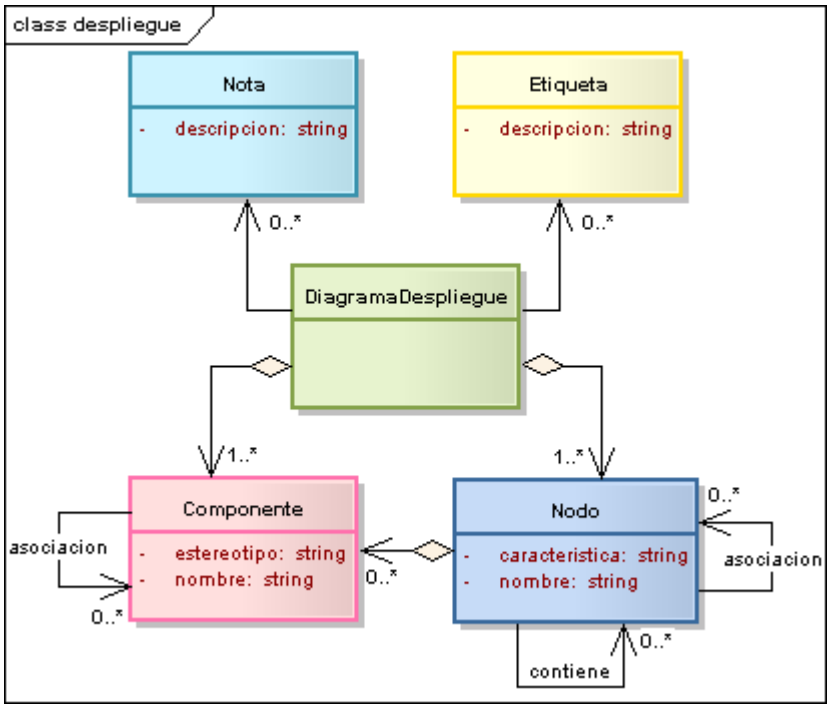


Figura 33: Arquitectura de Diagrama de Despliegue

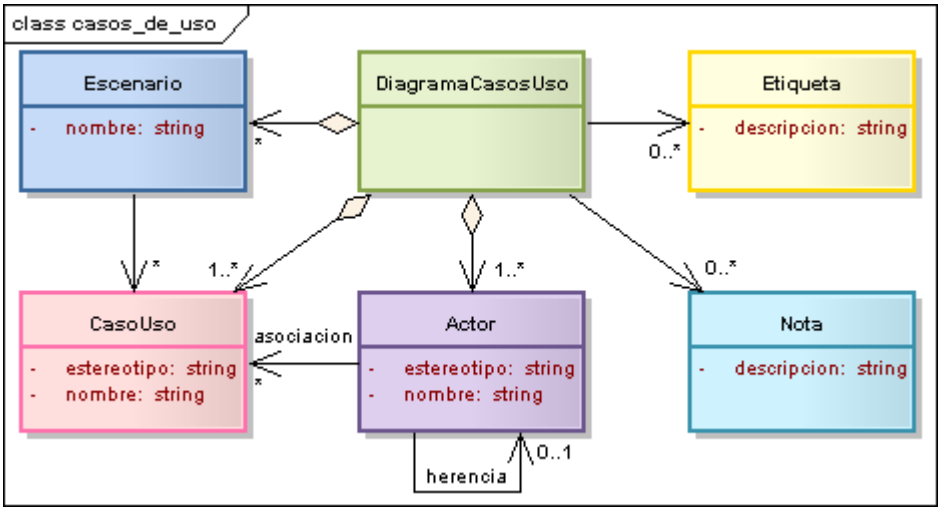


Figura 34: Arquitectura de Diagrama de Casos de Uso

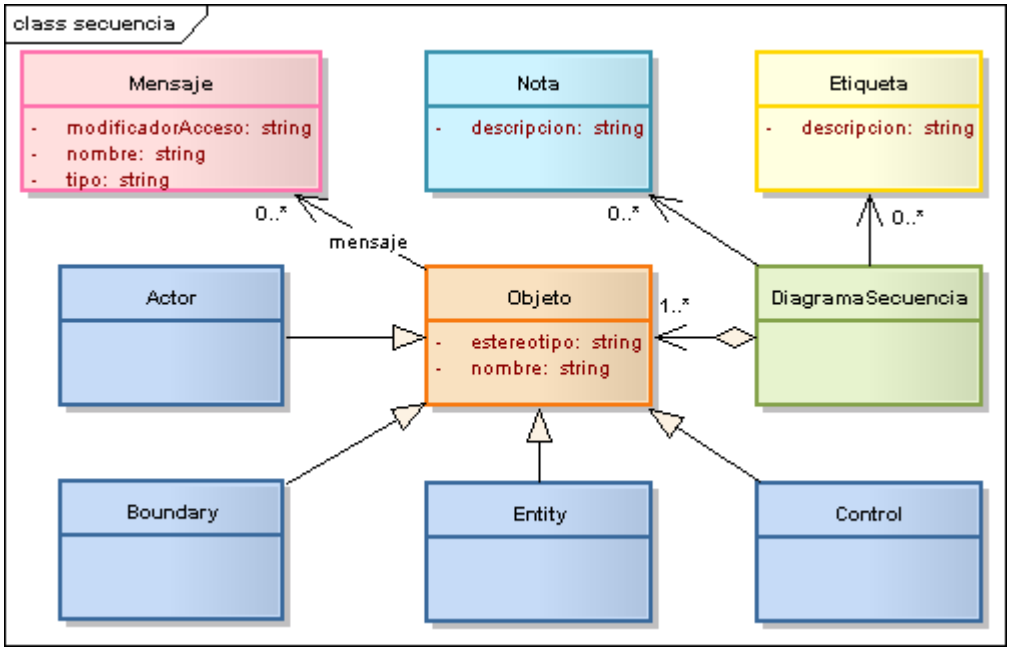


Figura 35: Arquitectura de Diagrama de Secuencia

8.2. ELABORACIÓN

8.2.1. Descripción del problema

Las herramientas para el desarrollo de software evolucionan constantemente gracias a los continuos avances científicos presentados día a día. La demanda de aplicaciones de buena calidad, orientadas al cliente, desarrolladas a bajo costo y en corto tiempo, se incrementa constantemente. La aparición de nuevas tecnologías ha ayudado en

gran medida a disminuir la inversión de recursos tanto en tiempo como en dinero, aspecto imposible de realizar hace tal vez 10 o 15 años.

En la actualidad los desarrolladores de software deben seleccionar una herramienta de análisis y diseño que presente facilidades de uso, que sea eficiente y que enmarque al proyecto dentro un proceso ordenado y sistemático que asegure su éxito.

Hablando específicamente del caso de las Herramientas CASE, si bien es cierto que existen gran variedad de opciones a la hora de utilizar una, la gran mayoría es software comercial orientado a funcionar sobre la plataforma Windows, lo que obliga a los desarrolladores a dejar de lado a otros sistemas operativos. Si a ello sumamos los altos precios con los que se comercializan, resulta prácticamente imposible adquirir las licencias de dichas herramientas, especialmente para aquellos desarrolladores que apenas comienzan con su actividad.

Por su parte las Herramientas CASE de uso libre no reúnen las características necesarias para ser utilizadas de manera permanente, ya que en ciertos casos no permiten la elaboración de los diagramas UML 2.0 más usados en nuestro medio o no se basan completamente en sus especificaciones; tal vez no permiten la generación de código fuente a partir de dichos diseños; o bien no poseen una interfaz amigable, haciendo que su uso pueda tornarse complejo y confuso para el usuario.

Por otro lado, a pesar de que C Sharp es un lenguaje de programación 100% orientado a objetos y a componentes, lamentablemente no existe una herramienta gratuita y open source[26] que ayude a diseñar una arquitectura enfocada a éste paradigma y mucho menos, a generar código fuente para éste lenguaje de programación. Esta situación sin duda alguna puede ocasionar que el desarrollador caiga en una programación estructurada a pesar de trabajar con un lenguaje orientado a objetos. Aunque algunas empresas han desarrollado plugin's (módulos) con soporte UML, ya sea para Visual Studio o MonoDevelop, lo que persiguen es una utilidad comercial y privativa.

Por lo antes mencionado se puede prácticamente asegurar que para los desarrolladores de software es inminente el riesgo de adquirir malos hábitos durante la

[26] Las herramientas open source o de código abierto permiten modificar el código fuente para implantar mejoras o nuevas funcionalidades a las mismas.

construcción de aplicaciones, ya sea codificar el proyecto para luego proceder a ejecutar la etapa de análisis y diseño, dejar de lado la documentación de la arquitectura del sistema, utilizar demos de herramientas CASE para análisis y diseño, o en el peor de los casos, utilizar licencias ilegales comúnmente conocidas como “piratas”. El único resultado esperado de ésta situación es la pérdida de tiempo por un mal diseño y los consecuentes retrasos en la planificación.

El **“Desarrollo de una Herramienta de análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp (ECMA – 334)”** se plantea con la finalidad de ofrecer una alternativa de solución ante la problemática antes mencionada, ya que se pretende crear una herramienta de diseño capaz de satisfacer los requerimientos de los desarrolladores que se orientan a realizar su trabajo en el lenguaje de programación C Sharp, ayudando a minimizar los riesgos de retrasos en la planificación asociados al mal diseño.

8.2.2. Determinación de Requerimientos

Se ha llevado a cabo un filtrado de requerimientos con la finalidad de obtener aquellos que se consideran más críticos para crear una herramienta de calidad.

8.2.2.1. Requerimientos Funcionales

El sistema permitirá:

Tabla 26: Requerimientos Funcionales

Código	Descripción	Categoría
RF01	Administrar un proyecto (crear, modificar, eliminar)	Evidente
RF02	Administrar artefactos (crear, modificar, eliminar)	Evidente
RF03	Administrar Diagrama de Clases (crear, modificar, eliminar)	Evidente
RF04	Administrar Diagrama de Paquetes (crear, modificar, eliminar)	Evidente
RF05	Administrar Diagrama de Componentes (crear, modificar, eliminar)	Evidente
RF06	Administrar Diagrama de Despliegue (crear, modificar, eliminar)	Evidente
RF07	Administrar Diagrama de Casos de Uso (crear, modificar, eliminar)	Evidente
RF08	Administrar Diagrama de Secuencia (crear, modificar, eliminar)	Evidente

RF09	Abrir un proyecto existente	Evidente
RF10	Generar código fuente para el Lenguaje de Programación C# a partir de los diagramas de clases y secuencia	Evidente
RF11	Presentar un cuadro de propiedades para la configuración de nodos y relaciones	Evidente
RF12	Visualizar los errores detectados a través de una consola	Oculto
RF13	Permitir la edición de las propiedades gráficas de los artefactos UML.	Evidente
RF14	Exportar los diagramas en un formato PNG	Evidente
RF15	El sistema deberá contar con las propiedades de Zoom In y Zoom Out	Evidente

8.2.2.2. Requerimientos no Funcionales

El sistema:

Tabla 27: Requerimientos no Funcionales

Código	Descripción
RNF01	Cumplirá con las especificaciones dadas por UML 2.0 y ECMA – 334
RNF02	Será desarrollada en el lenguaje de programación Java
RNF03	Utilizará el Patrón Controlador de Casos de Uso para dar funcionalidad a la interfaz gráfica de usuario
RNF04	Tendrá un módulo de ayuda para el manejo de la aplicación
RNF05	Deberá contar con una interfaz gráfica de usuario interactiva, intuitiva y amigable
RNF06	Tendrá un formato de documentación estandarizado que permita realizar actualizaciones y mantenimiento a la información, de forma rápida y fácil
RNF07	El sistema no deberá demandar demasiados recursos del ordenador
RNF08	Software libre
RNF09	Podrá ser ejecutada en los Sistemas Operativos Windows XP / Vista / 7 y Linux
RNF10	Idioma español

8.2.3. Modelo de Casos de Uso

8.2.3.1. Identificación de Casos de Uso

Tabla 28: Identificación de Casos de Uso

Actor	Meta	Código Meta	Caso de Uso	Código CU
Analista	Crear proyecto	MT01	Administrar Proyecto	CU01
	Imprimir proyecto	MT02		
	Abrir proyecto	MT03		
	Renombrar proyecto	MT04		
	Administrar objeto de modelo	MT05		
	Guardar proyecto	MT06		
	Guardar proyecto como	MT07		
	Cerrar proyecto	MT08		
	Eliminar proyecto	MT09		
	Crear diagrama	MT10	Administrar Diagrama	CU02
	Imprimir diagrama	MT11		
	Abrir diagrama	MT12		
	Editar diagrama	MT13		
	Exportar diagrama en formato *.png	MT14		
	Cerrar diagrama	MT15		
	Eliminar diagrama	MT16		
	Agregar artefacto	MT17	Administrar Artefacto	CU03
	Editar artefacto	MT18		
	Administrar Atributo	MT19		
	Administrar Método	MT20		
	Administrar Mecanismo de Extensibilidad	MT21		
	Eliminar artefacto	MT22		
	Generar código fuente para C#	MT23	Generar Código	CU04

8.2.3.2. Diagramas de Casos de Uso

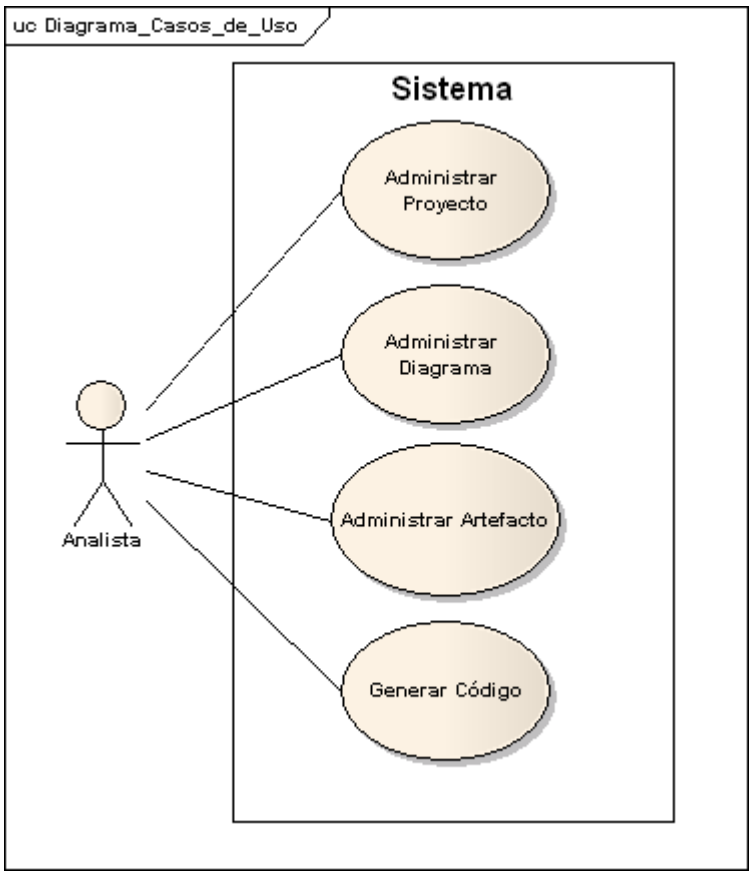


Figura 36: DCU General

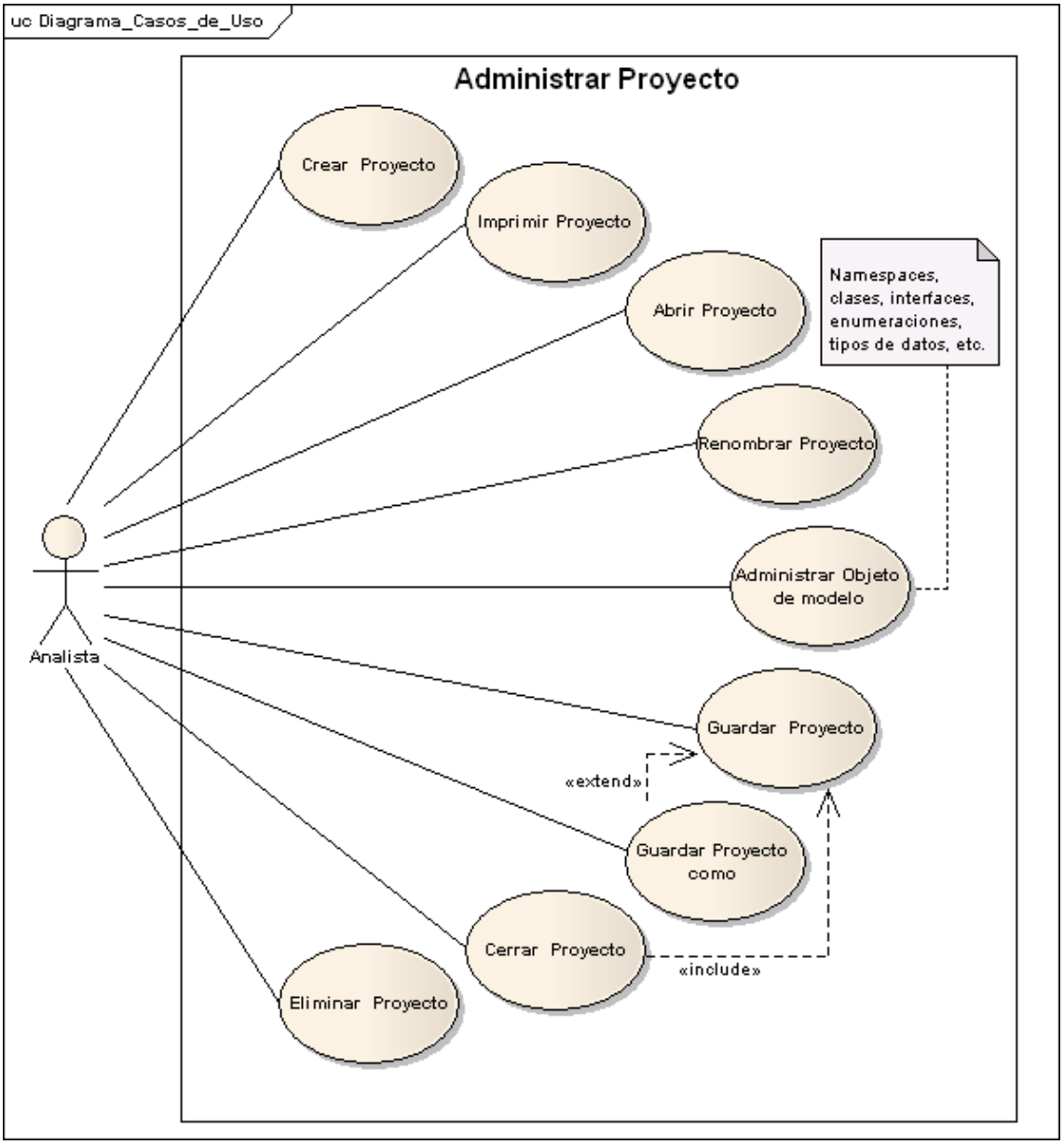


Figura 37: DCU Administrar Proyecto

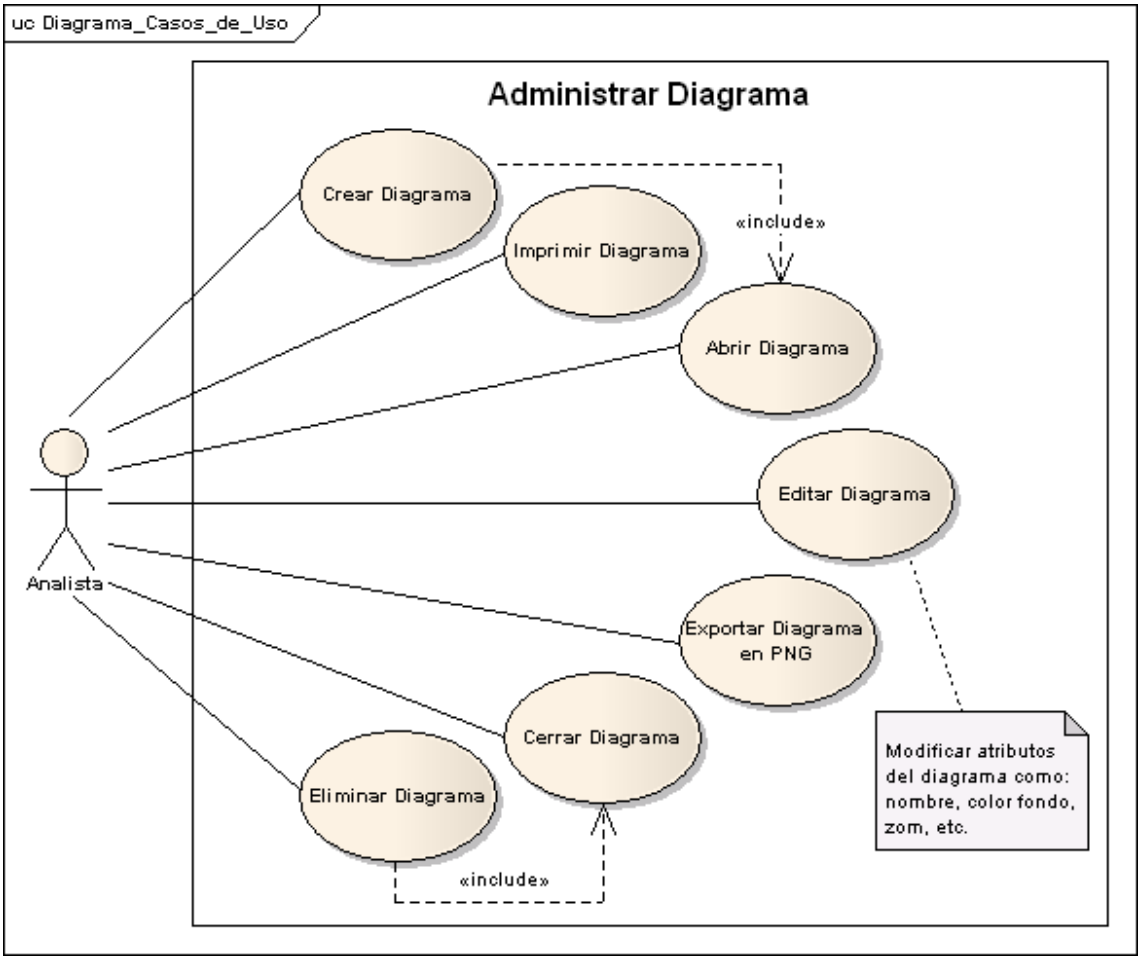


Figura 38: DCU Administrar Diagrama

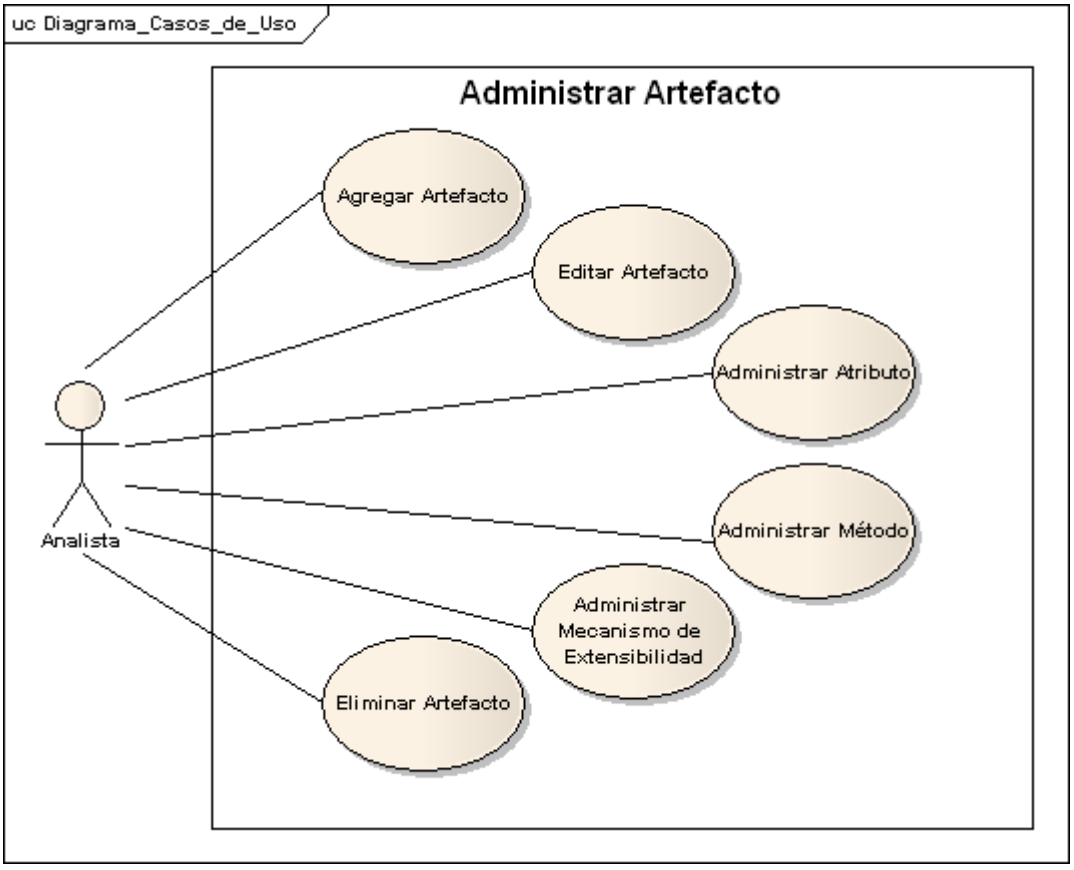


Figura 39: DCU Administrar Artefacto

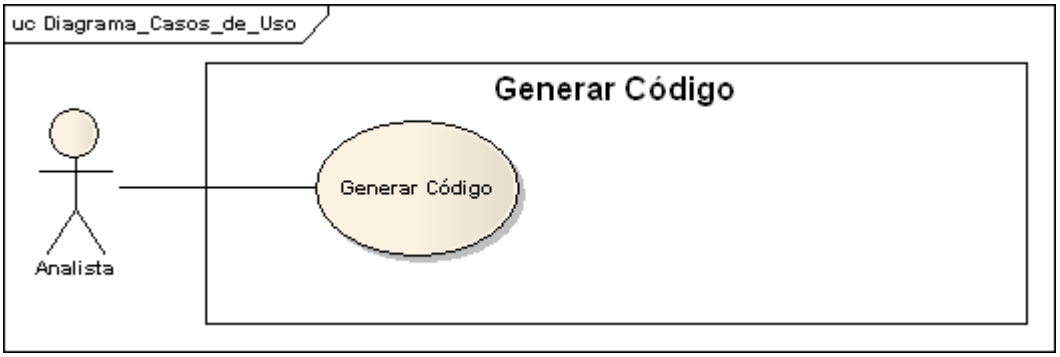


Figura 40: DCU Generar Código

8.2.3.3. Descripción de casos de uso

Tal como se puede apreciar en la Figura 41, la pantalla principal de la herramienta AleDan UML se encuentra estructurada de la siguiente manera:

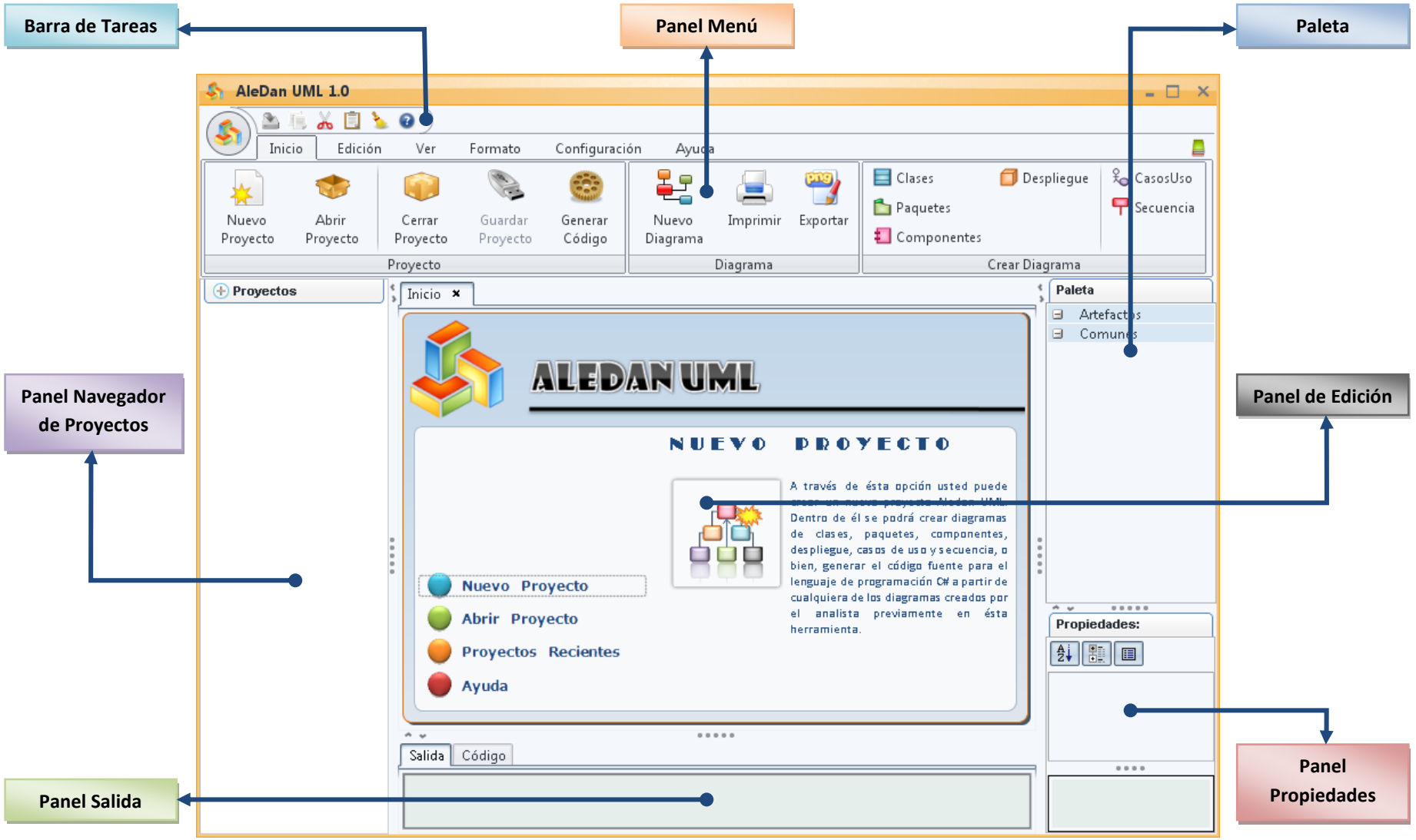


Figura 41: Pantalla Principal MainAledanUML

8.2.3.3.1. **Caso de Uso:** Administrar Proyecto

8.2.3.3.1.1. **Meta MT01:** Crear proyecto

Tabla 29: Prototipo de la Pantalla MainAledanUML

Nombre de la Pantalla:	MainAledanUML	**Código:**	P001
Tipo de Interfaz Gráfica:	JFrame		

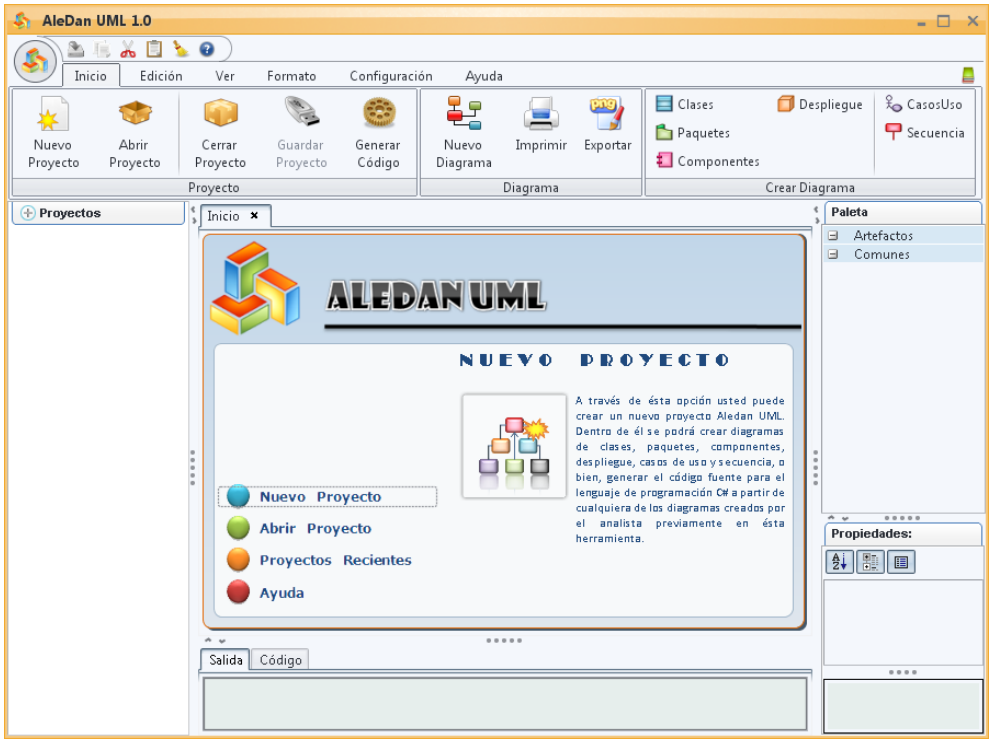


Tabla 30: Prototipo de la Pantalla AsistenteProyecto

Nombre de la Pantalla:	AsistenteProyecto	**Código:**	P002
Tipo de Interfaz Gráfica:	JFileChooser		
Caso:	Crear proyecto		

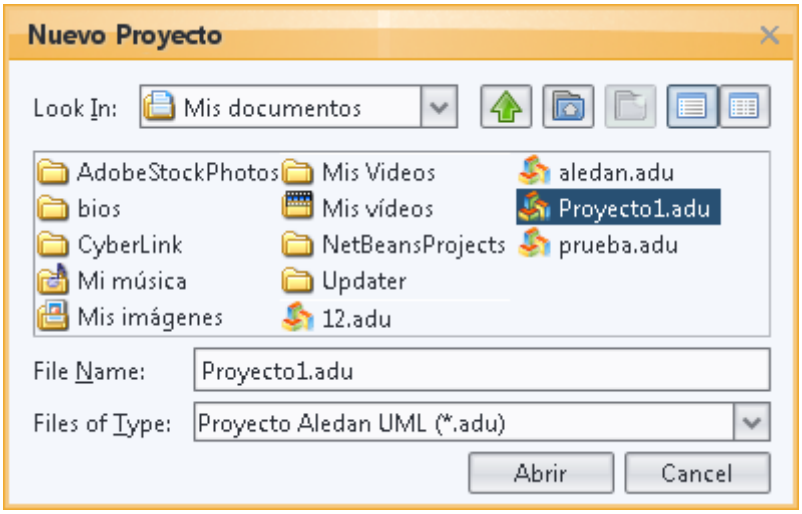


Tabla 31: Descripción de la Meta Crear proyecto

Caso de Uso:	Administrar Proyecto	**Código Caso de Uso:**	CU01
Meta:	Crear proyecto	**Código Meta:**	MT01
Tipo:	Primario		
Referencia a requerimientos:	RF01		
Actor:	Analista		
Propósito:	Permitir la creación de un nuevo proyecto		
Descripción:	El analista crea un nuevo proyecto con nombre y ubicación asignados por defecto, o bien, dando al programa la ubicación y el nombre del archivo; cuando el analista acepta la operación, el sistema crea un nuevo espacio de trabajo para dicho proyecto		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML]		
Post condiciones:	• Crear un diagrama • Editar un diagrama • Guardar un proyecto • Cerrar un proyecto • Eliminar un proyecto		
Curso Normal de Eventos			
Acción del Actor		**Respuesta del Sistema**	
1. Selecciona la opción [Nuevo Proyecto] en el menú [Inicio], o bien, en el panel de bienvenida de la pantalla [P001: MainAledanUML]		**2.** Asigna un nombre por defecto al nuevo proyecto	
		3. Presenta la pantalla [P002: AsistenteProyecto]	
4. Selecciona la ubicación en la que se deberá crear el nuevo proyecto			
5. Ingresa el nombre del nuevo proyecto, si así lo prefiere			
6. Presiona la opción [Crear] de la pantalla [P002: AsistenteProyecto]		**7.** Valida que el campo [Nombre de archivo] de la pantalla [P002: AsistenteProyecto] no esté vacío y que el nombre del proyecto no exista en la ubicación seleccionada	
		8. Crea el espacio de trabajo para el nuevo proyecto con el nombre y la ubicación asignados	
		9. Muestra el nuevo proyecto en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]	
		10. Agrega el nuevo proyecto a la lista [Proyectos Recientes]	
		11. La meta **MT01** finaliza	
Curso Alterno de Eventos			
A. Reemplazar			
Acción del Actor		**Respuesta del Sistema**	
		A.8. Presenta un mensaje de aviso indicando al analista que el proyecto ya existe y preguntando si desea reemplazarlo	
A.9. Si el analista confirma el mensaje de aviso, la meta **MT01** continúa en el		**A.10.** Verifica que el proyecto a reemplazar no se encuentre abierto	

paso **A.10 del Curso Alterno de Eventos Reemplazar**. Caso contrario, la creación se cancela y la meta **MT01** finaliza	
	A.11. Sobrescribe el proyecto
	A.12. La meta **MT01** continúa en el **paso 8 del Curso Normal de Eventos**.
B. Crear proyecto a partir del menú de la aplicación	
Acción del Actor	**Respuesta del Sistema**
B.1. Hace clic sobre el icono de la aplicación ubicado en la esquina superior izquierda de la pantalla [P001: MainAledanUML]	**B.2.** Despliega el menú de la aplicación
B.3. Selecciona la opción [Nuevo] del menú desplegado	**B.4.** Despliega un submenú de opciones
B.5. Selecciona la opción [Proyecto] del submenú desplegado	**B.6.** La meta **MT01** continúa en el **paso 2 del Curso Normal de Eventos**.

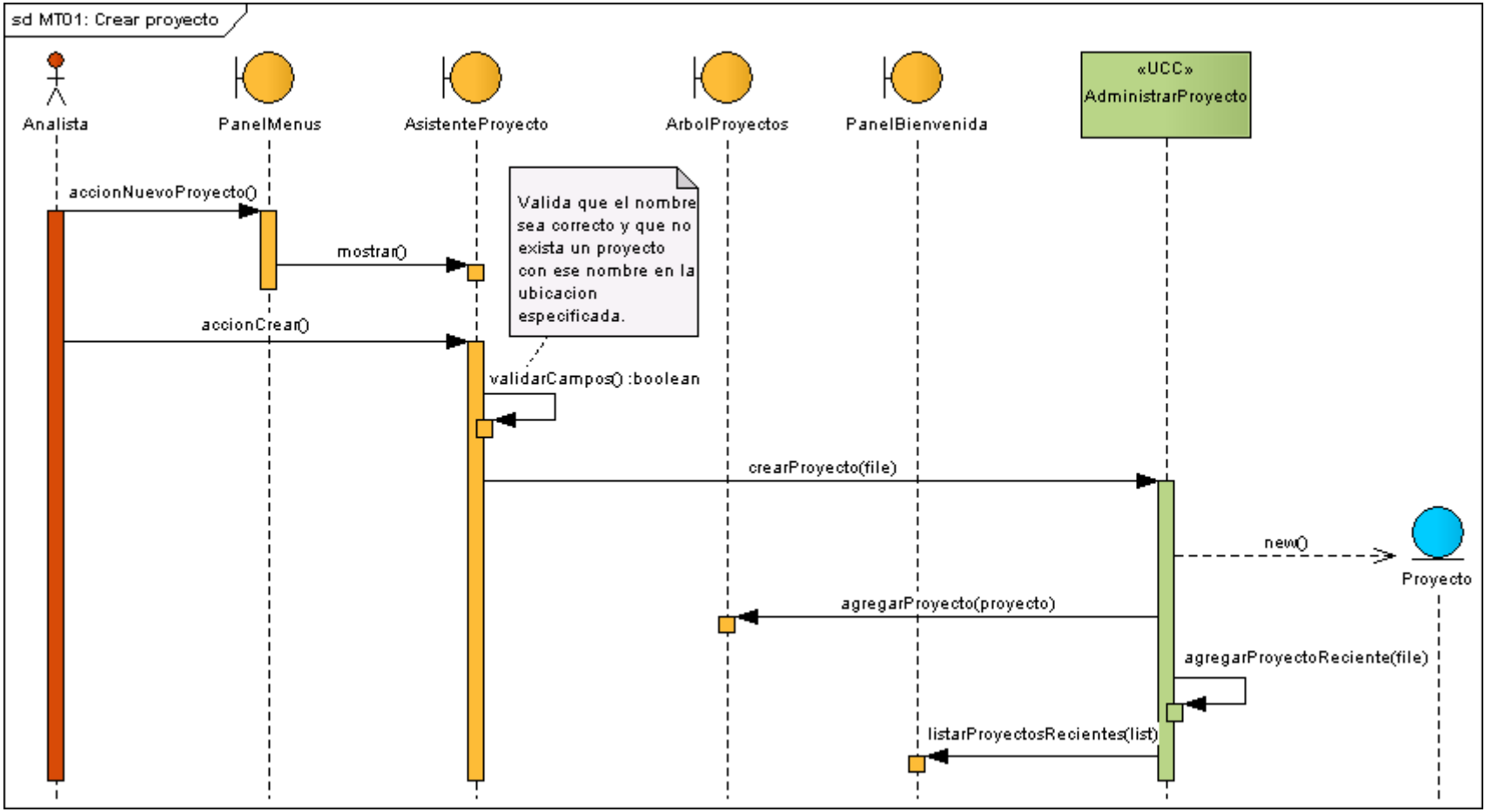


Figura 42: DS Curso normal crear proyecto

8.2.3.3.1.2. **Meta MT02:** Imprimir proyecto

Tabla 32: Prototipo de la Pantalla AsistenteImpresionProyecto

Nombre de la Pantalla:	AsistenteImpresionProyecto	**Código:**	P006
Tipo de Interfaz Gráfica:	JDialog		

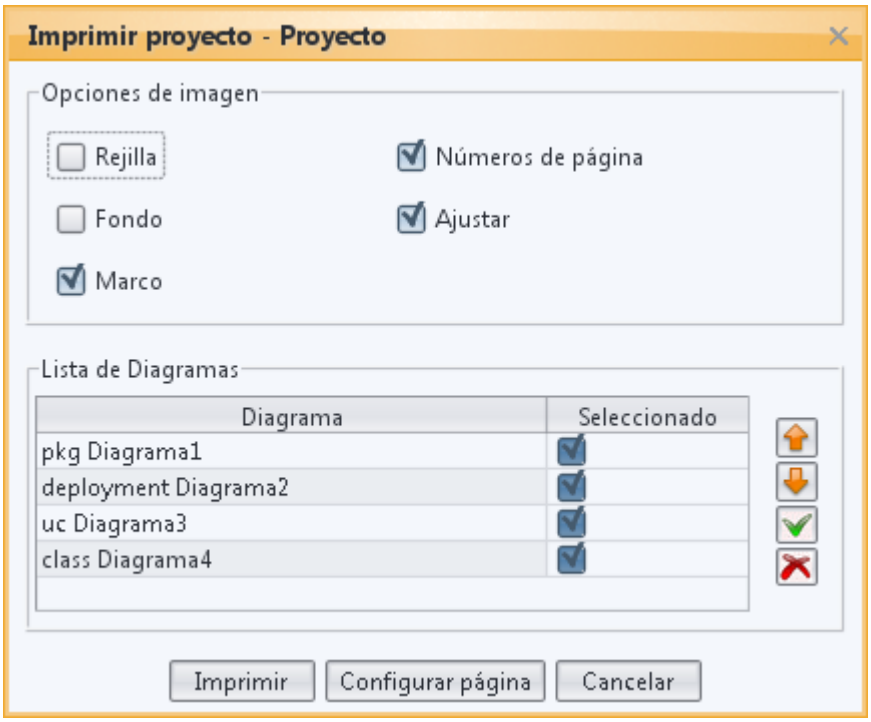


Tabla 33: Prototipo de la Pantalla Configurar página

Nombre de la Pantalla:	Configurar página	**Código:**	P007
Tipo de Interfaz Gráfica:	PrintJob		

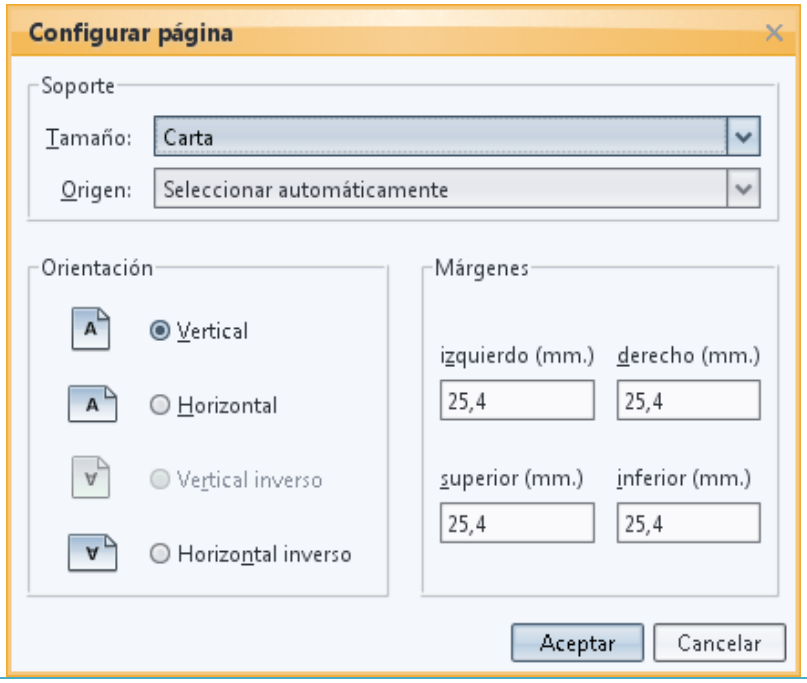


Tabla 34: Descripción de la Meta Imprimir proyecto

Caso de Uso:	Administrar Proyecto	**Código Caso de Uso:**	CU01
Meta:	Imprimir proyecto	**Código Meta:**	MT02
Tipo:	Primario		
Referencia a requerimientos:	RF01		
Actor:	Analista		
Propósito:	Permitir la impresión de todos los diagramas existentes dentro de un proyecto		
Descripción:	El analista selecciona el proyecto que desea imprimir y ejecuta dicha acción; el sistema solicita las preferencias de impresión y si el analista acepta la operación, el sistema da inicio al servicio de impresión		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un diagrama creado previamente dentro del proyecto seleccionado		

Curso Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Hace clic derecho sobre el nombre del proyecto que desea imprimir en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]	**2.** Obtiene el proyecto seleccionado
	3. Despliega un menú de opciones
4. Selecciona la opción [Imprimir] del menú desplegado	**5.** Presenta la pantalla [P006: AsistenteImpresionProyecto]
6. Selecciona los casilleros correspondientes a sus preferencias de impresión en el panel [Opciones de imagen]	
7. Selecciona los diagramas que desea imprimir en el panel [Lista de Diagramas]	
8. Presiona la opción [Configurar página] de la pantalla [P006: AsistenteImpresionProyecto]	**9.** Presenta la pantalla [P007: Configurar página]
10. Define los atributos de página	
11. Selecciona la opción [Aceptar] de la pantalla [P007: Configurar página]	
12. Oprime la opción [Imprimir] de la pantalla [P006: AsistenteImpresionProyecto]	**13.** Inicia el servicio de impresión
	14. La Meta **MT02** finaliza
Curso Alterno de Eventos	
A. No existen diagramas	
Acción del Actor	**Respuesta del Sistema**
	A.3. Presenta un mensaje de aviso indicando al usuario que el proyecto no contiene diagramas para su impresión
	A.4. La meta **MT02** finaliza
B. Diagrama no seleccionado	
Acción del Actor	**Respuesta del Sistema**
	B.13. Presenta un mensaje de aviso indicando al usuario que no se han seleccionado diagramas para imprimir
	B.14. La meta **MT02** finaliza

C. Imprimir proyecto a partir del menú de la aplicación	
Acción del Actor	**Respuesta del Sistema**
C.1. Selecciona el proyecto que desea imprimir en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]	
C.2. Hace clic sobre el icono de la aplicación ubicado en la esquina superior izquierda de la pantalla [P001: MainAledanUML]	**C.3.** Despliega el menú de la aplicación
C.4. Selecciona la opción [Imprimir Proyecto] del menú desplegado	**C.5.** Obtiene el proyecto seleccionado
	C.6. La meta **MT02** continúa en el **paso 5 del Curso Normal de Eventos**

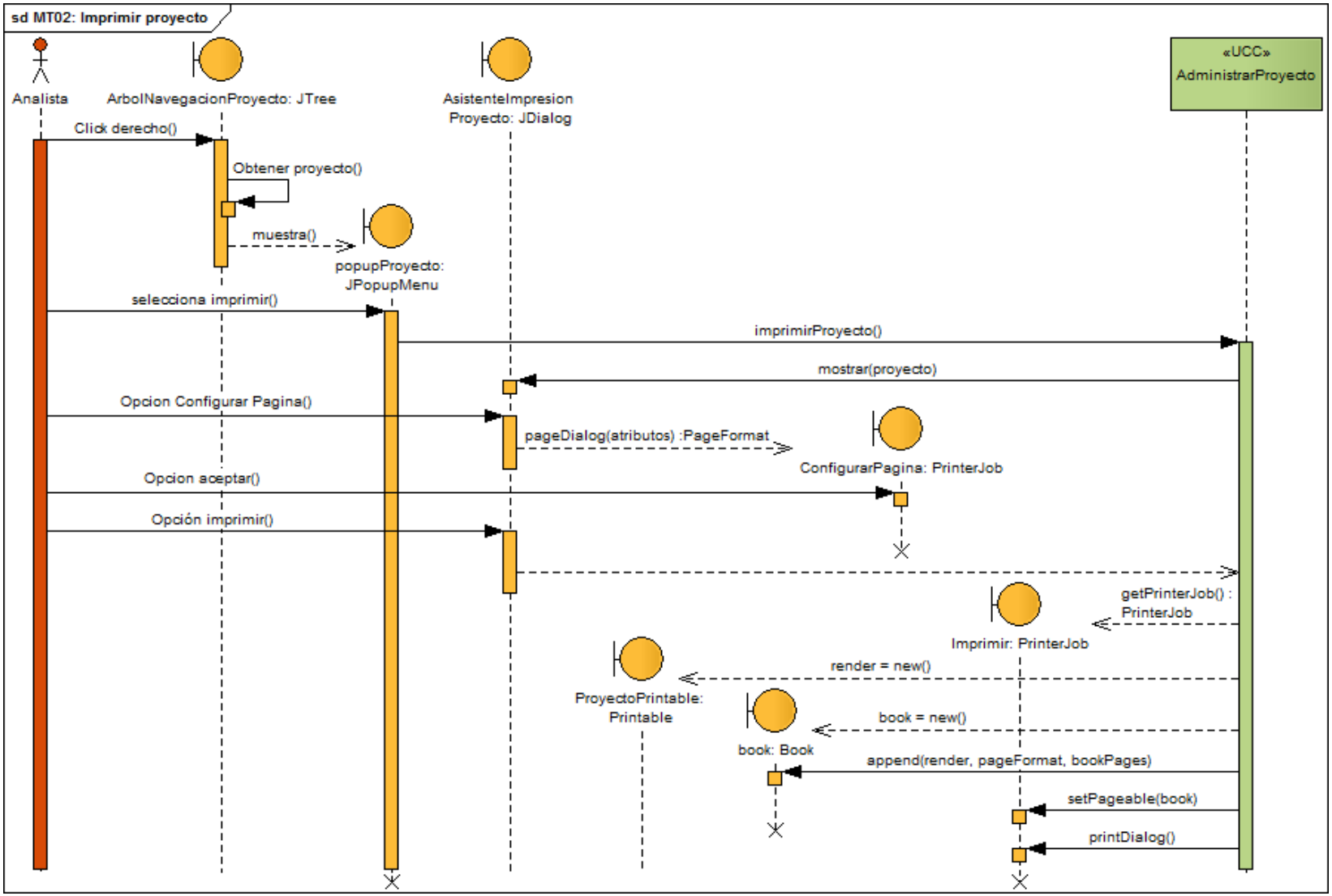


Figura 43: DS Curso normal imprimir proyecto

8.2.3.3.1.3. **Meta MT03:** Abrir proyecto

Tabla 35: Prototipo de la Pantalla AsistenteProyecto – Abrir proyecto

Nombre de la Pantalla:	AsistenteProyecto	**Código:**	P002
Tipo de Interfaz Gráfica:	JFileChooser		
Caso:	Abrir proyecto		

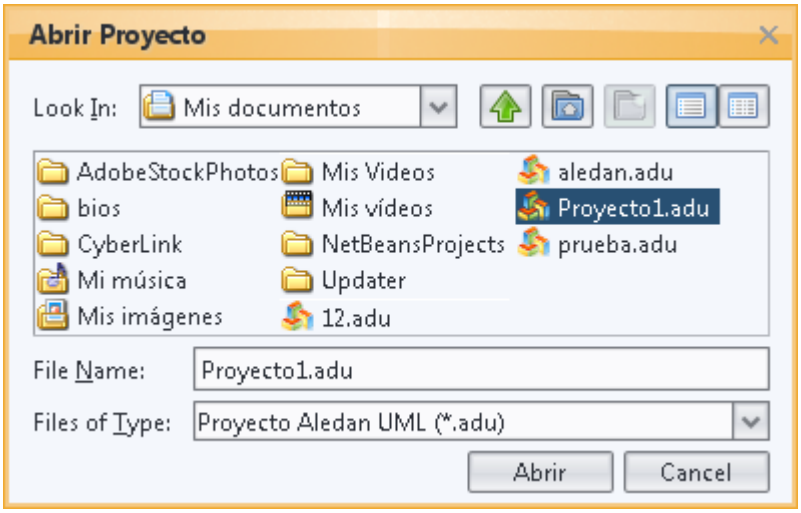


Tabla 36: Descripción de la Meta Abrir proyecto

Caso de Uso:	Administrar Proyecto	**Código Caso de Uso:**	CU01
Meta:	Abrir proyecto	**Código Meta:**	MT03
Tipo:	Primario		
Referencia a requerimientos:	RF09		
Actor:	Analista		
Propósito:	Permitir la recuperación de un proyecto existente desde un archivo .adu		
Descripción:	El analista abre un proyecto existente, indicando al sistema la ubicación y el nombre del archivo; cuando el analista acepta la operación, el sistema carga el proyecto y lo muestra en pantalla		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un proyecto creado previamente con extensión .adu		
Post condiciones:	• Cerrar el proyecto • Editar el proyecto • Eliminar el proyecto • Guardar cambios en el proyecto		

Curso Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Selecciona la opción [Abrir Proyecto] en el menú [Inicio], o bien, en el panel de bienvenida de la pantalla [P001: MainAledanUML]	**2.** Presenta la pantalla [P002: AsistenteProyecto]
3. Indica la ubicación del proyecto y selecciona el archivo que desea abrir	
4. Selecciona la opción [Abrir] de la	**5.** Verifica si el proyecto está cerrado

pantalla [P002: AsistenteProyecto]	
	6. Agrega el proyecto abierto al panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]
	7. Verifica si el proyecto está en la Lista [Proyectos Recientes]
	8. Agrega el proyecto en la Lista [Proyectos Recientes] del panel de bienvenida ubicado en el [Panel de Edición] de la pantalla [P001: MainAledanUML]
	9. La meta **MT03** finaliza
Curso Alterno de Eventos	
A. Abrir proyecto a partir del menú de la aplicación	
Acción del Actor	**Respuesta del Sistema**
A.1. Hace clic sobre el icono de la aplicación ubicado en la esquina superior izquierda de la pantalla [P001: MainAledanUML]	**A.2.** Despliega el menú de la aplicación
A.3. Hace clic sobre la opción [Abrir Proyecto] del menú desplegado	**A.4.** La meta **MT03** continúa en el **paso 2 del Curso Normal de Eventos**.
B. Abrir proyecto reciente	
Acción del Actor	**Respuesta del Sistema**
B.1. Hace clic sobre la opción [Proyectos Recientes] en el panel de bienvenida de la pantalla [P001: MainAledanUML]	**B.2.** Obtiene la lista de proyectos recientes
	B.3. Presenta la lista de proyectos recientes en el panel de bienvenida la pantalla [P001: MainAledanUML]
B.4. Hace clic derecho sobre el nombre del proyecto que desea abrir en la lista [Proyectos Recientes] de la pantalla [P001: MainAledanUML]	**B.5.** Despliega un submenú de opciones
B.6. Hace clic sobre la opción [Abrir Proyecto] del submenú desplegado	**B.7.** La meta **MT03** continúa en el **paso 5 del Curso Normal de Eventos**.
C. Abrir proyecto reciente a partir del menú de la aplicación	
Acción del Actor	**Respuesta del Sistema**
C.1. Hace clic sobre el icono de la aplicación ubicado en la esquina superior izquierda de la pantalla [P001: MainAledanUML]	**C.2.** Despliega el menú de la aplicación
C.3. Selecciona la opción [Abrir Proyecto] del menú desplegado	**C.4.** Despliega un submenú con la lista de proyectos recientes
C.5. Hace clic sobre el nombre del proyecto que desea abrir en el submenú desplegado	**C.6.** La meta **MT03** continúa en el **paso 5 del Curso Normal de Eventos**.

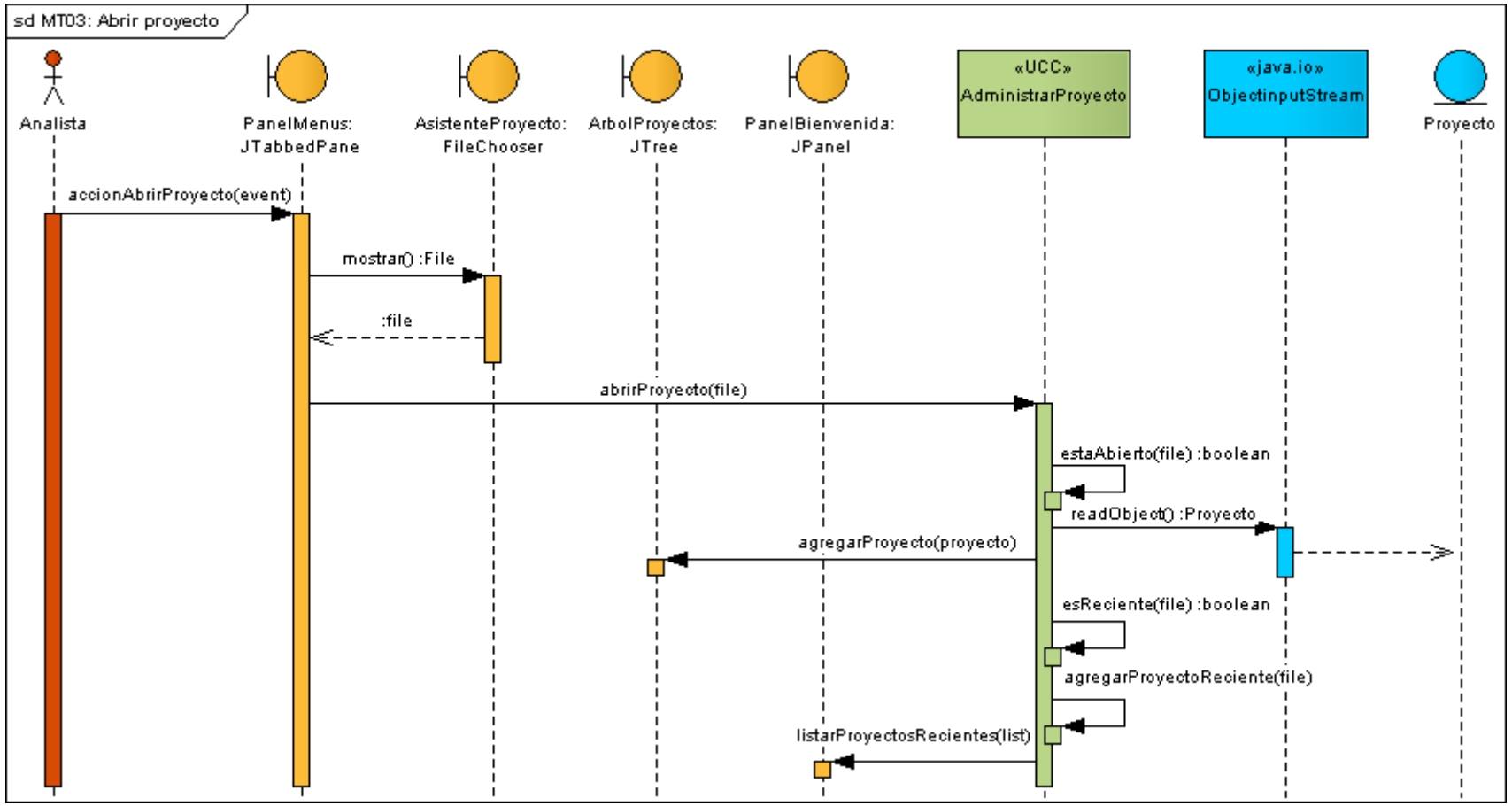


Figura 44: DS Curso normal abrir proyecto

8.2.3.3.1.4. **Meta MT04:** Renombrar proyecto

Tabla 37: Descripción de la Meta Renombrar proyecto

Caso de Uso:	Administrar Proyecto	**Código Caso de Uso:**	CU01
Meta:	Renombrar proyecto	**Código Meta:**	MT04
Tipo:	Primario		
Referencia a requerimientos:	RF01		
Actor:	Analista		
Propósito:	Permitir al analista cambiar el nombre de un proyecto		
Descripción:	El analista decide cambiar el nombre de un determinado proyecto, lo modifica y cuando acepta la operación, el sistema renombra el proyecto		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un proyecto en edición		
Post condiciones:	• Cumplimiento de una o varias metas del CU02 Administrar Diagrama • Guardar un proyecto • Cerrar un proyecto		

Curso Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Hace clic derecho sobre el nombre del proyecto que desea renombrar en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]	**2.** Obtiene el proyecto seleccionado
	3. Despliega un menú de opciones
4. Selecciona la opción [Renombrar] del menú desplegado	**5.** Presenta un mensaje de confirmación
6. Ingresa el nuevo nombre del proyecto en el campo [Nombre del proyecto]	
7. Selecciona el casillero [Renombrar fichero del proyecto]	
8. Presiona la opción [Aceptar] del mensaje de confirmación	**9.** Valida que el nombre ingresado no sea una cadena vacía y que el nombre no exista previamente en la ubicación seleccionada
	10. Renombra el proyecto y su fichero con el nombre asignado
	11. Actualiza el entorno de trabajo
	12. Presenta información de la operación realizada en el panel [Salida] de la pantalla [P001: MainAledanUML]
	13. La meta **MT04** finaliza
Curso Alterno de Eventos	
A. No renombrar fichero	
Acción del Actor	**Respuesta del Sistema**
A.7. Presiona la opción [Aceptar] del mensaje de confirmación	**A.8.** Valida que el nombre ingresado no sea una cadena vacía y que no exista en la ubicación seleccionada
	A.9. Renombra el proyecto con el nombre asignado
	A.10. La meta **MT04** continúa el **paso 11 del Curso Normal de Eventos**

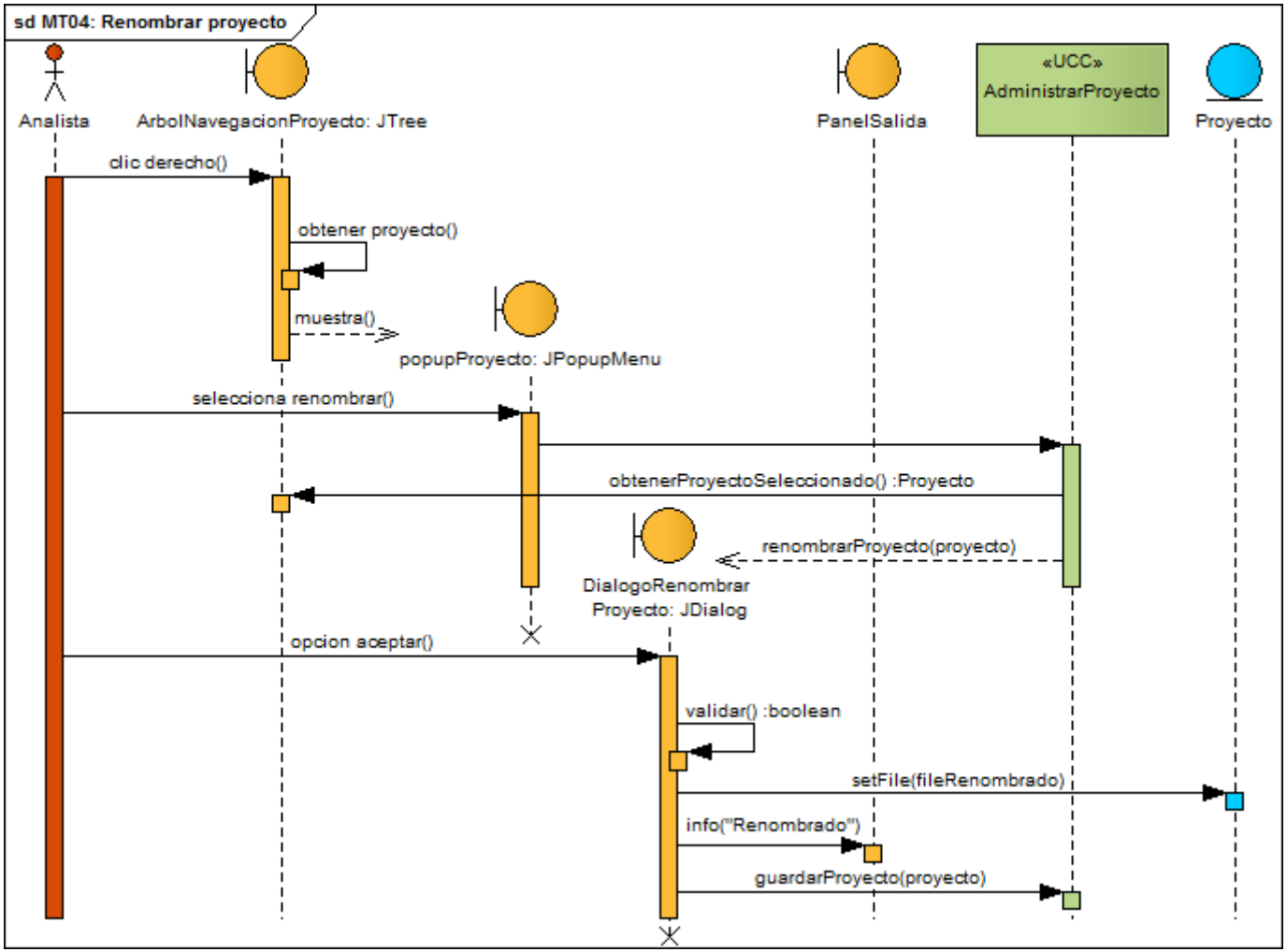


Figura 45: DS Curso normal renombrar proyecto

8.2.3.3.1.5. **Meta MT05:** Administrar objeto de modelo

Tabla 38: Descripción de la Meta Administrar objeto de modelo

Caso de Uso:	Administrar Proyecto	**Código Caso de Uso:**	CU01
Meta:	Administrar objeto de modelo	**Código Meta:**	MT05
Tipo:	Primario		
Referencia a requerimientos:	RF01		
Actor:	Analista		
Propósito:	Permitir al analista la administración (crear, renombrar y eliminar) de los objetos de modelo, mismos que agrupan namespaces, clases, interfaces, enumeraciones y tipos de dato		
Descripción:	El analista selecciona el proyecto sobre el cual quiere crear, renombrar o eliminar un objeto de modelo, y procede a ejecutar la acción deseada		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un proyecto en edición		
Post condiciones:	• Editar un diagrama • Guardar un proyecto • Cerrar un proyecto		

Curso Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Hace clic derecho sobre el nodo [Modelo] o sobre cualquiera de sus elementos, del proyecto en edición en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]	**2.** Obtiene el proyecto seleccionado en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]
	3. Despliega un menú de opciones
4. Selecciona la opción [Nuevo] del menú desplegado	**5.** Despliega un submenú de opciones
6. Selecciona la opción correspondiente al objeto de modelo que desea crear	**7.** Presenta un diálogo de entrada
8. Ingresa el nombre del objeto de modelo en el diálogo de entrada	
9. Presiona la opción [Aceptar] del diálogo de entrada	**10.** Valida que el nombre ingresado no sea una cadena vacía y que el nombre ingresado no exista en ese proyecto
	11. Crea el nuevo objeto de modelo
	12. Muestra el nuevo objeto de modelo en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]
	13. La Meta **MT05** finaliza
A. Creación a partir de un diagrama	
Acción del Actor	**Respuesta del Sistema**
A.1. Los objetos de modelo también podrán ser creados a partir del panel [Paleta] cargado al abrir un diagrama UML. Para ello se deberá dar cumplimiento al **Curso Normal de Eventos** de la meta **MT17 Agregar Artefacto**	
B. Renombrar objeto de modelo	
Acción del Actor	**Respuesta del Sistema**

B.1. Hace clic derecho sobre el nombre del objeto de modelo que desea renombrar en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]	**B.2.** Obtiene el objeto de modelo seleccionado en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]
	B.3. Despliega un menú de opciones
B.4. Selecciona la opción [Renombrar] del menú desplegado	**B.5.** Presenta un diálogo de entrada
B.6. Ingresa el nuevo nombre del objeto de modelo en el campo [Nombre] del diálogo de entrada	
B.7. Presiona la opción [Aceptar] del diálogo de entrada	**B.8.** Valida que el nombre del objeto de modelo no sea una cadena vacía y que el nombre ingresado no exista en ese proyecto
	B.9. Renombra el objeto de modelo con el nombre asignado
	B.10. Actualiza el entorno de trabajo
	B.11. La Meta **MT05** finaliza
C. Eliminar objeto de modelo	
Acción del Actor	**Respuesta del Sistema**
C.1. Hace clic derecho sobre el nombre del objeto de modelo que desea eliminar en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]	**C.2.** Obtiene el objeto de modelo seleccionado
	C.3. Despliega un menú de opciones
C.4. Selecciona la opción [Eliminar] del menú desplegado	**C.5.** Presenta un mensaje de confirmación, en donde se pregunta al analista si desea eliminar el objeto de modelo seleccionado
C.6. Presiona la opción [Sí] del mensaje de confirmación	**C.7.** Elimina el objeto de modelo seleccionado junto a todos sus elementos, si los contiene
	C.8. Actualiza el entorno de trabajo
	C.9. La Meta **MT05** finaliza
D. Objeto de modelo incrustado	
Acción del Actor	**Respuesta del Sistema**
	D.C.5. Presenta un mensaje de aviso indicando al analista que el objeto de modelo está incrustado en algún diagrama y que primero lo debe retirar para proceder a eliminarlo
	D.C.6. La meta **MT05** finaliza

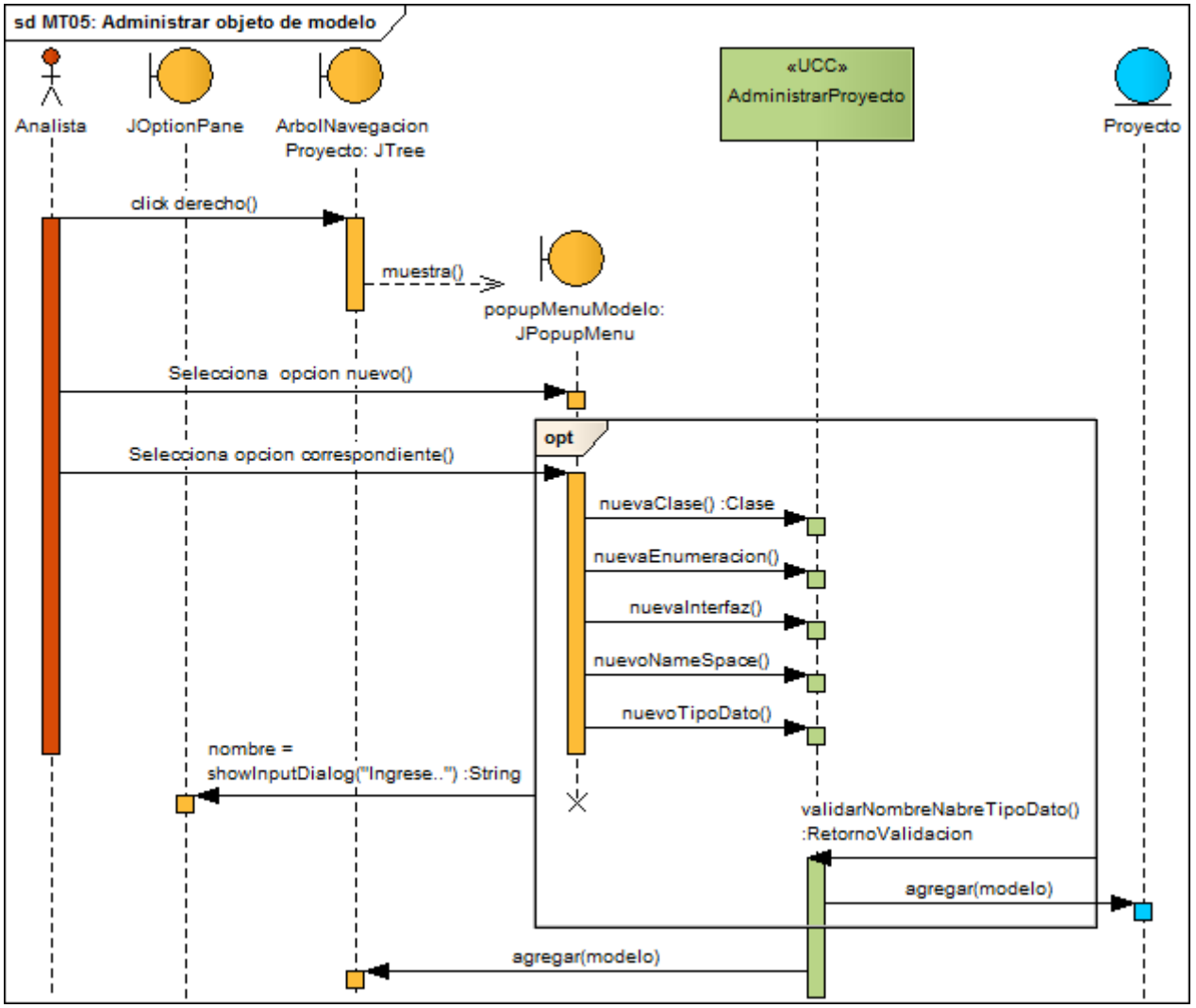


Figura 46: DS Curso normal administrar objeto de modelo

8.2.3.3.1.6. **Meta MT06:** Guardar proyecto

Tabla 39: Descripción de la Meta Guardar proyecto

Caso de Uso:	Administrar Proyecto	**Código Caso de Uso:**	CU01
Meta:	Guardar proyecto	**Código Meta:**	MT06
Tipo:	Primario		
Referencia a requerimientos:	RF01		
Actor:	Analista		
Propósito:	Permitir el almacenamiento de los cambios realizados en un proyecto		
Descripción:	El analista almacena los cambios realizados a un proyecto		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista un proyecto en edición		
Post condiciones:	• Cerrar un proyecto • Editar un proyecto		
Curso Normal de Eventos			
Acción del Actor		**Respuesta del Sistema**	
1. Selecciona la opción [Guardar Proyecto] en el menú [Inicio] de la pantalla [P001: MainAledanUML], o bien, en la barra de tareas de la misma pantalla		**2.** Almacena los cambios realizados dentro del proyecto, en la unidad de almacenamiento	
		3. Actualiza el entorno de trabajo	
		4. Presenta información de la operación realizada en el panel [Salida] de la pantalla [P001: MainAledanUML]	
		5. La meta **MT06** finaliza	
Curso Alterno de Eventos			
A. Guardar proyecto a partir del menú de la aplicación			
Acción del Actor		**Respuesta del Sistema**	
A.1. Hace clic sobre el icono de la aplicación ubicado en la esquina superior izquierda de la pantalla [P001: MainAledanUML]		**A.2.** Despliega el menú de la aplicación	
A.3. Selecciona la opción [Guardar] del menú desplegado		**A.4.** Despliega un submenú de opciones	
A.5. Hace clic sobre la opción [Guardar] del submenú desplegado		**A.6.** La meta **MT06** continúa en el **paso 2 del Curso Normal de Eventos**.	

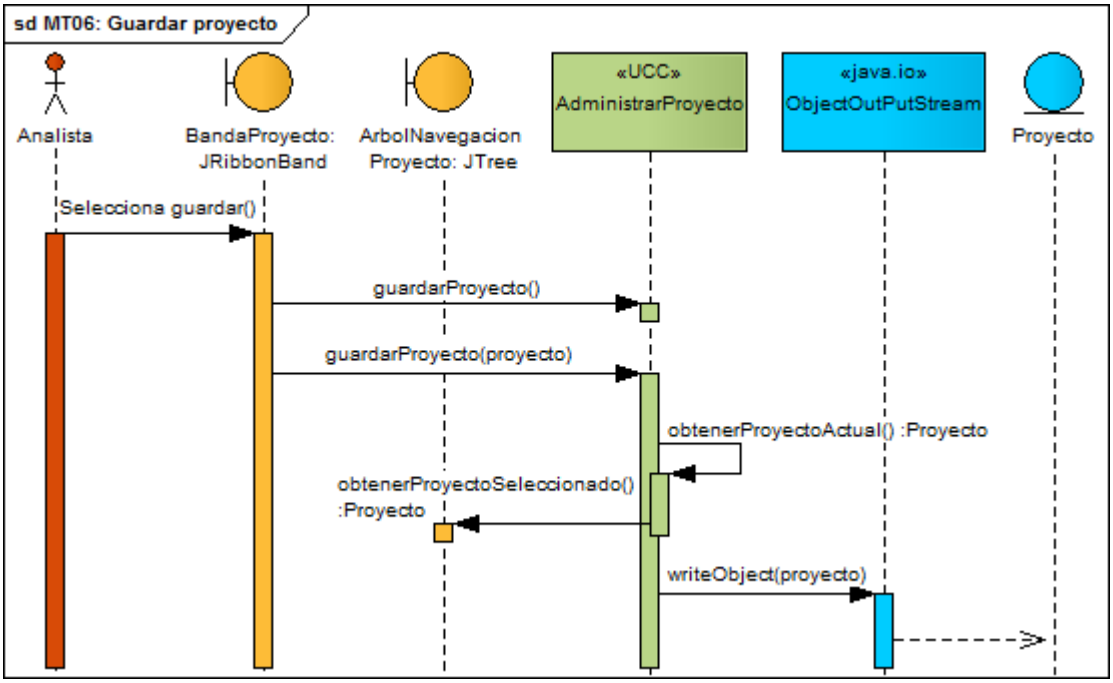


Figura 47: DS Curso normal guardar proyecto

8.2.3.3.1.7. **Meta MT07:** Guardar proyecto como

Tabla 40: Prototipo de la Pantalla AsistenteProyecto – Guardar como

Nombre de la Pantalla:	AsistenteProyecto	**Código:**	P002
Tipo de Interfaz Gráfica:	JFileChooser		
Caso:	Guardar proyecto como		

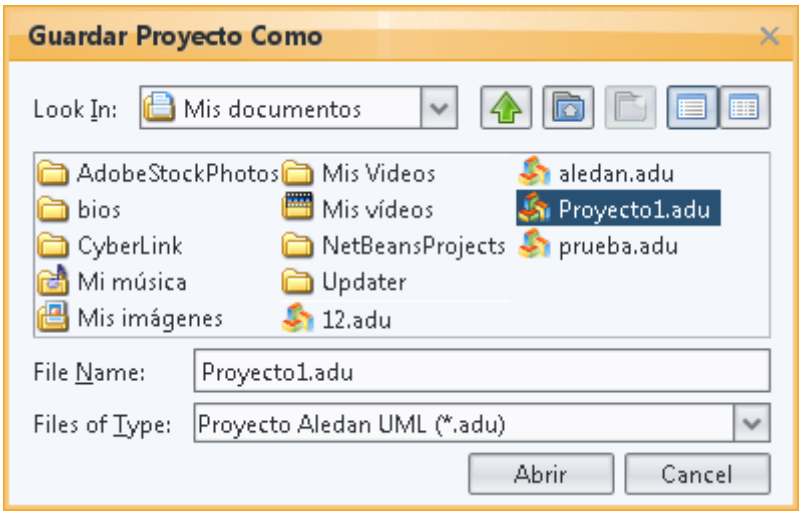


Tabla 41: Descripción de la Meta Guardar proyecto como

Caso de Uso:	Administrar Proyecto	**Código Caso de Uso:**	CU01
Meta:	Guardar proyecto como	**Código Meta:**	MT07
Tipo:	Primario		
Referencia a requerimientos:	RF01		
Actor:	Analista		
Propósito:	Permitir guardar una copia del proyecto actual con otro nombre, con los fines que el usuario crea convenientes		
Descripción:	El analista guarda el proyecto seleccionado con otro nombre, mientras continúa trabando sobre el proyecto actual		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un proyecto abierto		
Post condiciones:	• Cerrar un proyecto • Editar un proyecto		

Curso Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Hace clic sobre el icono de la aplicación ubicado en la esquina superior izquierda de la pantalla [P001: MainAledanUML]	**2.** Despliega el menú de la aplicación
3. Selecciona la opción [Guardar] del menú desplegado	**4.** Despliega un submenú de opciones
5. Selecciona la opción [Guardar como] del submenú desplegado	**6.** Presenta la pantalla [P002: AsistenteProyecto]
7. Selecciona la ubicación para el nuevo proyecto	

8. Ingresa el nuevo nombre del proyecto	
9. Presiona la opción [Guardar] de la pantalla [P002: AsistenteProyecto]	**10.** Valida que el campo [Nombre de archivo] no esté vacío y que el nombre del proyecto no exista en la ubicación seleccionada
	11. Asigna el nuevo archivo al proyecto en cuestión
	12. Guarda el proyecto en la unidad de almacenamiento
	13. Actualiza el entorno de trabajo
	14. Presenta información de la operación realizada en el panel [Salida] de la pantalla [P001: MainAledanUML]
	15. La Meta **MT07** finaliza
Curso Alterno de Eventos	
A. Reemplazar	
Acción del Actor	**Respuesta del Sistema**
	A.11. Presenta un mensaje de aviso indicando al analista que el proyecto ya existe y preguntando si desea reemplazarlo
A.12. Si el analista confirma el mensaje de aviso, la meta **MT07** continúa en el paso **A.13 del Curso Alterno de Eventos Reemplazar**. Caso contrario, la creación se cancela y la meta **MT07** finaliza	**A.13.** El sistema sobrescribe el proyecto y la meta **MT07** continúa en el **paso 11 del Curso Normal de Eventos**.

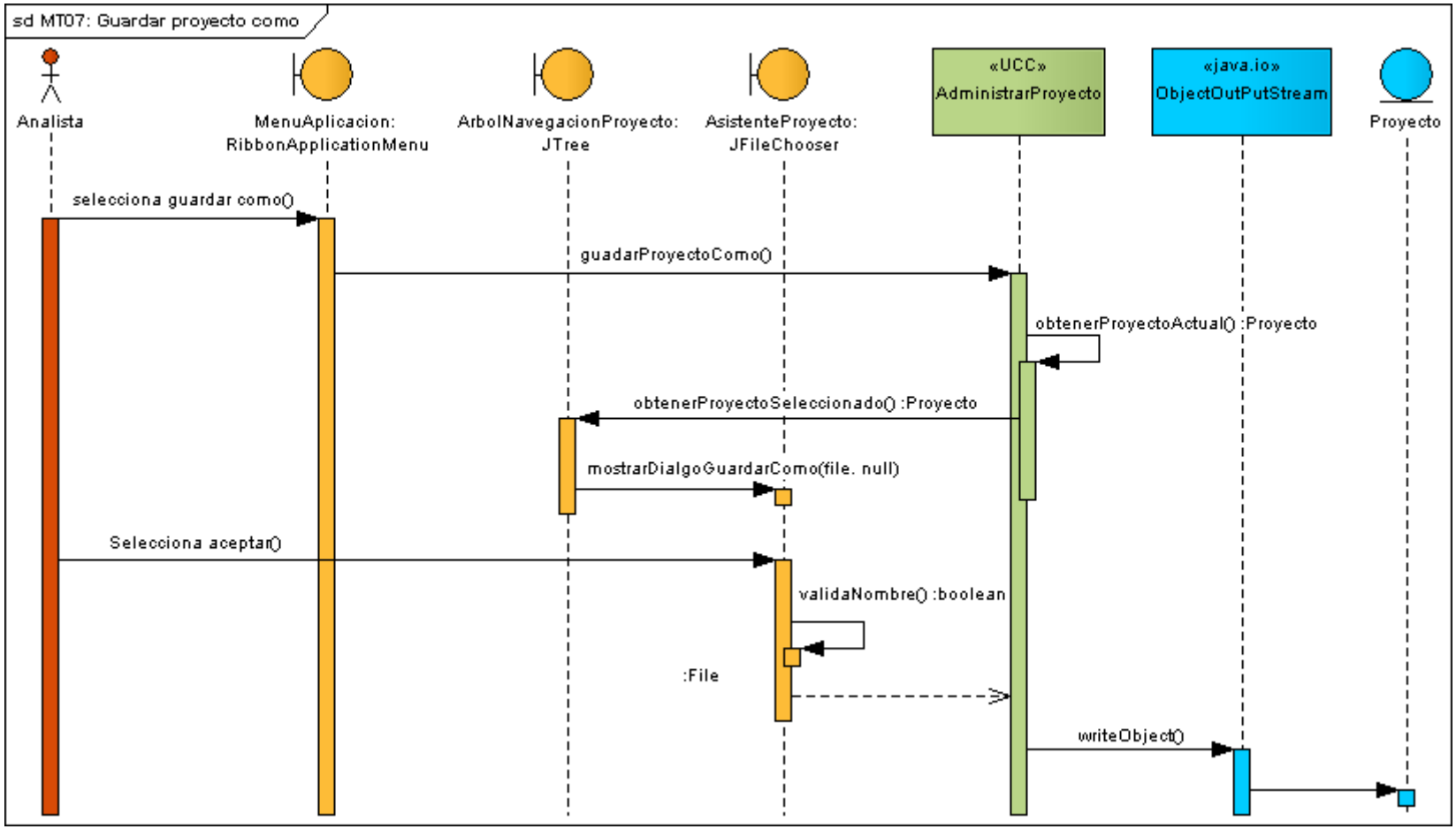


Figura 48: DS Curso normal guardar proyecto como

8.2.3.3.1.8. **Meta MT08:** Cerrar proyecto

Tabla 42: Descripción de la Meta Cerrar proyecto

Caso de Uso:	Administrar Proyecto	**Código Caso de Uso:**	CU01
Meta:	Cerrar proyecto	**Código Meta:**	MT08
Tipo:	Primario		
Referencia a requerimientos:	RF01		
Actor:	Analista		
Propósito:	Permitir al analista cerrar un proyecto abierto en la aplicación, para evitar modificaciones que lo afecten negativamente		
Descripción:	El analista selecciona el proyecto que desea cerrar y procede a ejecutar dicha acción		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un proyecto abierto		
Curso Normal de Eventos			
Acción del Actor		**Respuesta del Sistema**	
1. Hace clic derecho sobre el nombre del proyecto que desea cerrar en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]		**2.** Obtiene el proyecto seleccionado	
		3. Despliega un menú de opciones	
4. Escoge la opción [Cerrar] en el menú desplegado		**5.** Valida si han existido cambios dentro del proyecto	
		6. Presenta un mensaje de confirmación preguntando al analista si desea guardar los cambios realizados	
7. Elige la opción [Sí] del mensaje de confirmación		**8.** Guarda las modificaciones realizadas al proyecto	
		9. Cierra todos los componentes del proyecto que se encuentren abiertos	
		10. Retira el proyecto del panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]	
		11. Presenta información de la operación realizada en el panel [Salida] de la pantalla [P001: MainAledanUML]	
		12. La meta **MT08** finaliza	
Curso Alterno de Eventos			
A. Cerrar a partir del menú [Inicio]			
Acción del Actor		**Respuesta del Sistema**	
A.1. Selecciona el proyecto que desea cerrar en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]			
A.2. Selecciona la opción [Cerrar Proyecto] en el menú [Inicio] de la pantalla [P001: MainAledanUML]		**A.3.** La meta **MT08** continúa en el **paso 5 del Curso Normal de Eventos**.	
B. No guardar cambios			
B.8. Presiona la opción [No] del mensaje de confirmación		**B.9.** La meta **MT08** continúa en el **paso 9 del Curso Normal de Eventos**	

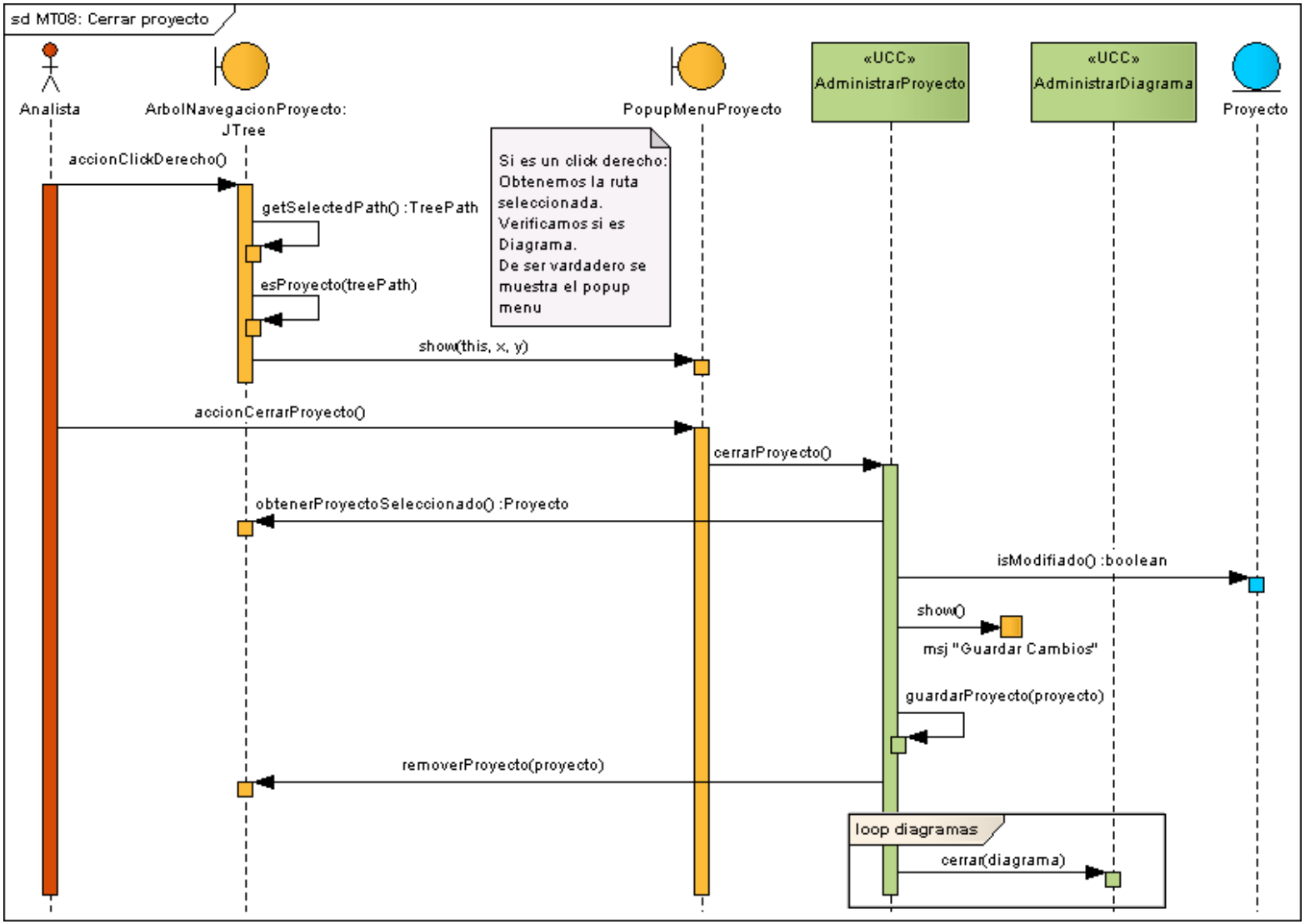


Figura 49: DS Curso normal cerrar proyecto

8.2.3.3.1.9. **Meta MT09:** Eliminar proyecto

Tabla 43: Descripción de la Meta Eliminar proyecto

Caso de Uso:	Administrar Proyecto	**Código Caso de Uso:**	CU01
Meta:	Eliminar proyecto	**Código Meta:**	MT09
Tipo:	Primario		
Referencia a requerimientos:	RF01		
Actor:	Analista		
Propósito:	Permitir la eliminación de un proyecto		
Descripción:	El analista selecciona el proyecto que desea eliminar y procede a ejecutar dicha acción; el sistema solicita la confirmación para eliminar el proyecto y si el analista la ratifica, el sistema lo elimina		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un proyecto abierto		
Curso Normal de Eventos			
Acción del Actor		**Respuesta del Sistema**	
1. Hace clic derecho sobre el nombre del proyecto que desea eliminar en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]		**2.** Obtiene el proyecto seleccionado	
		3. Despliega un menú de opciones	
4. Selecciona la opción [Eliminar] del menú desplegado		**5.** Presenta un mensaje de confirmación, en donde se pregunta al analista si desea eliminar el proyecto seleccionado del disco duro	
6. Selecciona el casillero [Eliminar también del disco] del mensaje de confirmación			
7. Selecciona la opción [Aceptar] del mensaje de confirmación		**8.** Cierra todos los componentes del proyecto que se encuentren abiertos	
		9. Retira el proyecto del panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]	
		10. Elimina el proyecto seleccionado del disco duro	
		11. Retira el proyecto de la lista [Proyectos Recientes]	
		12. La Meta **MT09** finaliza	
Curso Alterno de Eventos			
A. Eliminación del entorno de trabajo			
Acción del Actor		**Respuesta del Sistema**	
A.6. Selecciona la opción [Aceptar] del mensaje de confirmación		**A.7.** Cierra todos los componentes del proyecto que se encuentren abiertos	
		A.8. Retira el proyecto del panel [Navegador de Proyectos] y de la lista [Proyectos Recientes] de la pantalla [P001: MainAledanUML]	
		A.9. La meta **MT09** finaliza	

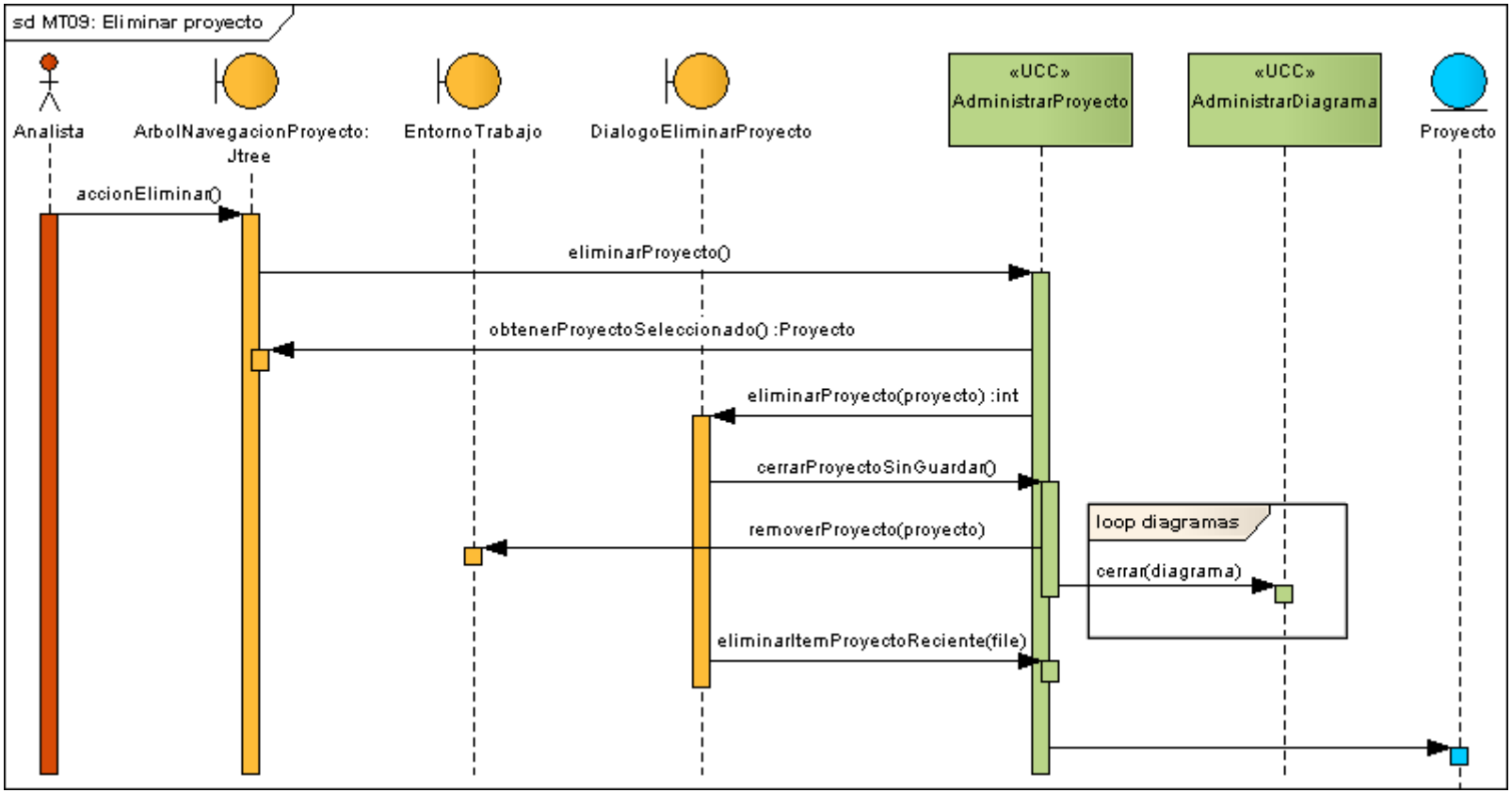


Figura 50: DS Curso normal eliminar proyecto

8.2.3.3.2. **Caso de Uso:** Administrar Diagrama

8.2.3.3.2.1. **Meta MT10:** Crear diagrama

Tabla 44: Prototipo de la Pantalla AsistenteNuevoDiagrama

Nombre de la Pantalla:	AsistenteNuevoDiagrama	**Código:**	P003
Tipo de Interfaz Gráfica:	JDialog		

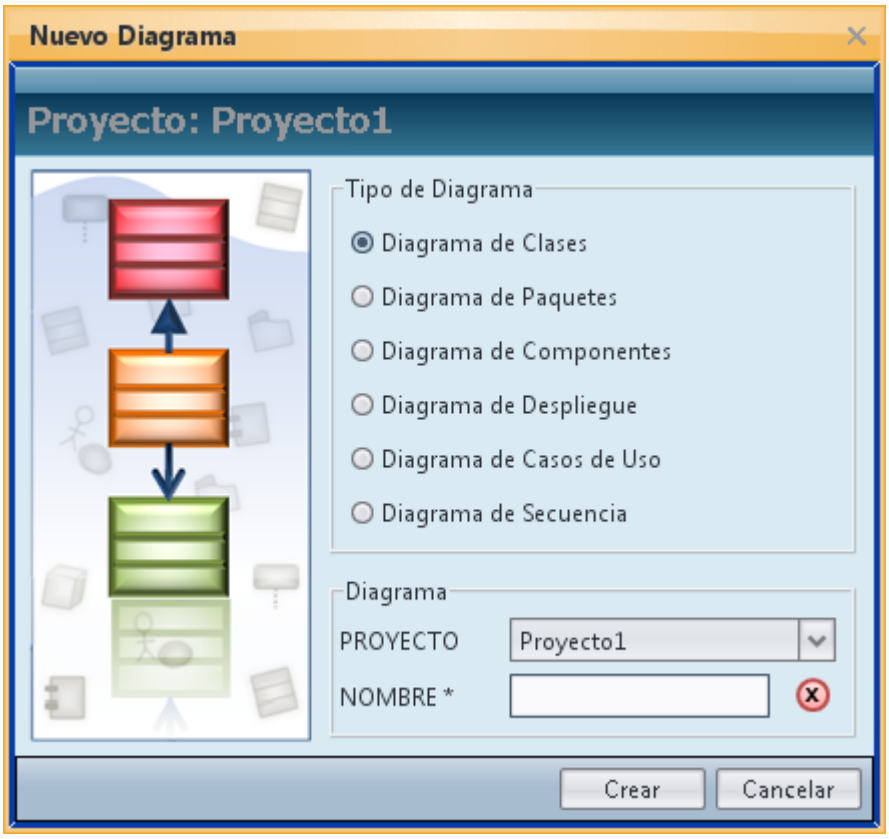


Tabla 45: Descripción de la Meta Crear diagrama

Caso de Uso:	Administrar Diagrama	**Código Caso de Uso:**	CU02
Meta:	Crear diagrama	**Código Meta:**	MT10
Tipo:	Primario		
Referencia a requerimientos:	RF03, RF04, RF05, RF06, RF07, RF08		
Actor:	Analista		
Propósito:	Permitir al analista la creación de un nuevo diagrama (clases, paquetes, comportamiento, despliegue, casos de uso y secuencia)		
Descripción:	El analista crea un nuevo diagrama, indicando su tipo (clases, paquetes, comportamiento, despliegue, casos de uso o secuencia) y el proyecto dentro del que será creado; el sistema actualiza el entorno de trabajo y muestra una nueva pestaña para la edición del diagrama		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un proyecto abierto seleccionado		
Post condiciones:	• Editar un diagrama • Cerrar un diagrama • Exportar un diagrama en formato *.png • Imprimir un diagrama • Cumplimiento de una o varias metas del CU03 Administrar Artefacto • Guardar un proyecto		
Curso Normal de Eventos			
Acción del Actor		**Respuesta del Sistema**	
1. Hace clic derecho sobre el nodo [Diagrama] o sobre cualquiera de sus elementos, en el proyecto en el que se va a crear el diagrama, en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]		**2.** Obtiene el proyecto seleccionado	
		3. Despliega un menú de opciones	
4. Selecciona la opción [Nuevo Diagrama] del menú desplegado		**5.** Presenta la pantalla [P003: AsistenteNuevoDiagrama]	
6. Selecciona el tipo de diagrama que desea crear			
7. Ingresa el nombre del nuevo diagrama en el campo de texto [Nombre*] de la pantalla [P003: AsistenteNuevoDiagrama]			
8. Presiona la opción [Crear] de la pantalla [P003: AsistenteNuevoDiagrama]		**9.** Valida que el campo de texto [Nombre*] no esté vacío y que el nombre ingresado no exista en el proyecto en edición	
		10. Crea el nuevo diagrama de acuerdo a las especificaciones dadas por el analista	
		11. Muestra el nuevo diagrama en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]	
		12. Carga una nueva pestaña en el panel [Panel de Edición] de la pantalla [P001: MainAledanUML] para la edición del nuevo diagrama	
		13. Carga una paleta con los artefactos utilizados de acuerdo al tipo de	

	diagrama creado, en el panel [Paleta] de la pantalla [P001: MainAledanUML]
	14. La Meta **MT10** finaliza
Curso Alterno de Eventos	
A. Creación a partir del menú [Inicio]	
Acción del Actor	**Respuesta del Sistema**
A.1. Selecciona la opción [Nuevo Diagrama] en el menú [Inicio] de la pantalla [P001: MainAledanUML]	**A.2.** Obtiene el proyecto en edición del panel [Navegador de Proyectos] en la pantalla [P001: MainAledanUML]
	A.3. La meta **MT10** continúa en el **paso 5 del Curso Normal de Eventos**
B. Creación directa a partir del menú [Inicio]	
Acción del Actor	**Respuesta del Sistema**
B.1. Hace clic sobre el icono correspondiente al tipo de diagrama que desea crear en el menú [Inicio] de la pantalla [P001: MainAledanUML]	**B.2.** Obtiene el tipo de diagrama de acuerdo a la especificación del analista
	B.3. Asigna un nombre por defecto al nuevo diagrama
	B.4. La meta **MT10** continúa en el **paso 10 del Curso Normal de Eventos**
C. Diagrama existente	
Acción del Actor	**Respuesta del Sistema**
	C.10. Presenta un mensaje de aviso indicando al analista que el diagrama ya existe y que debe cambiar el nombre
	C.11. La meta **MT10** continúa en el **paso 7 del Curso Normal de Eventos**
D. Crear diagrama a partir del menú de la aplicación	
Acción del Actor	**Respuesta del Sistema**
D.1. Hace clic sobre el icono de la aplicación ubicado en la esquina superior izquierda de la pantalla [P001: MainAledanUML]	**D.2.** Despliega el menú de la aplicación
D.3. Selecciona la opción [Nuevo] del menú desplegado	**D.4.** Despliega un submenú de opciones
D.5. Hace clic sobre la opción [Diagrama] del submenú desplegado	**D.6.** La meta **MT10** continúa en el **paso 5 del Curso Normal de Eventos**.

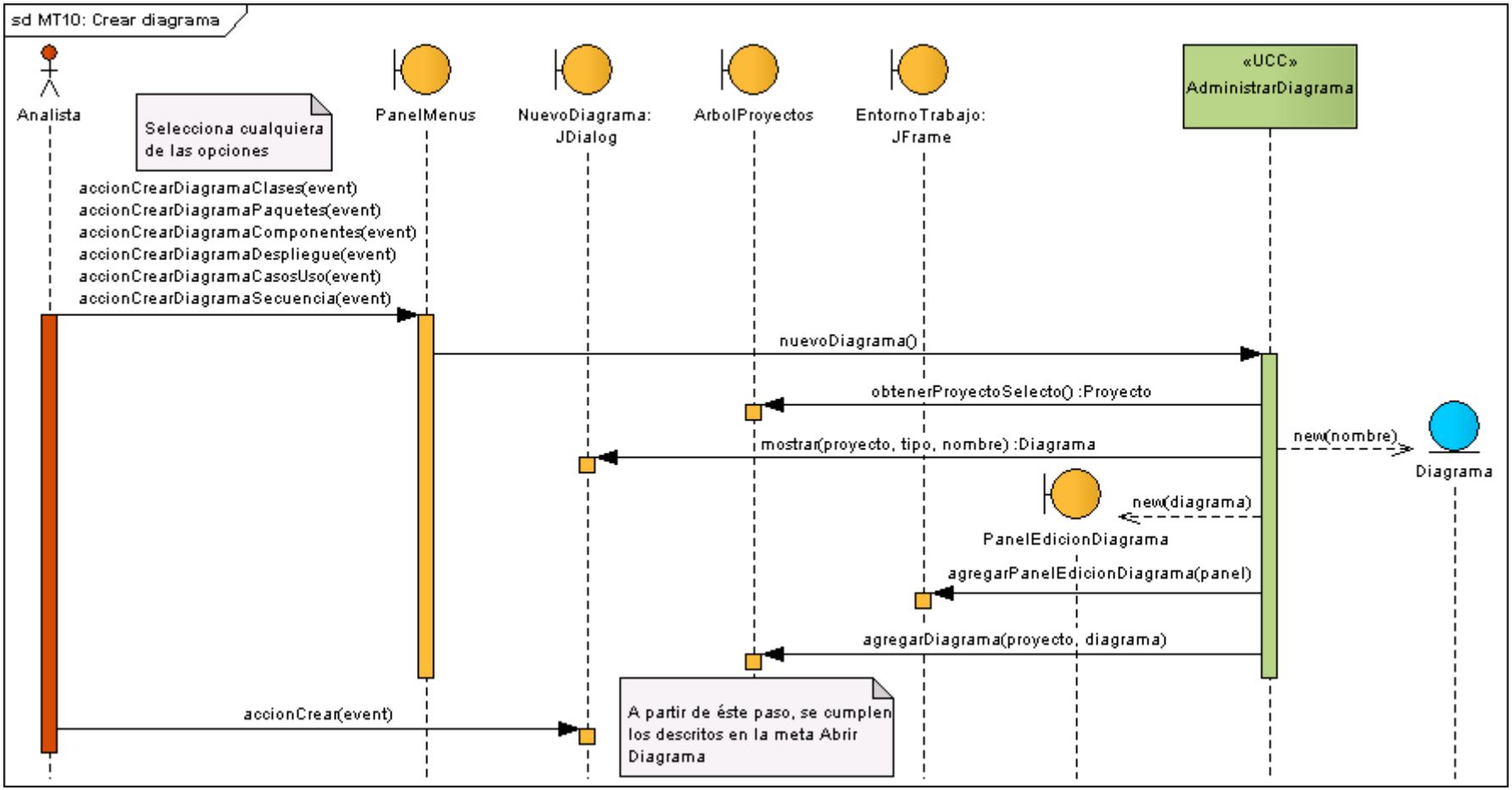


Figura 51: DS Curso normal crear diagrama

8.2.3.3.2.2. **Meta MT11:** Imprimir diagrama

Tabla 46: Prototipo de la Pantalla AsistenteImpresionDiagrama

Nombre de la Pantalla:	AsistenteImpresionDiagrama	**Código:**	P005
Tipo de Interfaz Gráfica:	JDialog		

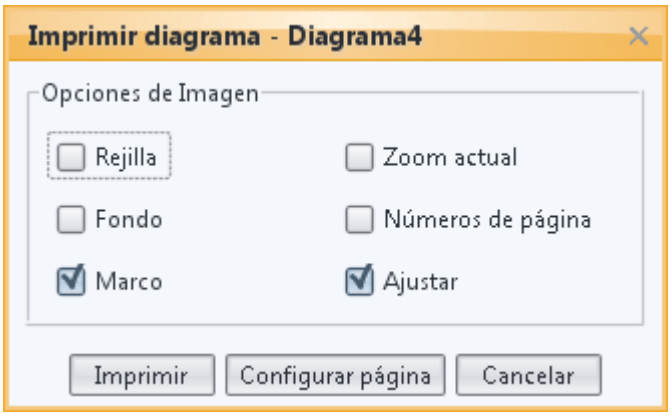


Tabla 47: Descripción de la Meta Imprimir diagrama

Caso de Uso:	Administrar Diagrama	**Código Caso de Uso:**	CU02
Meta:	Imprimir diagrama	**Código Meta:**	MT11
Tipo:	Primario		
Referencia a requerimientos:	RF03, RF04, RF05, RF06, RF07, RF08		
Actor:	Analista		
Propósito:	Permitir al analista la impresión de un diagrama		
Descripción:	El analista selecciona el diagrama que desea imprimir y ejecuta dicha acción; el sistema solicita al analista las preferencias de impresión, y, luego de que éste acepta la operación, el sistema procede a imprimir el diagrama seleccionado		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un diagrama seleccionado o en edición		

Curso Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Selecciona la pestaña del diagrama que desea imprimir en el panel [Panel de Edición] de la pantalla [P001: MainAledanUML]	
2. Presiona la opción [Imprimir] en el menú [Inicio] de la pantalla [P001: MainAledanUML]	**3.** Obtiene diagrama seleccionado en el panel [Panel de Edición] de la pantalla [P001: MainAledanUML]
	4. Presenta la pantalla [P005: AsistenteImpresionDiagrama]
5. Selecciona los casilleros correspondientes a sus preferencias de impresión en el panel [Opciones de imagen]	
6. Presiona la opción [Configurar página] de la pantalla [P005: AsistenteImpresionDiagrama]	**7.** Presenta la pantalla [P007: Configurar página]

8. Define los atributos de página	
9. Selecciona la opción [Aceptar] de la pantalla [P007: Configurar página]	
10. Presiona la opción [Imprimir] de la pantalla [P005: AsistenteImpresionDiagrama]	**11.** Inicia el servicio de impresión
	12. La Meta **MT11** finaliza
Curso Alterno de Eventos	
A. Diagrama no seleccionado	
Acción del Actor	**Respuesta del Sistema**
	A.4. Presenta un mensaje de aviso indicando al usuario que no se ha seleccionado un diagrama para su impresión
	A.5. La meta **MT11** finaliza

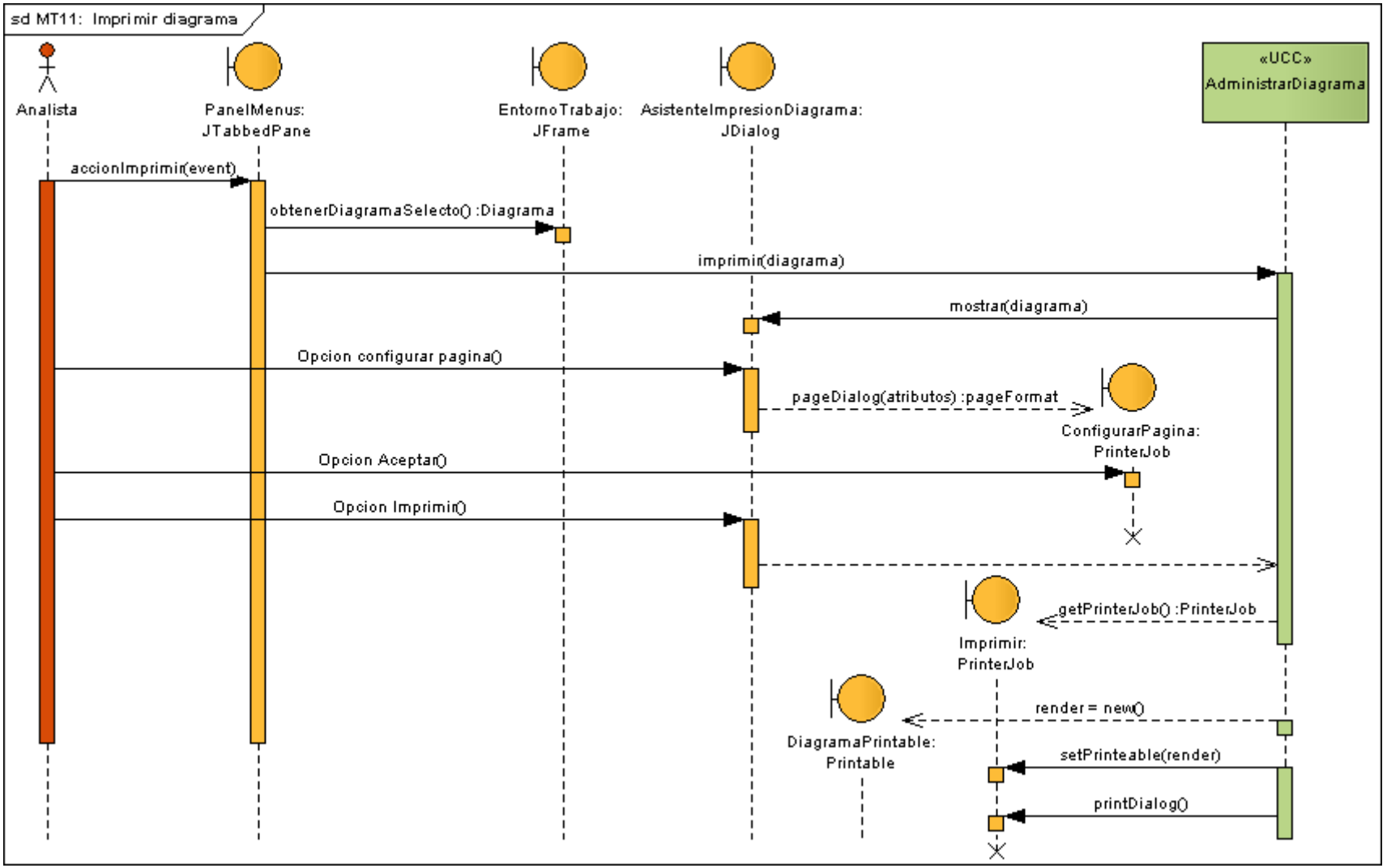


Figura 52: DS Curso normal imprimir diagrama

8.2.3.3.2.3. **Meta MT12:** Abrir diagrama

Tabla 48: Descripción de la Meta Abrir diagrama

Caso de Uso:	Administrar Diagrama	**Código Caso de Uso:**	CU02
Meta:	Abrir diagrama	**Código Meta:**	MT12
Tipo:	Primario		
Referencia a requerimientos:	RF03, RF04, RF05, RF06, RF07, RF08		
Actor:	Analista		
Propósito:	Permitir la recuperación de un diagrama creado previamente dentro de un proyecto		
Descripción:	El analista abre un diagrama existente, el sistema lo carga y lo muestra en la pantalla principal		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un diagrama creado previamente dentro de un proyecto		
Post condiciones:	• Cerrar el diagrama • Editar el diagrama • Guardar cambios en el proyecto • Exportar el diagrama en formato PNG • Imprimir el diagrama		

Curso Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Hace clic derecho sobre el nombre del diagrama que desea abrir, en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]	**2.** Obtiene el diagrama seleccionado
	3. Despliega un menú de opciones
4. Selecciona la opción [Abrir] del menú desplegado	**5.** Carga todos los componentes del diagrama seleccionado y los presenta en una nueva pestaña en el panel [Panel de Edición] de la pantalla [P001: MainAledanUML]
	6. Carga una paleta con los artefactos utilizados de acuerdo al tipo de diagrama que se ha abierto, en el panel [Paleta] de la pantalla [P001: MainAledanUML]
	7. La meta **MT12** finaliza
Curso Alterno de Eventos	
A. Abrir diagrama a partir del panel [Navegador de Proyectos]	
Acción del Actor	**Respuesta del Sistema**
A.1. Hace doble clic sobre el diagrama que desea abrir, en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]	**A.2.** Obtiene el diagrama seleccionado
	A.3. La meta **MT12** continúa en el **paso 5 del Curso Normal de Eventos**

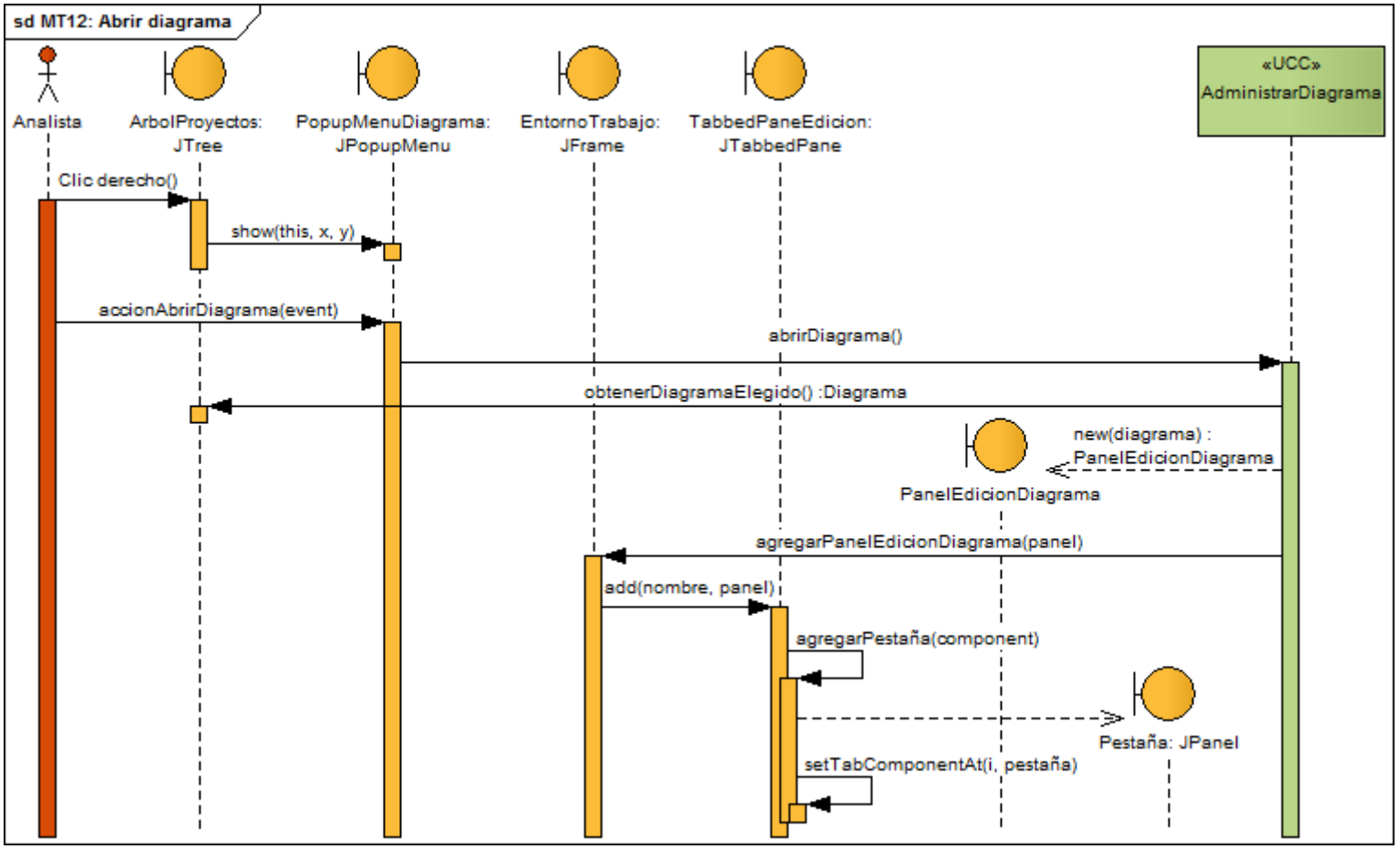


Figura 53: DS Curso normal abrir diagrama

8.2.3.3.2.4. **Meta MT13:** Editar diagrama

Tabla 49: Descripción de la Meta Editar diagrama

Caso de Uso:	Administrar Diagrama	**Código Caso de Uso:**	CU02
Meta:	Editar diagrama	**Código Meta:**	MT13
Tipo:	Primario		
Referencia a requerimientos:	RF03, RF04, RF05, RF06, RF07, RF08		
Actor:	Analista		
Propósito:	Permitir al analista la edición de un diagrama de clases, paquetes, comportamiento, despliegue, casos de uso o de secuencia creado dentro de un proyecto		
Descripción:	El analista selecciona un diagrama creado dentro de un proyecto para la correspondiente edición en cualquiera de sus propiedades y procede a modificarlo		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un diagrama en edición		
Post condiciones:	• Cerrar un diagrama • Exportar un diagrama en formato PNG • Imprimir un diagrama • Guardar un proyecto		
Curso Normal de Eventos			
Acción del Actor		**Respuesta del Sistema**	
1. Cumplimiento de una o varias metas del **CU03 Administrar Artefacto**			
Curso Alterno de Eventos			
A. Editar Propiedades			
Acción del Actor		**Respuesta del Sistema**	
A.1. Selecciona el diagrama que desea editar en el panel [Panel de Edición] de la pantalla [P001: MainAledanUML]		**A.2.** Obtiene el diagrama seleccionado	
		A.3. Carga las propiedades del diagrama seleccionado y las muestra en el panel [Propiedades] de la pantalla [P001: MainAledanUML]	
A.4. Modifica los valores en una o varias propiedades del diagrama en el panel [Propiedades] de la pantalla [P001: MainAledanUML]		**A.5.** Verifica que los valores modificados por el analista estén dentro del rango permitido y que sean válidos	
		A.6. Actualiza los valores de las propiedades del diagrama	
		A.7. La meta **MT13** finaliza	
B. Renombrar diagrama			
Acción del Actor		**Respuesta del Sistema**	
B.1. Hace clic derecho sobre el nombre del diagrama que desea renombrar en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]		**B.2.** Obtiene el diagrama seleccionado	
		B.3. Despliega un menú de opciones	
B.4. Selecciona la opción [Renombrar] del menú desplegado		**B.5.** Presenta un diálogo de entrada	
B.6. Ingresa el nuevo nombre del diagrama en el campo [Nombre] del diálogo de			

entrada	
B.7. Presiona la opción [Aceptar] del diálogo de entrada	**B.8.** Valida que el campo [Nombre] del diálogo de entrada no esté vacío y que no exista previamente en la ubicación seleccionada
	B.9. Renombra el diagrama con el nombre asignado
	B.10. Actualiza el entorno de trabajo
	B.11. La meta **MT13** finaliza
C. Zoom In y Zoom Out	
Acción del Actor	**Respuesta del Sistema**
C.1. Manipula la barra de Zoom que se encuentra en el menú [Ver] de la pantalla [P001: MainAledanUML]	**C.2.** Obtiene el diagrama en edición
	C.3. Aplica el valor Zoom al diagrama en edición
	C.4. La Meta **MT13** finaliza

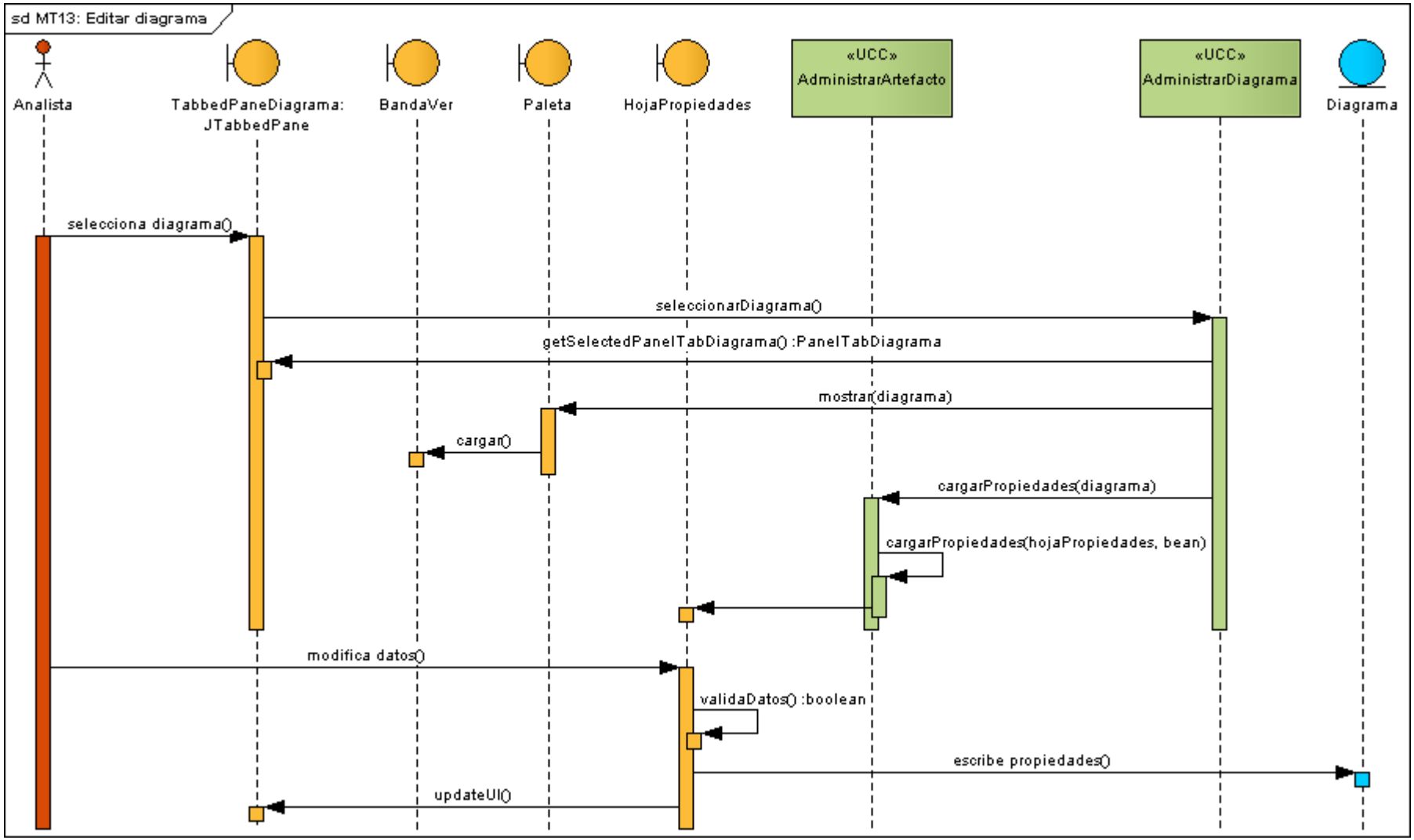


Figura 54: DS Curso normal editar diagrama

8.2.3.3.2.5. **Meta MT14:** Exportar diagrama

Tabla 50: Prototipo de la Pantalla AsistenteExportarPng

Nombre de la Pantalla:	AsistenteExportarPng	**Código:**	P004
Tipo de Interfaz Gráfica:	JFileChooser		

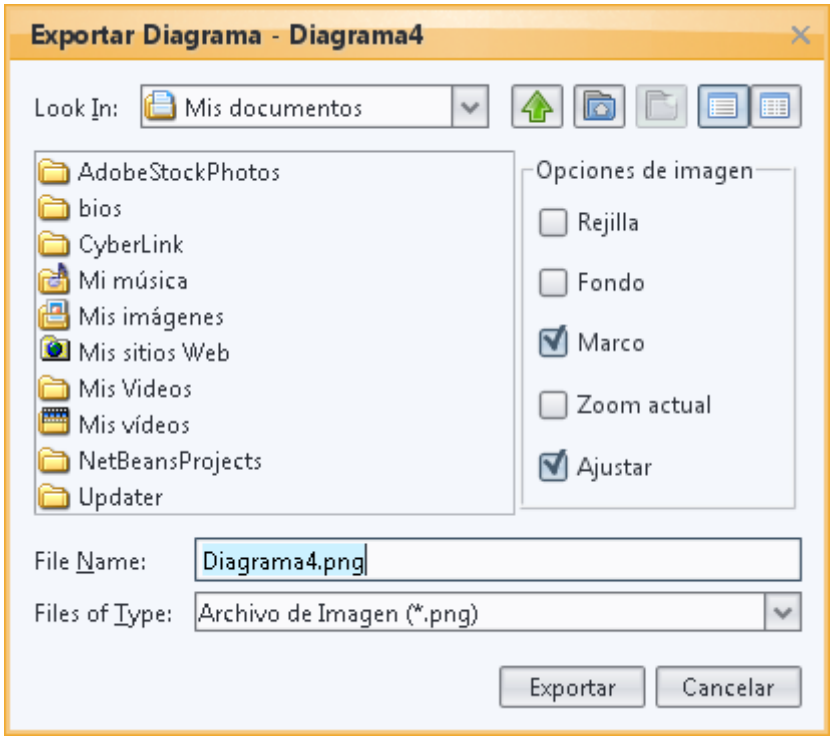


Tabla 51: Descripción de la Meta Exportar diagrama en formato *.png

Caso de Uso:	Administrar Diagrama	**Código Caso de Uso:**	CU02
Meta:	Exportar Diagrama en formato *.png	**Código Meta:**	MT14
Tipo:	Primario		
Referencia a requerimientos:	RF03, RF04, RF05, RF06, RF07, RF08		
Actor:	Analista		
Propósito:	Permitir al analista la exportación de un diagrama en el formato gráfico *.png		
Descripción:	El analista selecciona el diagrama que desea exportar y ejecuta dicha acción; el sistema crea un nuevo archivo con extensión *.png, mismo que contiene el diagrama seleccionado por el analista		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un diagrama seleccionado		

Curso Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Presiona la opción [Exportar] en el menú [Inicio] de la pantalla [P001: MainAledanUML]	**2.** Obtiene el diagrama que se encuentra en edición
	3. Asigna el nombre por defecto al archivo *.png

	4. Presenta la pantalla [P004: AsistenteExportarPng]
5. Elige la ubicación del nuevo archivo	
6. Ingresa el nombre del nuevo archivo, si así lo prefiere	
7. Define las características deseadas de imagen, en el panel [Opciones de imagen]	
8. Presiona la opción [Exportar]	**9.** Valida que el campo [Nombre de archivo] de la pantalla [P004: AsistenteExportarPng] no esté vacío y que el nombre del nuevo archivo no exista previamente
	10. Crea el archivo *.png a partir del diagrama *.adu, con el nombre, ubicación y características asignadas
	11. La Meta **MT14** finaliza
Curso Alterno de Eventos	
A. No existe un Diagrama en edición	
Acción del Actor	**Respuesta del Sistema**
	A.3. Presenta un mensaje de aviso notificando al analista que no se ha seleccionado un diagrama
	A.4. La meta **MT14** finaliza
B. Reemplazar	
Acción del Actor	**Respuesta del Sistema**
	B.9. Presenta un mensaje de aviso indicando al analista que el archivo ya existe y preguntando si desea reemplazarlo
B.10. Si el analista confirma el mensaje de aviso, la meta continúa en el **paso C.11 del Curso Alterno de Eventos Reemplazar.** Caso contrario, la exportación se cancela y la meta **MT14** finaliza	**B.11.** El sistema sobrescribe el archivo
	B.12. La meta **MT14** continúa en el **paso 8 de Curso Normal de Eventos**

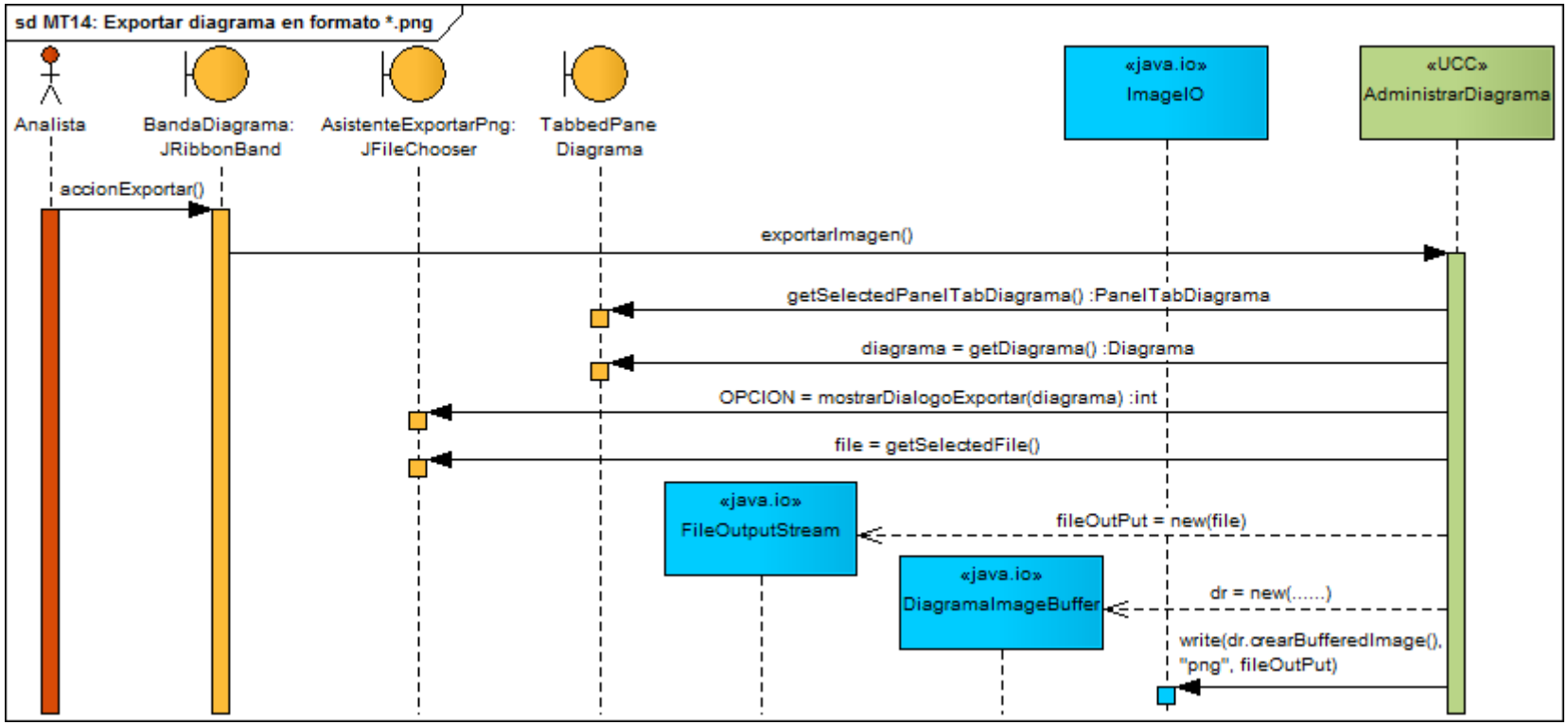


Figura 55: DS Curso normal exportar diagrama en formato *.png

8.2.3.3.2.6. **Meta MT15:** Cerrar diagrama

Tabla 52: Descripción de la Meta Cerrar diagrama

Caso de Uso:	Administrar Diagrama	**Código Caso de Uso:**	CU02
Meta:	Cerrar diagrama	**Código Meta:**	MT15
Tipo:	Primario		
Referencia a requerimientos:	RF03, RF04, RF05, RF06, RF07, RF08		
Actor:	Analista		
Propósito:	Permitir al analista cerrar un diagrama abierto para liberar recursos de la aplicación		
Descripción:	El analista cierra un determinado diagrama y el sistema actualiza el entorno de trabajo		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un diagrama abierto		

Curso Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Hace clic derecho sobre el nombre del diagrama que desea cerrar en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]	**2.** Obtiene el diagrama seleccionado
	3. Despliega un menú de opciones
4. Selecciona la opción [Cerrar] del menú desplegado	**5.** Verifica que el diagrama esté abierto
	6. Cierra la pestaña del diagrama seleccionado
	7. Retira la paleta del diagrama cerrado
	8. La meta **MT15** finaliza
Curso Alterno de Eventos	
A. Cerrar un diagrama a partir del panel [Panel de Edición]	
Acción del Actor	**Respuesta del Sistema**
A.1. Hace clic en el icono [Cerrar] sobre la pestaña del diagrama que desea cerrar, en el panel [Panel de Edición] de la pantalla [P001: MainAledanUML]	**A.2.** La meta **MT15** continúa en el **paso 6 del Curso Normal de Eventos**

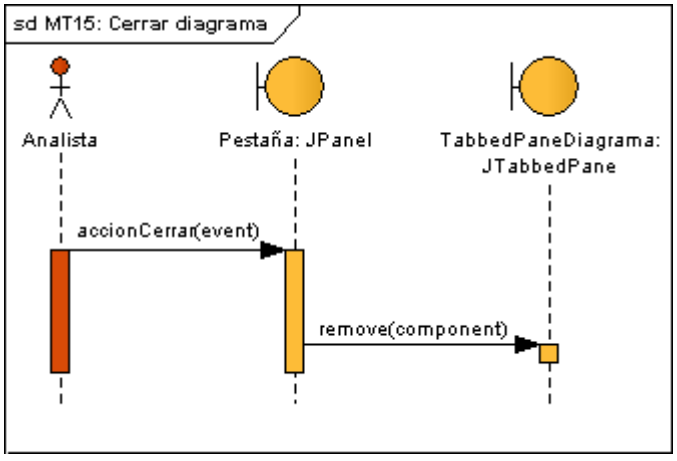


Figura 56: DS Curso normal cerrar diagrama

8.2.3.3.2.7. **Meta MT16:** Eliminar diagrama

Tabla 53: Descripción de la Meta Eliminar diagrama

Caso de Uso:	Administrar Diagrama	**Código Caso de Uso:**	CU02
Meta:	Eliminar diagrama	**Código Meta:**	MT16
Tipo:	Primario		
Referencia a requerimientos:	RF03, RF04, RF05, RF06, RF07, RF08		
Actor:	Analista		
Propósito:	Permitir al analista la eliminación de un diagrama creado previamente dentro de un proyecto		
Descripción:	El analista selecciona el diagrama que desea eliminar y ejecuta dicha acción; el sistema solicita la confirmación para eliminar el diagrama, y, si el analista la ratifica, el sistema lo elimina		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un proyecto abierto • Que exista al menos un diagrama creado previamente dentro del proyecto seleccionado		

Curso Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Hace clic derecho sobre el nombre del diagrama que desea eliminar en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]	**2.** Obtiene el diagrama seleccionado
	3. Despliega un menú de opciones
4. Selecciona la opción [Eliminar]	**5.** Presenta un mensaje de confirmación, en donde se pregunta al analista si desea eliminar el diagrama seleccionado
6. Presiona la opción [Sí] del mensaje de confirmación	**7.** Cierra la pestaña correspondiente al diagrama eliminado, en caso de encontrarse abierta
	8. Retira el diagrama del panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]
	9. Elimina el diagrama seleccionado
	10. La Meta **MT16** finaliza

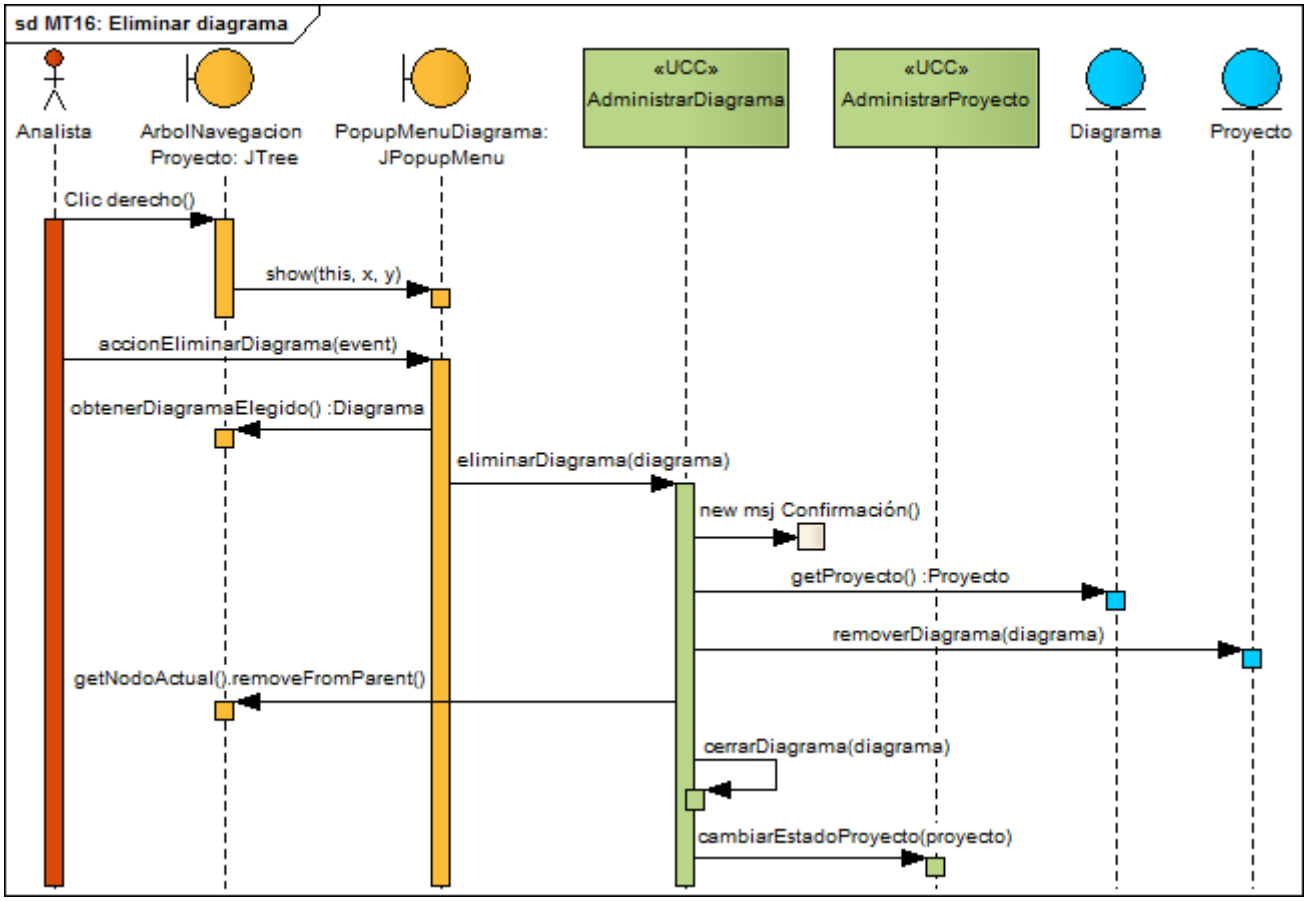


Figura 57: DS Curso normal eliminar diagrama

8.2.3.3.3. **Caso de Uso:** Administrar Artefacto

8.2.3.3.3.1. **Meta MT17:** Agregar artefacto

Tabla 54: Descripción de la Meta Agregar artefacto

Caso de Uso:	Administrar Artefacto	**Código Caso de Uso:**	CU03
Meta:	Agregar artefacto	**Código Meta:**	MT17
Tipo:	Primario		
Referencia a requerimientos:	RF02		
Actor:	Analista		
Propósito:	Permitir la creación de un artefacto dentro de un diagrama		
Descripción:	El analista crea un nuevo artefacto dentro del diagrama sobre el que esté trabajando y el sistema actualiza el entorno de trabajo con el nuevo elemento		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un diagrama abierto		
Post condiciones:	• Editar artefacto • Eliminar artefacto		

Curso Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Hace clic sobre el icono correspondiente al artefacto que desea agregar en el panel [Paleta] de la pantalla [P001: MainAledanUML]	
2. Hace clic sobre el área gráfica del diagrama en el panel [Panel de Edición] de la pantalla [P001: MainAledanUML]	**3.** Asigna valores por defecto al nuevo artefacto
	4. Carga las propiedades del nuevo artefacto y las muestra en el panel [Propiedades] de la pantalla [P001: MainAledanUML]
	5. Carga el nuevo artefacto y lo agrega en el lugar seleccionado por el analista en el panel [Panel de Edición] de la pantalla [P001: MainAledanUML]
	6. La meta **MT17** finaliza
A. Agregar relación	
Acción del Actor	**Respuesta del Sistema**
A.2. Hace clic sobre el nodo inicial en el panel [Panel de Edición] de la pantalla [P001: MainAledanUML]	**A.3.** Verifica si el artefacto seleccionado es válido como nodo inicial para el tipo de relación utilizada
A.4. Hace clic sobre el nodo final en el panel [Panel de Edición] de la pantalla [P001: MainAledanUML]	**A.5.** Verifica si el artefacto seleccionado es válido como nodo final para el tipo de relación utilizada
	A.6. La meta **MT17** continúa en el **paso 3 del Curso Normal de Eventos.**

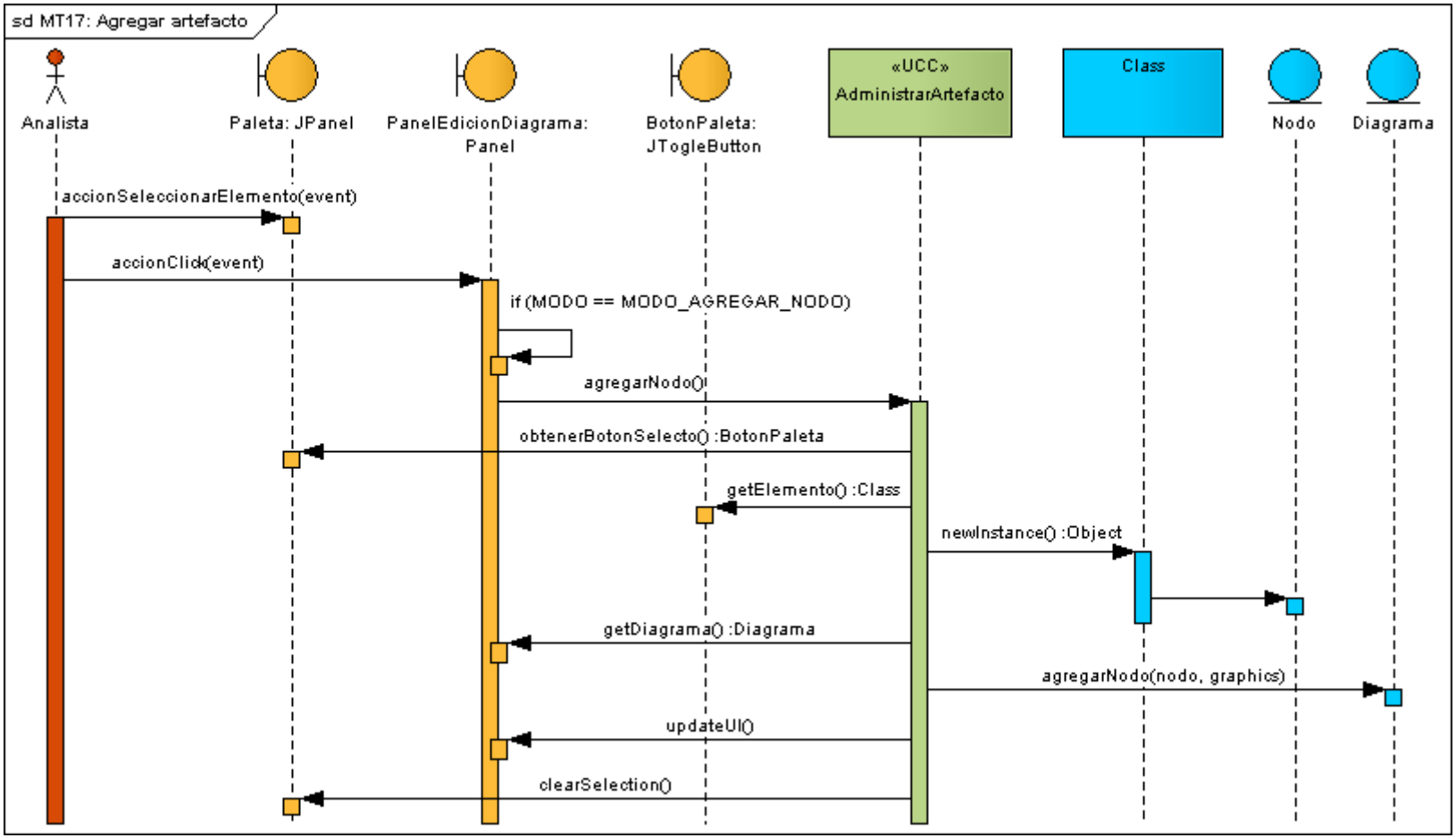
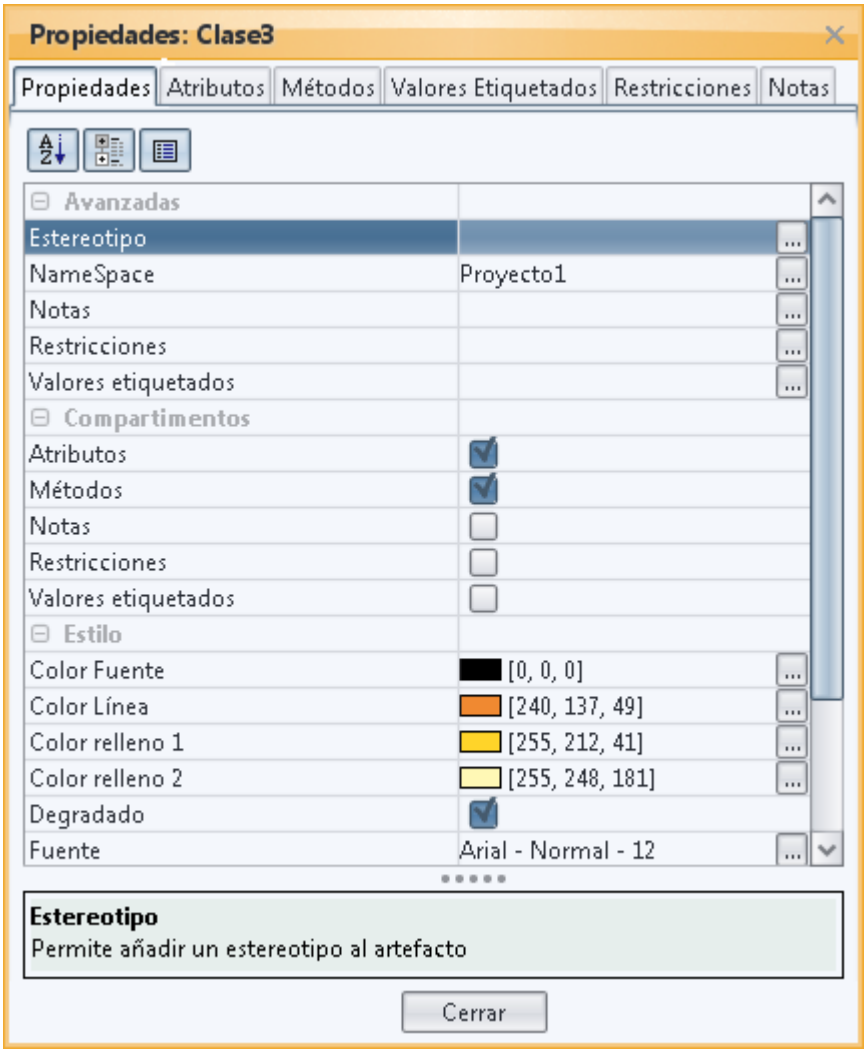


Figura 58: DS Curso normal agregar artefacto

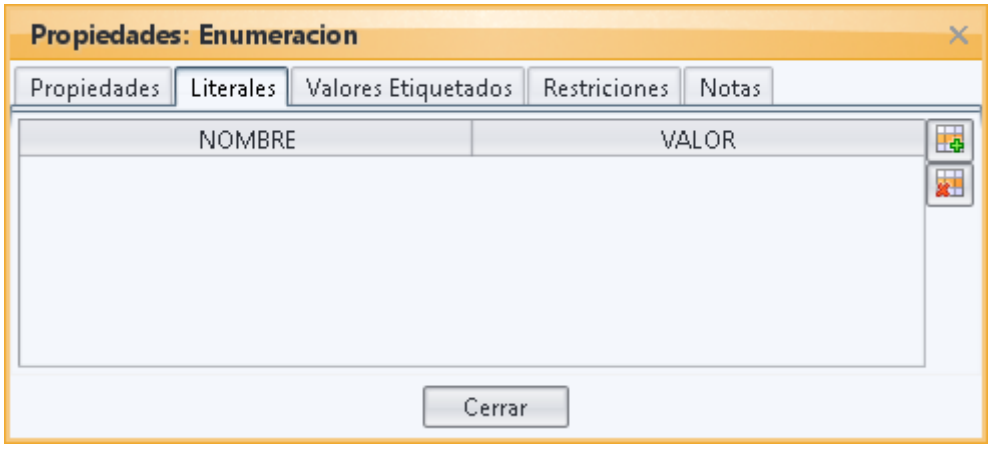
8.2.3.3.3.2. **Meta MT18:** Editar artefacto

Tabla 55: Prototipo de la Pantalla Propiedades

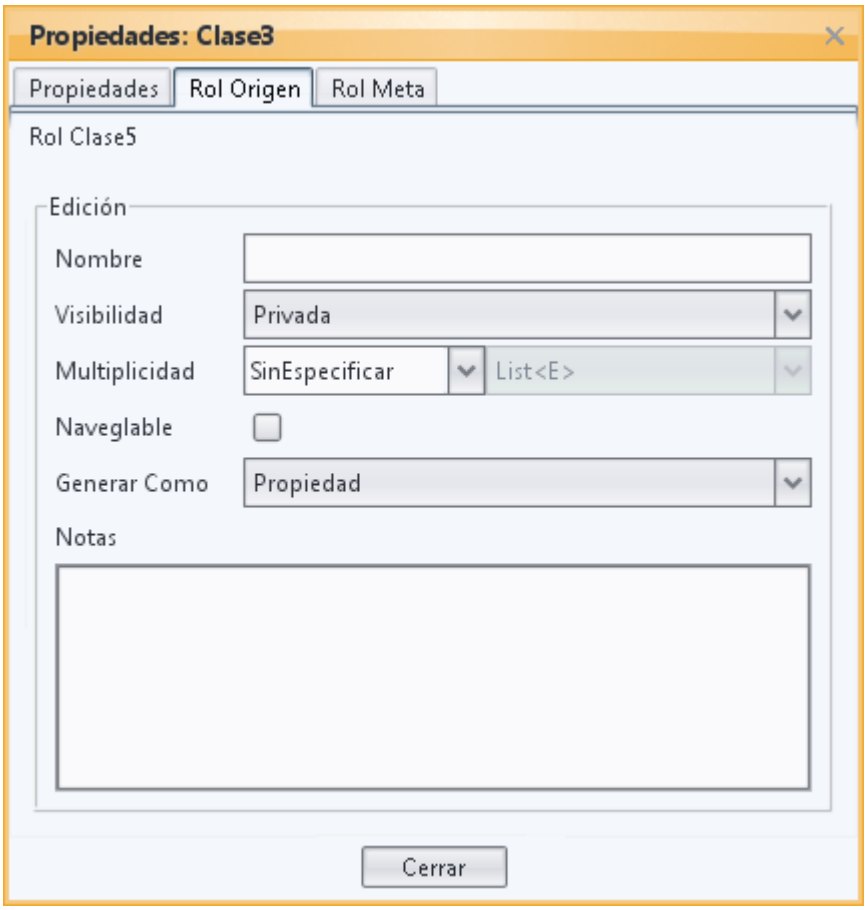
Nombre de la Pantalla:	Propiedades	**Código:**	P009
Tipo de Interfaz Gráfica:	JDialog		
Caso:	Propiedades		



Caso:	Literales



Caso:	Rol Origen y Meta



Caso:	Nota, Etiqueta, Restricción

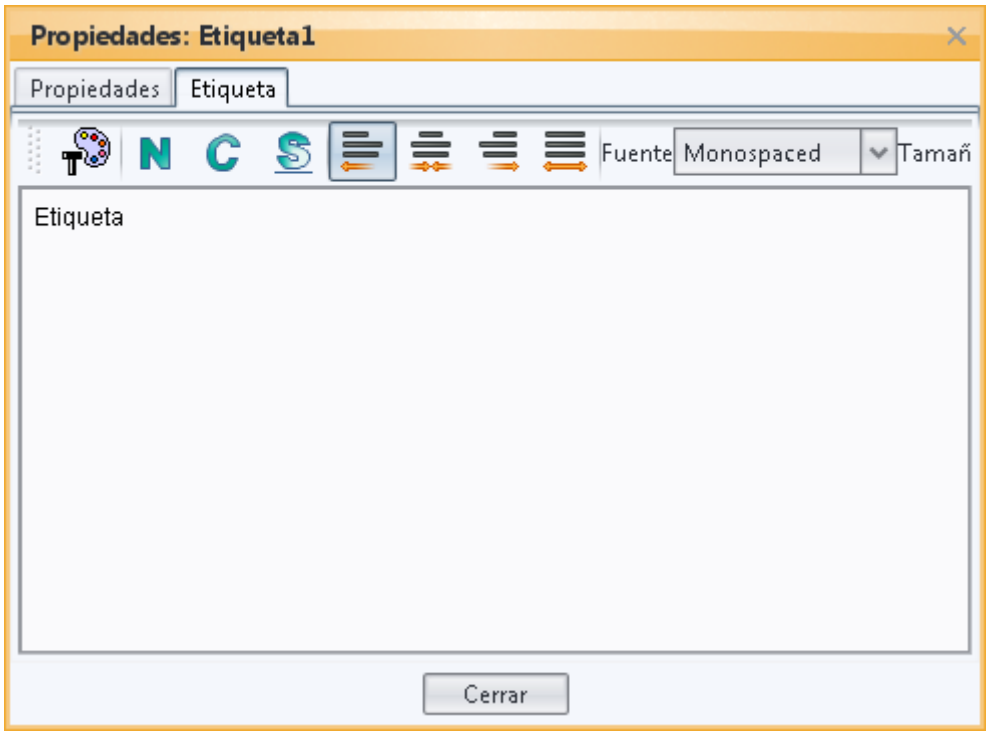


Tabla 56: Prototipo de la Pantalla DialogoSeleccionarClase

Nombre de la Pantalla:	DialogoSeleccionarClase	**Código:**	P011
Tipo de Interfaz Gráfica:	JDialog		

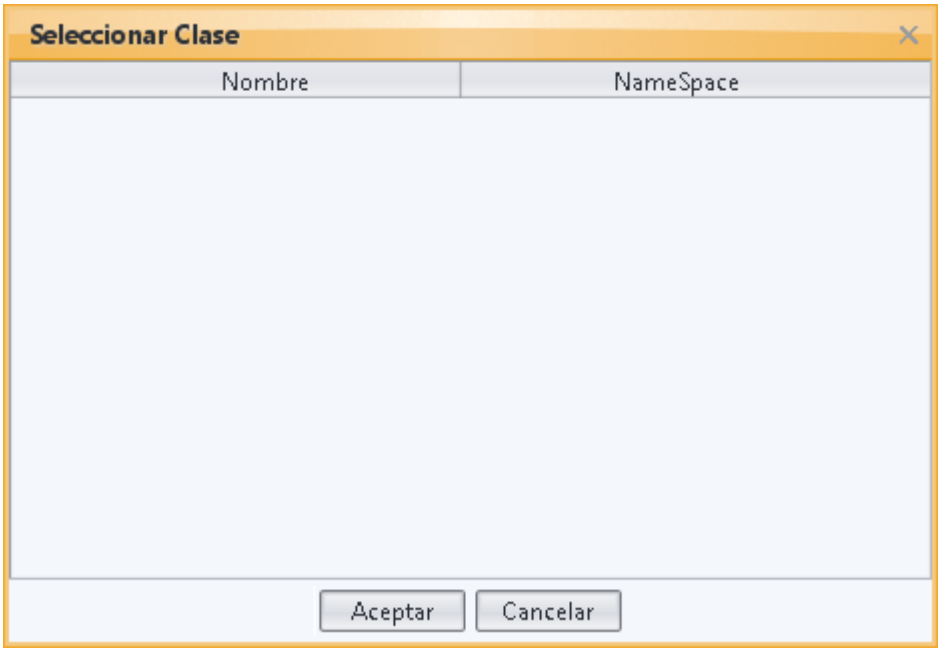


Tabla 57: Descripción de la Meta Editar artefacto

Caso de Uso:	Administrar Artefacto	**Código Caso de Uso:**	CU03
Meta:	Editar artefacto	**Código Meta:**	MT18
Tipo:	Primario		
Referencia a requerimientos:	RF02		
Actor:	Analista		
Propósito:	Permitir la edición de un artefacto creado dentro de un diagrama		
Descripción:	El analista selecciona un artefacto dentro de un diagrama, para la correspondiente edición en cualquiera de sus propiedades y procede a modificarlo. En caso de que el artefacto seleccionado sea una especialización de clasificador, el sistema le brinda la posibilidad de seleccionar la clase a la cual deberá hacer referencia el artefacto		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un artefacto creado • Si se desea vincular el artefacto a una clase, que sea una especialización de clasificador		

Curso Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Hace clic derecho sobre el artefacto que desea seleccionar en el panel [Panel de Edición] de la pantalla [P001: MainAledanUML]	**2.** Obtiene el artefacto seleccionado
	3. Carga las propiedades del artefacto
	4. Despliega un menú de opciones
5. Selecciona la opción [Propiedades] del menú desplegado	**6.** Presenta la pantalla [P009: Propiedades]
7. Modifica los valores en una o varias	**8.** Verifica que los valores modificados por

propiedades del artefacto en la pestaña [Propiedades] de la pantalla [P009: Propiedades]	el analista estén dentro del rango permitido y que sean válidos
	9. Actualiza los valores de las propiedades del artefacto
	10. La meta **MT18** finaliza
Curso Alterno de Eventos	
A. Editar artefacto a partir del [Panel de Edición]	
Acción del Actor	**Respuesta del Sistema**
A.1. Hace doble clic sobre el artefacto que desea editar en el panel [Panel de Edición] de la pantalla [P001: MainAledanUML]	**A.2.** Obtiene el artefacto que se desea editar
	A.3. Carga las propiedades del artefacto
	A.4. La meta **MT18** continúa en el **paso 6 del Curso Normal de Eventos**
B. Editar artefacto a partir del panel [Propiedades]	
Acción del Actor	**Respuesta del Sistema**
B.1. Hace clic sobre el artefacto que desea editar en el panel [Panel de Edición] de la pantalla [P001: MainAledanUML]	**B.2.** Obtiene el artefacto seleccionado
	B.3. Carga las propiedades del artefacto
B.4. Modifica los valores en cualquiera de las propiedades del artefacto en el panel [Propiedades] de la pantalla [P001: MainAledanUML]	**B.5.** La meta **MT18** continúa en el **paso 8 del Curso Normal de Eventos**
C. Editar formato gráfico	
Acción del Actor	**Respuesta del Sistema**
C.B.4. Modifica los valores en cualquiera de las propiedades gráficas del artefacto en el menú [Formato] de la pantalla [P001: MainAledanUML]	**C.B.5.** La meta **MT18** continúa en el **paso 8 del Curso Normal de Eventos**
D. Seleccionar clasificador	
Acción del Actor	**Respuesta del Sistema**
D.5. Selecciona la opción [Clasificador] del menú desplegado	**D.6.** Obtiene el listado de las clases creadas dentro del proyecto
	D.7. Presenta la pantalla [P011: DialogoSeleccionarClase] con las clases encontradas
D.8. Selecciona la clase a la que se desea vincular el artefacto	
D.9. Hace clic en la opción [Aceptar]	**D.10.** Verifica que se haya seleccionado una clase
	D.11. Vincula el artefacto a la clase seleccionada
	D.12. La meta **MT18** finaliza

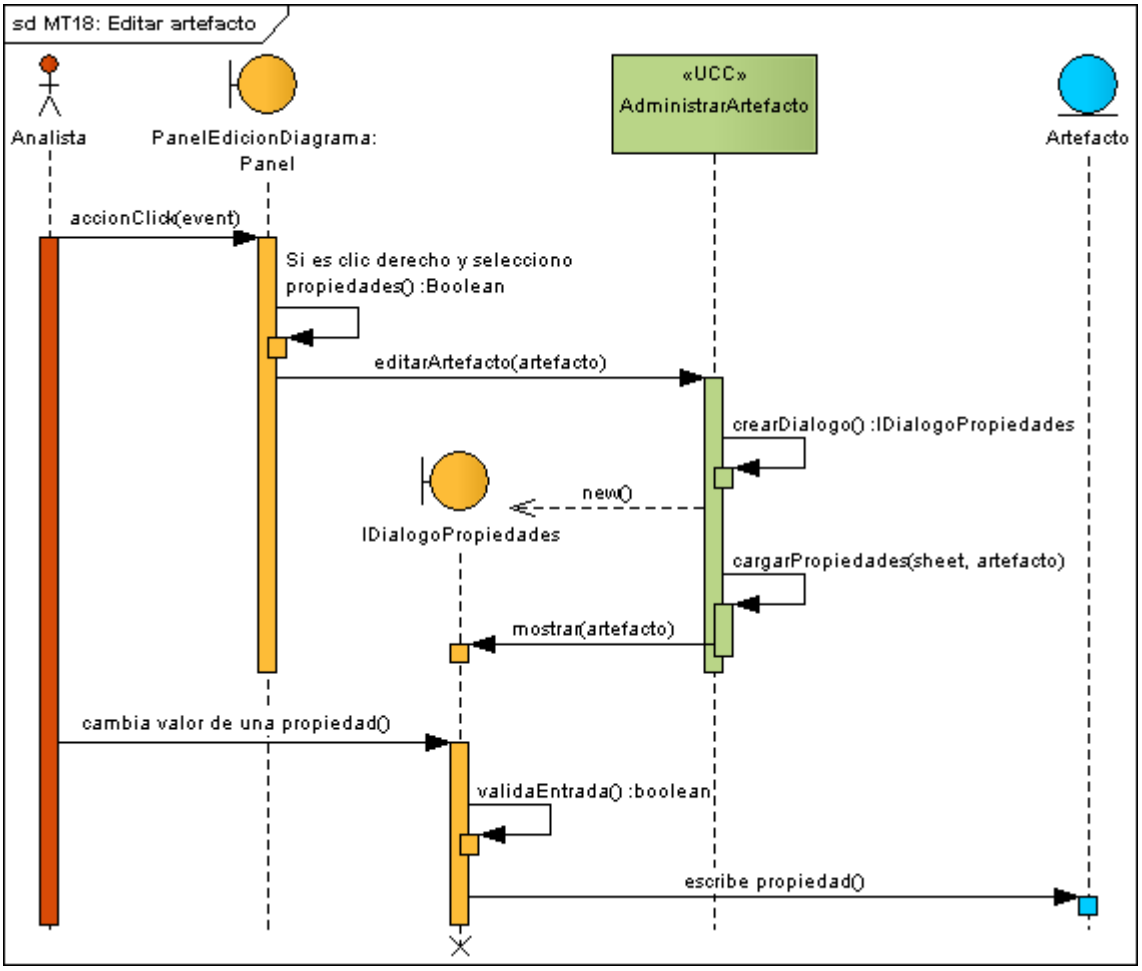


Figura 59: DS Curso normal editar artefacto

8.2.3.3.3.3. **Meta MT19:** Administrar atributo

Tabla 58: Prototipo de la Pantalla Propiedades - Atributos

Nombre de la Pantalla:	Propiedades	**Código:**	P009
Tipo de Interfaz Gráfica:	JDialog		
Caso:	Atributos		

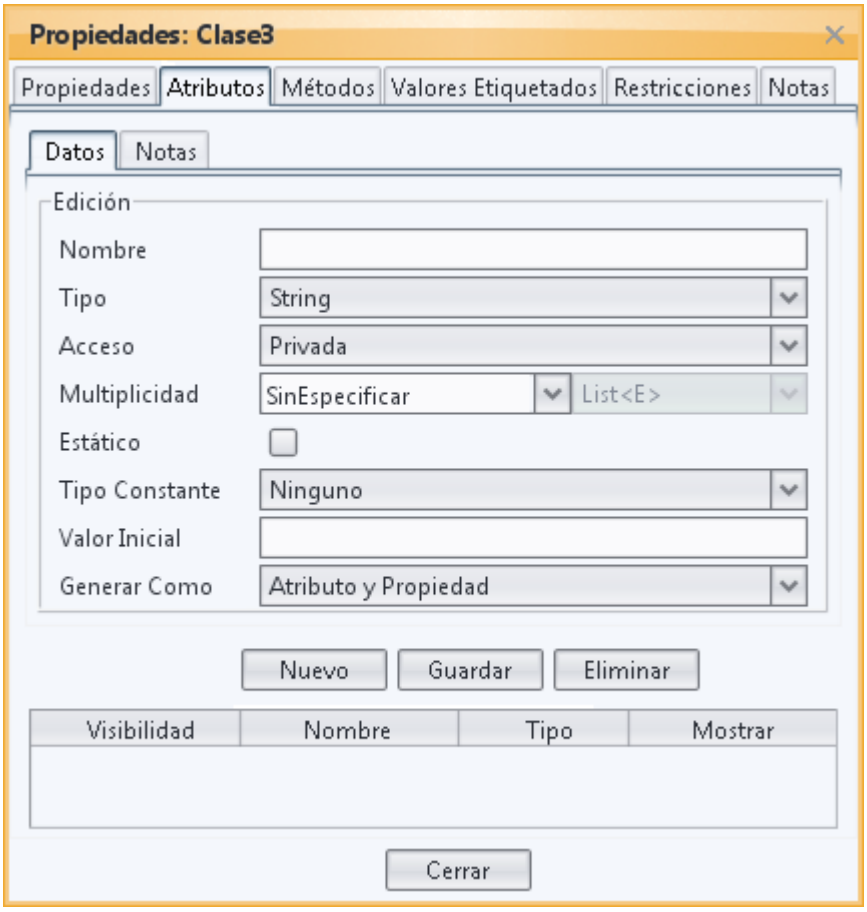


Tabla 59: Descripción de la Meta Administrar atributo

Caso de Uso:	Administrar Artefacto	**Código Caso de Uso:**	CU03
Meta:	Administrar atributo	**Código Meta:**	MT19
Tipo:	Primario		
Referencia a requerimientos:	RF02		
Actor:	Analista		
Propósito:	Permitir crear, modificar o eliminar atributos a un nodo determinado, siempre que éste lo permita		
Descripción:	El analista selecciona un nodo para la correspondiente administración de atributos		
Pre condiciones:	• Que el analista se encuentre en la pestaña [Atributos] de la pantalla [P009: Propiedades] • Que las especificaciones de UML 2.0 permitan la administración de atributos para el nodo seleccionado		

Curso Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Ingresa los datos del nuevo atributo, en los campos correspondientes de la pestaña [Datos] de la pantalla [P009: Propiedades]	
2. Selecciona la opción [Guardar] en la pantalla [P009: Propiedades]	**3.** Valida que los datos ingresados para el nuevo atributo sean correctos
	4. Almacena el nuevo atributo para el artefacto en edición
	5. Muestra la nueva propiedad en la pantalla [P009: Propiedades]
	6. La meta **MT19** finaliza
Curso Alterno de Eventos	
A. Eliminar Atributo	
Acción del Actor	**Respuesta del Sistema**
A.1. Selecciona el atributo que desea eliminar en la pantalla [P009: Propiedades]	
A.2. Selecciona la opción [Eliminar] en la pantalla [P009: Propiedades]	**A.3.** Obtiene atributo seleccionado
	A.4. Elimina el atributo seleccionado
	A.5. Retira el atributo eliminado de la pantalla [P009: Propiedades]
	A.6. La meta **MT19** finaliza
B. Editar Atributo	
Acción del Actor	**Respuesta del Sistema**
D.1. Selecciona el atributo que desea editar en la pantalla [P009: Propiedades]	**D.2.** Obtiene el atributo seleccionado
	D.3. Carga los datos del atributo seleccionado en los campos respectivos de la pantalla [P009: Propiedades]
D.4. Edita los datos que desea en la pestaña [Datos] de la pantalla [P009: Propiedades]	**D.5.** La meta **MT19** continúa en el **paso 2 del Curso Normal de Eventos**

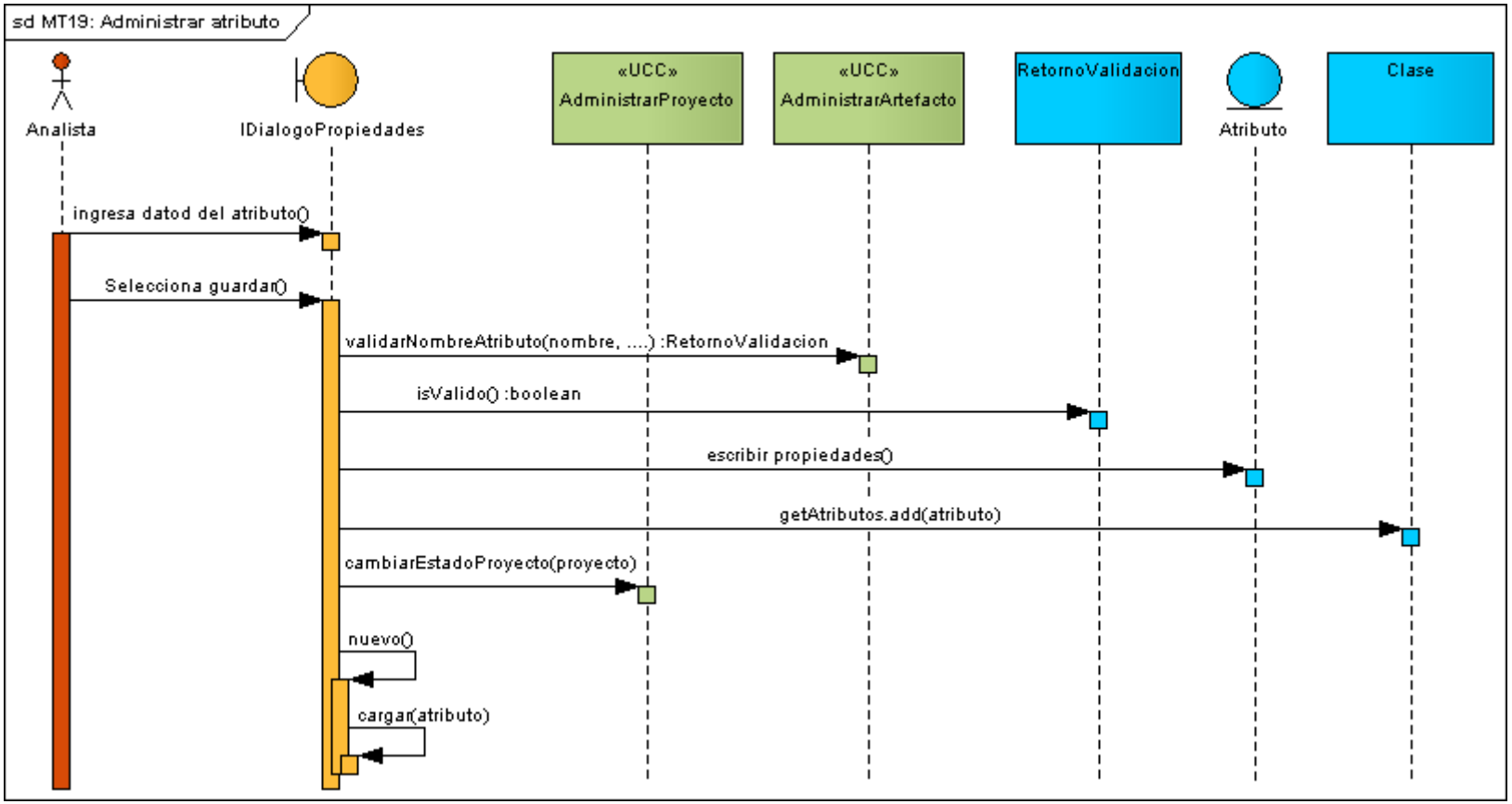


Figura 60: DS Curso normal administrar atributo

8.2.3.3.3.4. **Meta MT20:** Administrar método

Tabla 60: Prototipo de la Pantalla Propiedades – Métodos

Nombre de la Pantalla:	Propiedades	**Código:**	P009
Tipo de Interfaz Gráfica:	JDialog		
Caso:	Métodos		

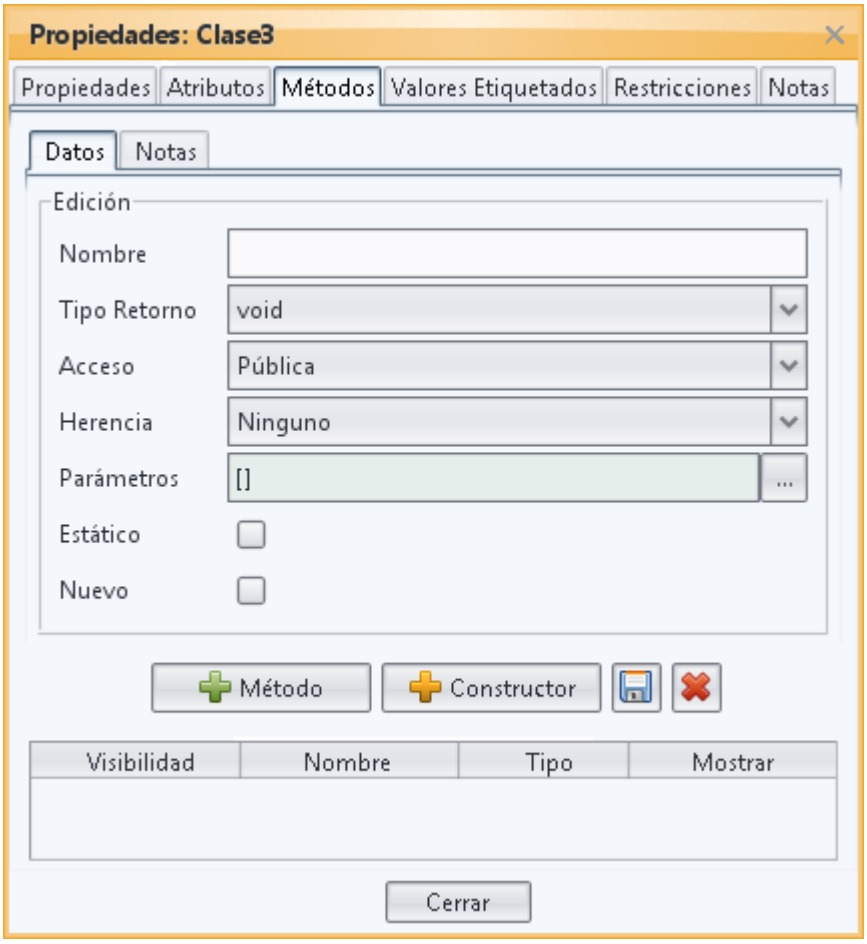


Tabla 61: Prototipo de la Pantalla DialogoEdicionParametros

Nombre de la Pantalla:	DialogoEdicionParametros	**Código:**	P010
Tipo de Interfaz Gráfica:	JDialog		

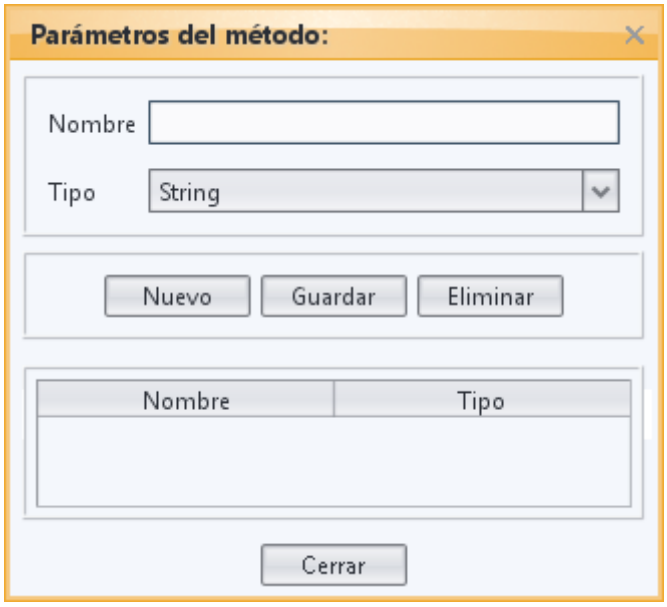


Tabla 62: Descripción de la Meta Administrar método

Caso de Uso:	Administrar Artefacto	**Código Caso de Uso:**	CU03
Meta:	Administrar método	**Código Meta:**	MT20
Tipo:	Primario		
Referencia a requerimientos:	RF02		
Actor:	Analista		
Propósito:	Permitir crear, modificar o eliminar métodos para un nodo determinado, siempre que éste lo permita		
Descripción:	El analista selecciona un nodo para la correspondiente administración de métodos		
Pre condiciones:	• Que el analista se encuentre en la pestaña [Métodos] de la pantalla [P009: Propiedades] • Que las especificaciones de UML 2.0 permitan la administración de métodos para el nodo seleccionado		
Post condiciones:			

Curso Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Ingresa los datos del nuevo método, en los campos correspondientes de la pestaña [Datos] en la pantalla [P009: Propiedades]	
2. Selecciona la opción [Guardar] en la pantalla [P009: Propiedades]	**3.** Valida que los datos ingresados para el nuevo método sean correctos
	4. Almacena el nuevo método para el nodo en edición
	5. Muestra la nueva propiedad en la pantalla [P009: Propiedades]

	6. La meta **MT20** finaliza
Curso Alterno de Eventos	
A. Agregar parámetros	
Acción del Actor	**Respuesta del Sistema**
A.2. Presiona la opción […] del campo [Parámetros] en la pestaña [Datos] de la pantalla [P009: Propiedades]	**A.3.** Valida que los datos ingresados para el método sean correctos
	A.4. Presenta la pantalla [P010: DialogoEdicionParametros]
A.5. Ingresa los datos del nuevo parámetro en la pantalla [P010: DialogoEdicionParametros]	
A.6. Selecciona la opción [Guardar] en la pantalla [P010: DialogoEdicionParametros]	**A.7.** Valida que los datos del nuevo parámetro sean correctos
	A.8. Almacena el nuevo parámetro para el método en edición
	A.9. Muestra el nuevo parámetro en la pantalla [P010: DialogoEdicionParametros]
A.10. Hace clic sobre la opción [Cerrar] de la pantalla [P010: DialogoEdicionParametros]	**A.11.** La meta **MT20** continúa en el **paso 2 del Curso Normal de Eventos**
B. Eliminar Método	
Acción del Actor	**Respuesta del Sistema**
B.1. Selecciona el método que desea eliminar en la pantalla [P009: Propiedades]	
B.2. Selecciona la opción [Eliminar] en la pantalla [P009: Propiedades]	**B.3.** Obtiene método seleccionado en la pantalla [P009: Propiedades]
	B.4. Elimina el método seleccionado
	B.5. Retira el método eliminado de la pantalla [P009: Propiedades]
	B.6. La meta **MT20** finaliza
C. Editar Método	
Acción del Actor	**Respuesta del Sistema**
C.1. Selecciona el método que desea editar en la pantalla [P009: Propiedades]	**C.2.** Obtiene el método seleccionado
	C.3. Carga los datos del método seleccionado en los campos correspondientes de la pantalla [P009: Propiedades]
C.4. Edita los datos que desea en la pantalla [P009: Propiedades]	**C.5.** La meta **MT20** continúa en el **paso 2 del Curso Normal de Eventos**

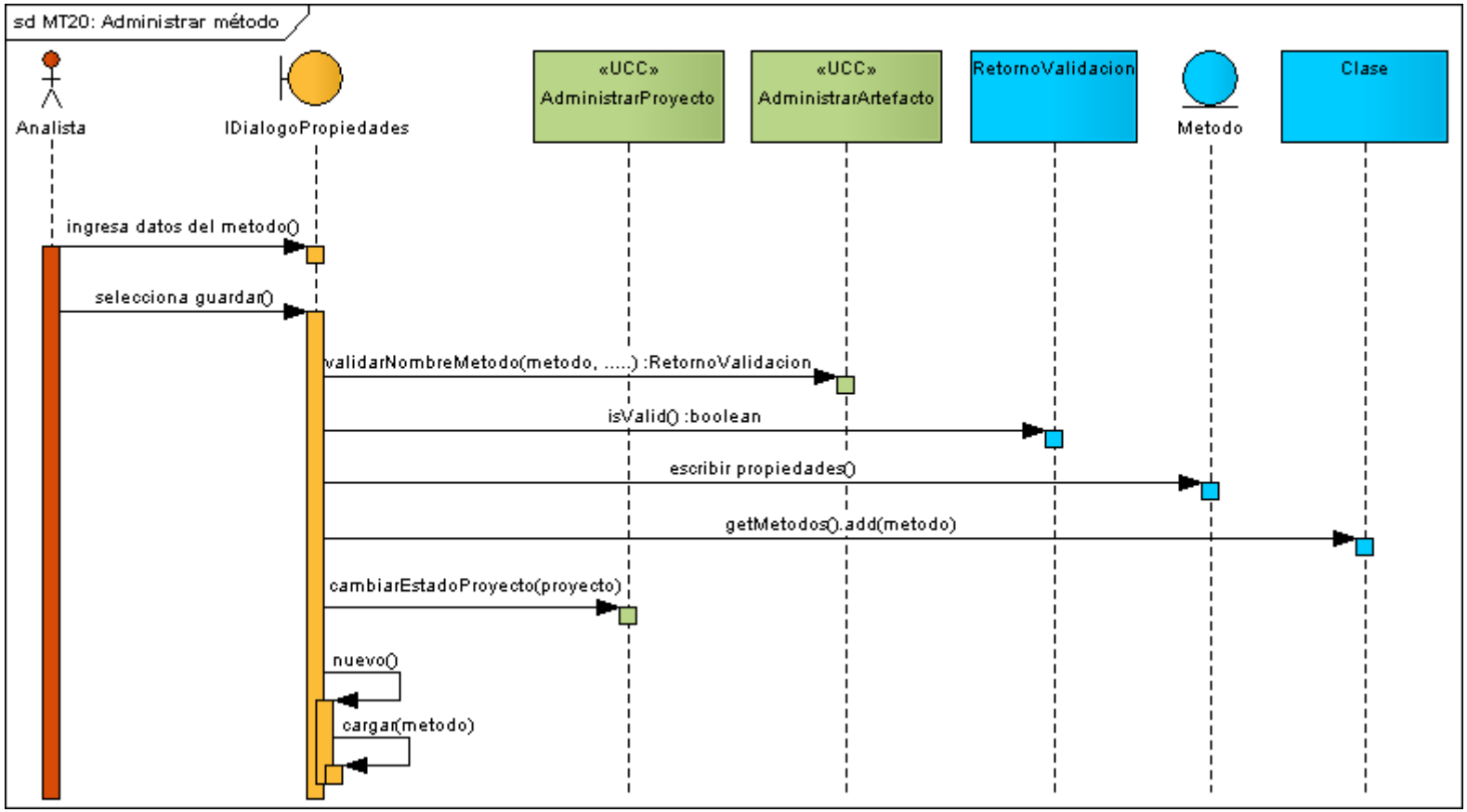
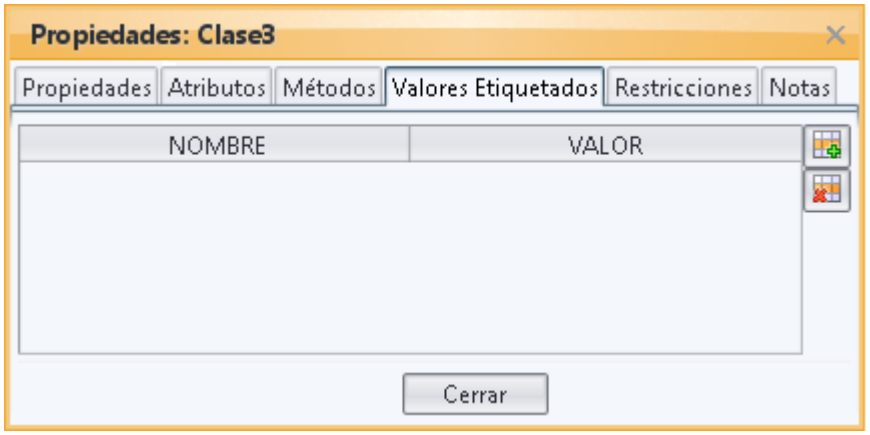


Figura 61: DS Curso normal administrar método

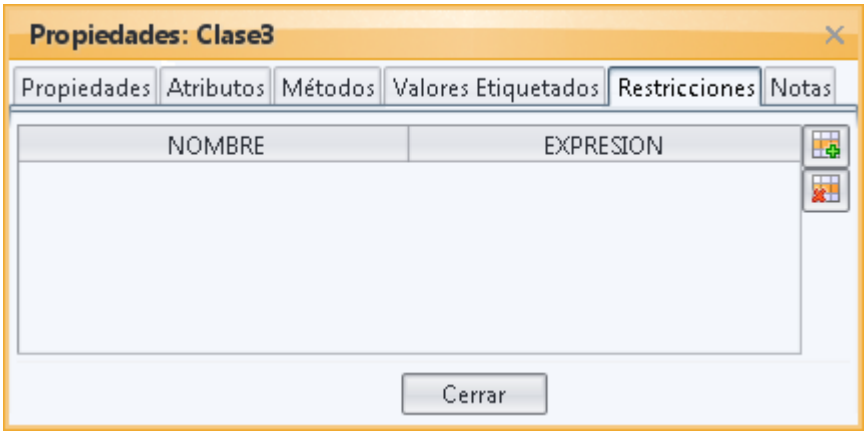
8.2.3.3.3.5. **Meta MT21:** Administrar mecanismo de extensibilidad

Tabla 63: Prototipo de la Pantalla Propiedades – Valores Etiquetados

Nombre de la Pantalla:	Propiedades	**Código:**	P009
Tipo de Interfaz Gráfica:	JDialog		
Caso:	Valores Etiquetados		



Caso:	Restricciones



Caso:	Notas

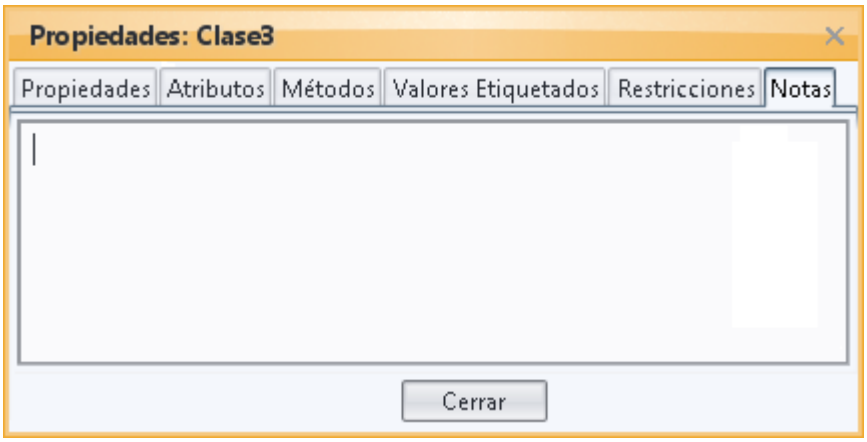


Tabla 64: Descripción de la Meta Administrar mecanismo de extensibilidad

Caso de Uso:	Administrar Artefacto	**Código Caso de Uso:**	CU03
Meta:	Administrar mecanismo de extensibilidad	**Código Meta:**	MT21
Tipo:	Primario		
Referencia a requerimientos:	RF02		
Actor:	Analista		
Propósito:	Permitir crear, modificar o eliminar mecanismos de extensibilidad para un artefacto determinado		
Descripción:	El analista selecciona un artefacto para la correspondiente administración de mecanismos de extensibilidad		
Pre condiciones:	• Que el analista se encuentre en cualquiera de las pestañas [Valores Etiquetados], [Restricciones] o [Notas] de la pantalla [P009: Propiedades], según sea el caso		
Post condiciones:			

Curso Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Presiona la opción [Agregar] de la pestaña [Valores Etiquetados] o [Restricciones] en la pantalla [P009: Propiedades]	**2.** Presenta un nueva fila en la pantalla [P009: Propiedades]
3. Edita los campos requeridos para crear el nuevo mecanismo de extensibilidad en la pantalla [P009: Propiedades]	**4.** Verifica que los valores editados sean válidos
	5. Crea el nuevo mecanismo de extensibilidad de acuerdo a las especificaciones dadas por el analista
	6. Muestra el nuevo mecanismo de extensibilidad en la pantalla [P009: Propiedades]
7. Hace clic sobre la opción [Cerrar] de la pantalla [P009: Propiedades]	**8.** La meta **MT21** finaliza
Curso Alterno de Eventos	
A. Agregar nota	
Acción del Actor	**Respuesta del Sistema**
A.1. Ingresa el texto deseado en la pestaña [Nota] de la pantalla [P009: Propiedades]	**A.2.** La meta **MT21** continúa en el **paso 5 del Curso Normal de Eventos**
B. Eliminar valor etiquetado o restricción	
Acción del Actor	**Respuesta del Sistema**
B.1. Selecciona el valor etiquetado o restricción que desea eliminar, en la pestaña correspondiente de la pantalla [P009: Propiedades]	
B.2. Selecciona la opción [Eliminar] de la pantalla [P009: Propiedades]	**B.3.** Obtiene mecanismo de extensibilidad seleccionado
	B.4. Elimina el mecanismo de extensibilidad seleccionado
	B.5. Retira el mecanismo de extensibilidad de la pantalla [P009: Propiedades]
	B.6. La meta **MT21** continúa en el **paso 6 del Curso Normal de Eventos**

C. Eliminar nota	
Acción del Actor	**Respuesta del Sistema**
C.1. Borra el texto en el campo [Nota] de la pantalla [P009: Propiedades]	**C.2.** La meta **MT21** continúa en el **paso 6 del Curso Normal de Eventos**

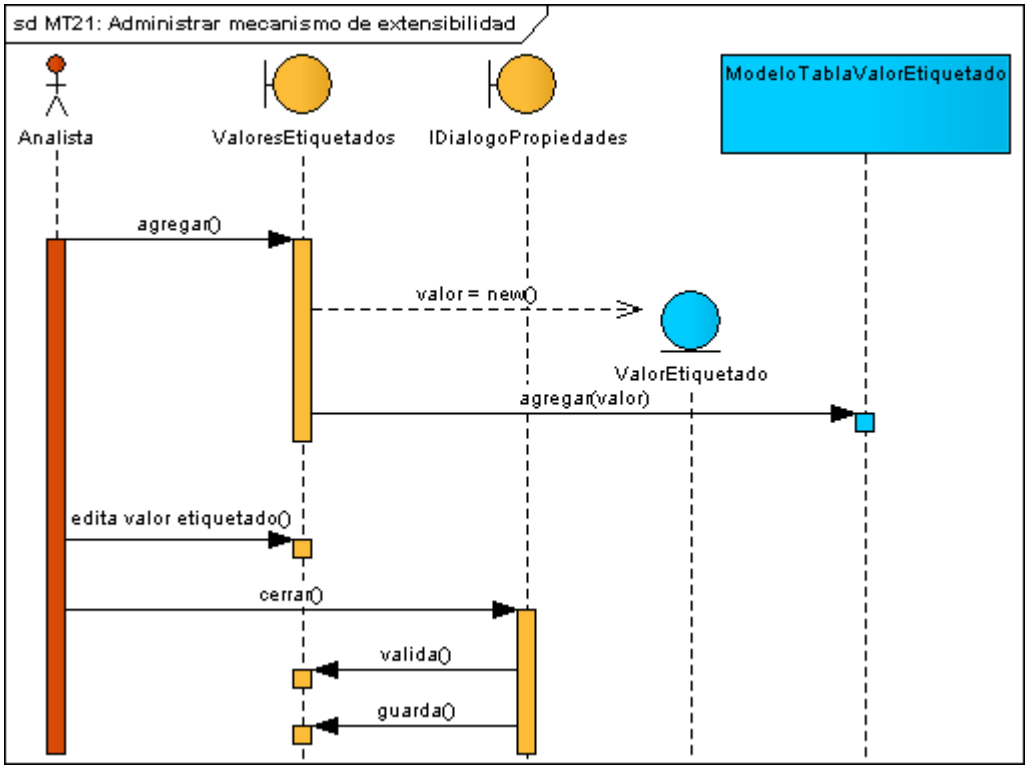


Figura 62: DS Curso normal administrar mecanismo de extensibilidad

8.2.3.3.3.6. **Meta MT22:** Eliminar artefacto

Tabla 65: Descripción de la Meta Eliminar artefacto

Caso de Uso:	Administrar Artefacto	**Código Caso de Uso:**	CU03
Meta:	Eliminar artefacto	**Código Meta:**	MT22
Tipo:	Primario		
Referencia a requerimientos:	RF02		
Actor:	Analista		
Propósito:	Permitir al analista la eliminación de un artefacto creado dentro de un diagrama		
Descripción:	El analista selecciona el artefacto que desea eliminar del diagrama y ejecuta dicha acción; el sistema elimina el artefacto tanto del diagrama como del modelo del proyecto		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un artefacto creado		
Post condiciones:	• Hacer y deshacer acciones		

Curso Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Hace clic derecho sobre el artefacto que desea seleccionar en el panel [Panel de Edición] de la pantalla [P001: MainAledanUML]	**2.** Obtiene el artefacto seleccionado
	3. Carga las propiedades del artefacto en el panel [Propiedades] de la pantalla [P001: MainAledanUML]
	4. Despliega un menú de opciones
5. Selecciona la opción [Eliminar] del menú desplegado	**6.** Elimina el artefacto seleccionado del panel [Panel de Edición]
	7. Presenta información de la operación realizada en el panel [Salida] de la pantalla [P001: MainAledanUML]
	8. Actualiza el entorno de trabajo
	9. La meta **MT22** finaliza

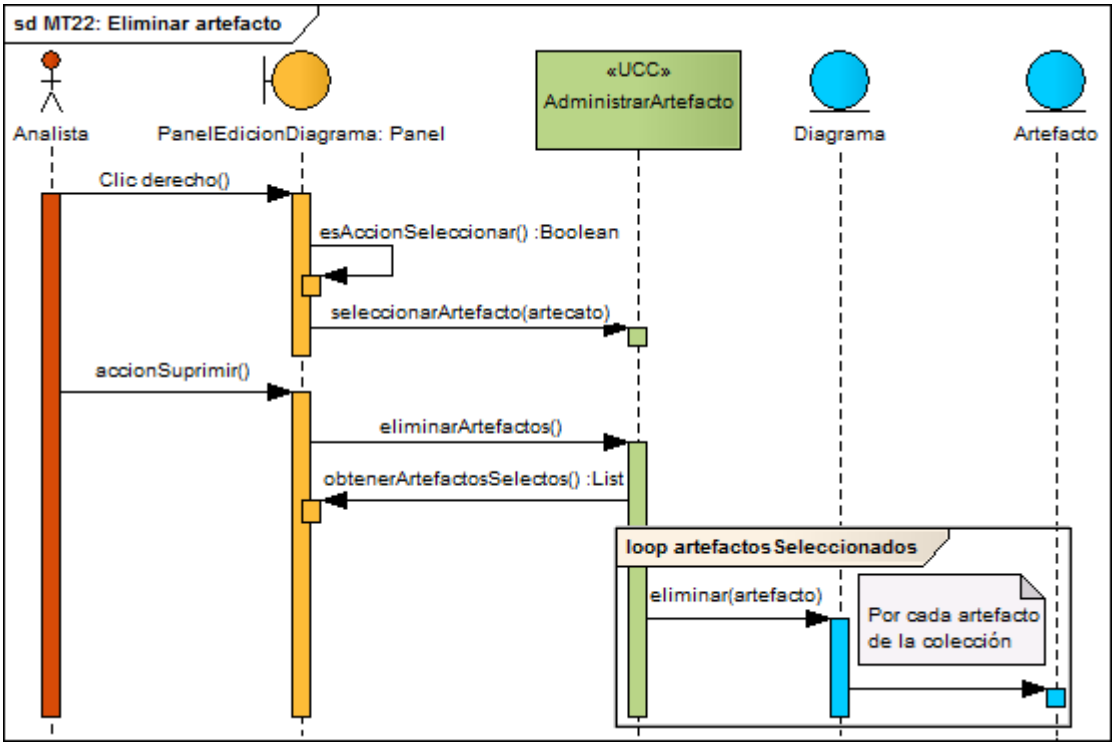


Figura 63: DS Curso normal eliminar artefacto

8.2.3.3.4. Caso de Uso CU04: Generar código

8.2.3.3.4.1. Meta MT23: Generar código fuente para C#

Tabla 66: Prototipo de la Pantalla AsistenteGenerarCodigo

Nombre de la Pantalla:	AsistenteGenerarCodigo	Código:	P008
Tipo de Interfaz Gráfica:	JFileChooser		

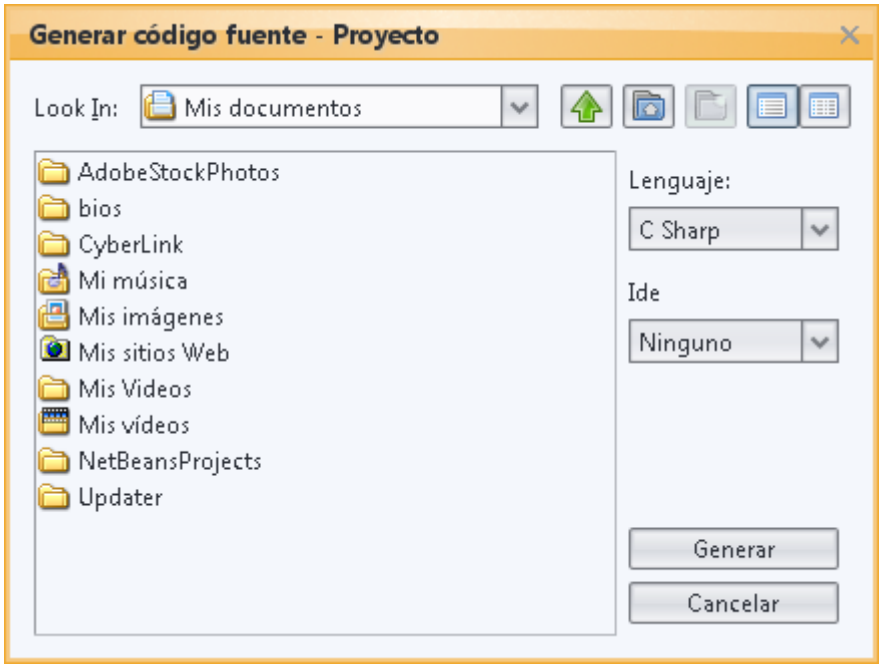


Tabla 67: Descripción de la Meta Generar código fuente para C#

Caso de Uso:	Generar Código	Código Caso de Uso:	CU04
Meta:	Generar código fuente para C#	Código Meta:	MT23
Tipo:	Primario		
Referencia a requerimientos:	RF15		
Actor:	Analista		
Propósito:	Permitir al analista generar código fuente para el lenguaje de programación C#, a partir de los diagramas diseñados previamente		
Descripción:	El analista selecciona el diagrama a partir del cual desea obtener el código fuente y ejecuta dicha acción; el sistema obtiene todos los artefactos utilizados en el diagrama y presenta el código fuente generado en un archivo de texto, cuya ubicación es definida por el analista		
Pre condiciones:	• Que el analista se encuentre en la pantalla [P001: MainAledanUML] • Que exista al menos un diagrama abierto		
Curso Normal de Eventos			
Acción del Actor		**Respuesta del Sistema**	
1. Hace clic derecho sobre el nombre del proyecto a partir del cual se desea generar código, en el panel [Navegador		**2.** Obtiene el proyecto seleccionado	

de Proyectos] de la pantalla [P001: MainAledanUML]	
	3. Despliega un menú de opciones
4. Selecciona la opción [Generar Código] del menú desplegado	**5.** Presenta la pantalla [P008: AsistenteGenerarCodigo]
6. Selecciona la carpeta de destino en la que se generará el código	
7. Selecciona las opciones deseadas para el código generado	
8. Presiona la opción [Generar] de la pantalla	**9.** Genera el código fuente del proyecto
	10. La meta **MT23** finaliza
Curso Alterno de Eventos	
A. Generar código a partir del menú [Inicio]	
Acción del Actor	**Respuesta del Sistema**
A.1. Selecciona el nombre del proyecto a partir del cual se desea generar código, en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]	
A.2. Presiona la opción [Generar Código] en el menú [Inicio] de la pantalla [P001: MainAledanUML]	**A.3.** Obtiene el proyecto seleccionado en el panel [Navegador de Proyectos] de la pantalla [P001: MainAledanUML]
	A.4. La meta **MT23** continúa en el **paso 5 del Curso Normal de Eventos**

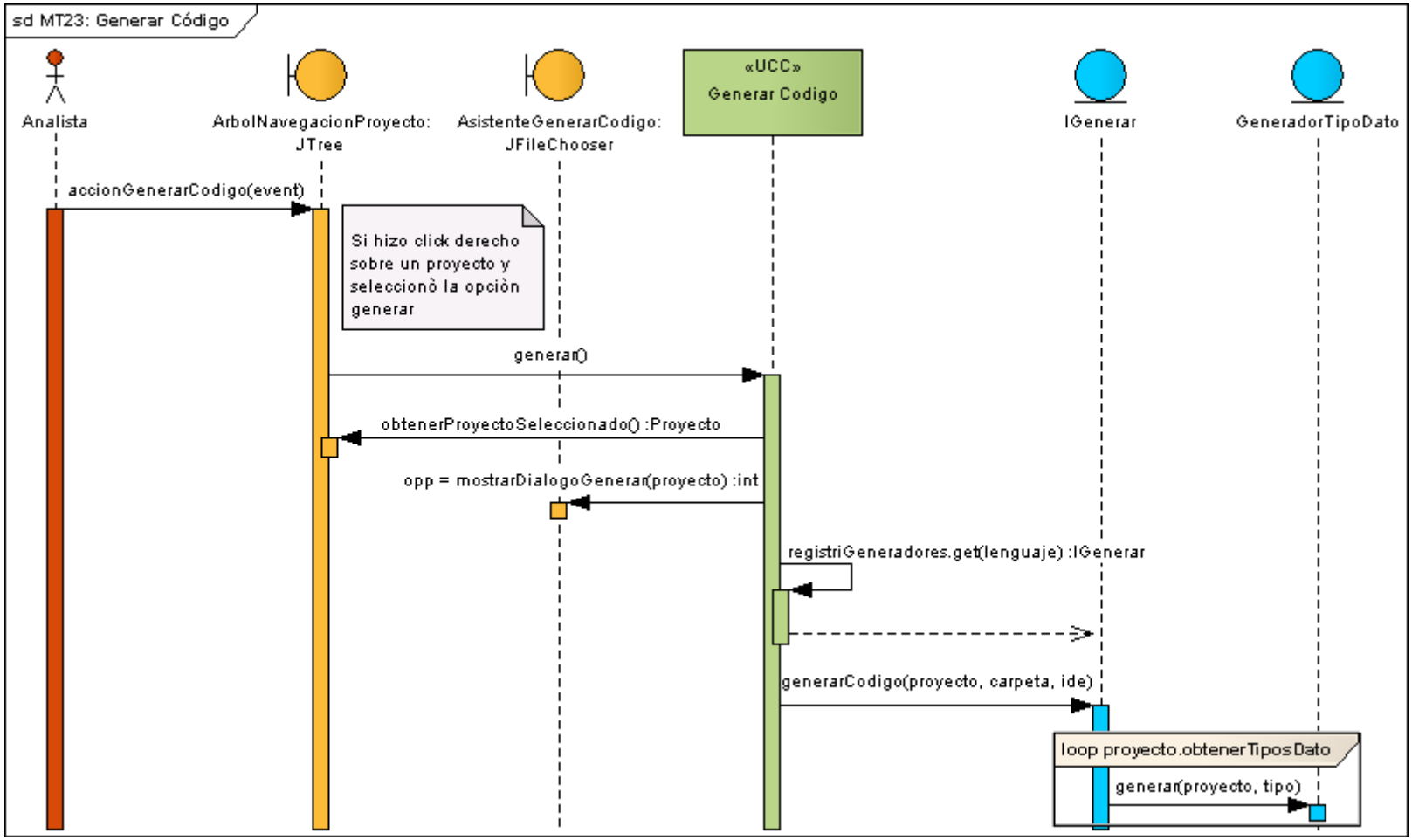


Figura 64: DS Curso normal generar código

8.2.4. Diagramas del sistema

8.2.4.1. Diagramas de Clases

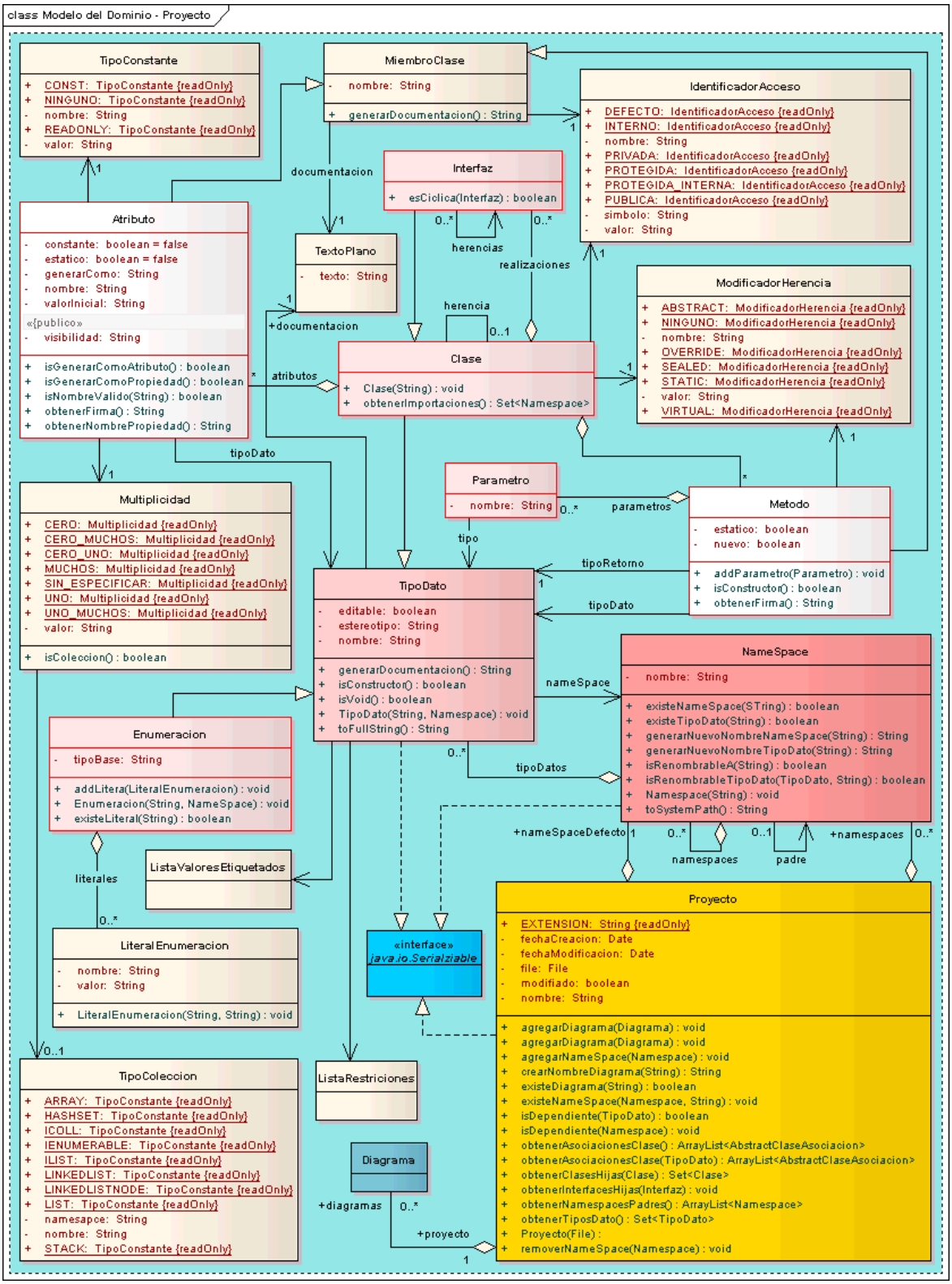


Figura 65: DC Proyecto

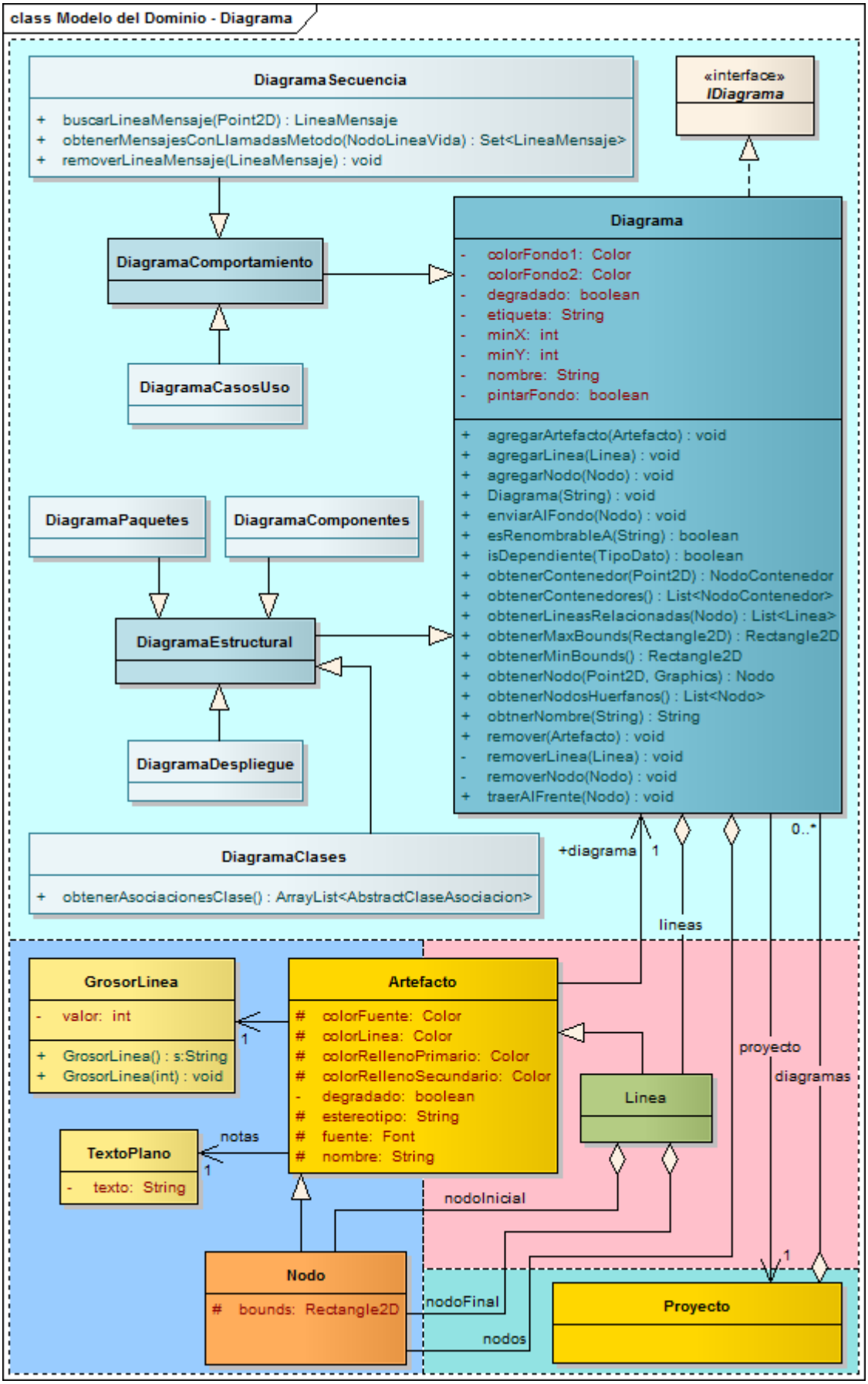


Figura 66: DC Diagrama

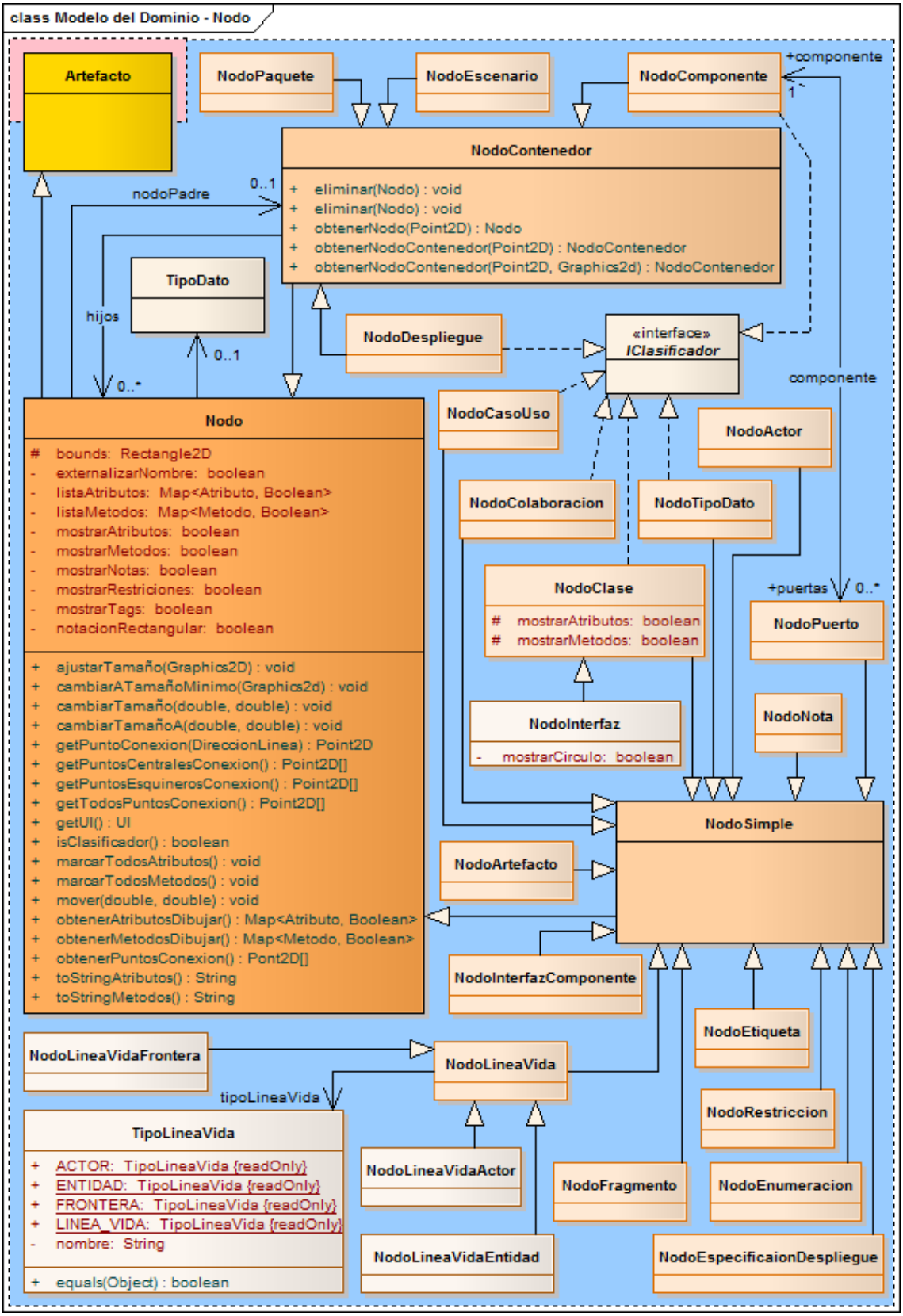


Figura 67: DC Nodo

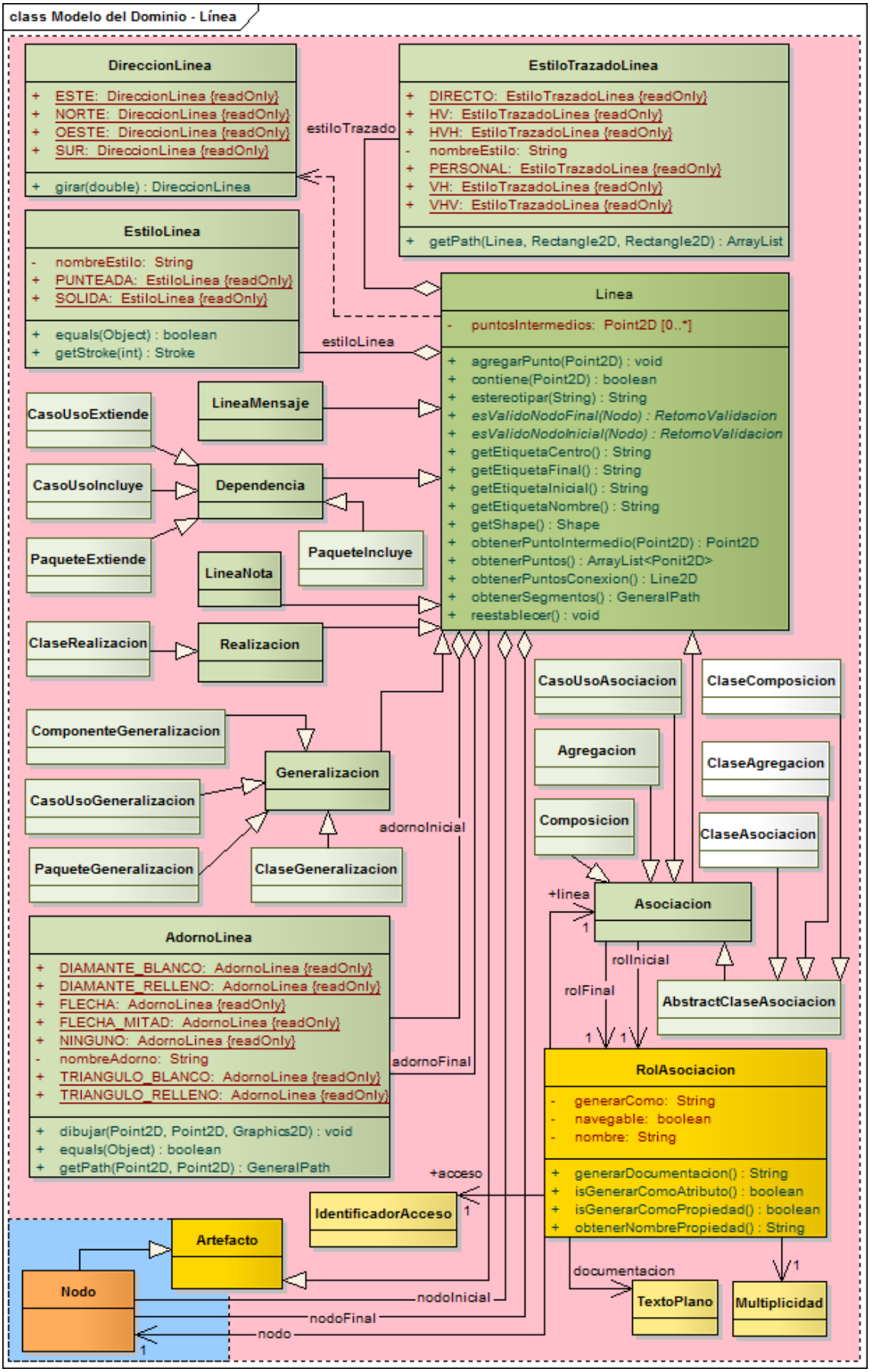


Figura 68: DC Linea

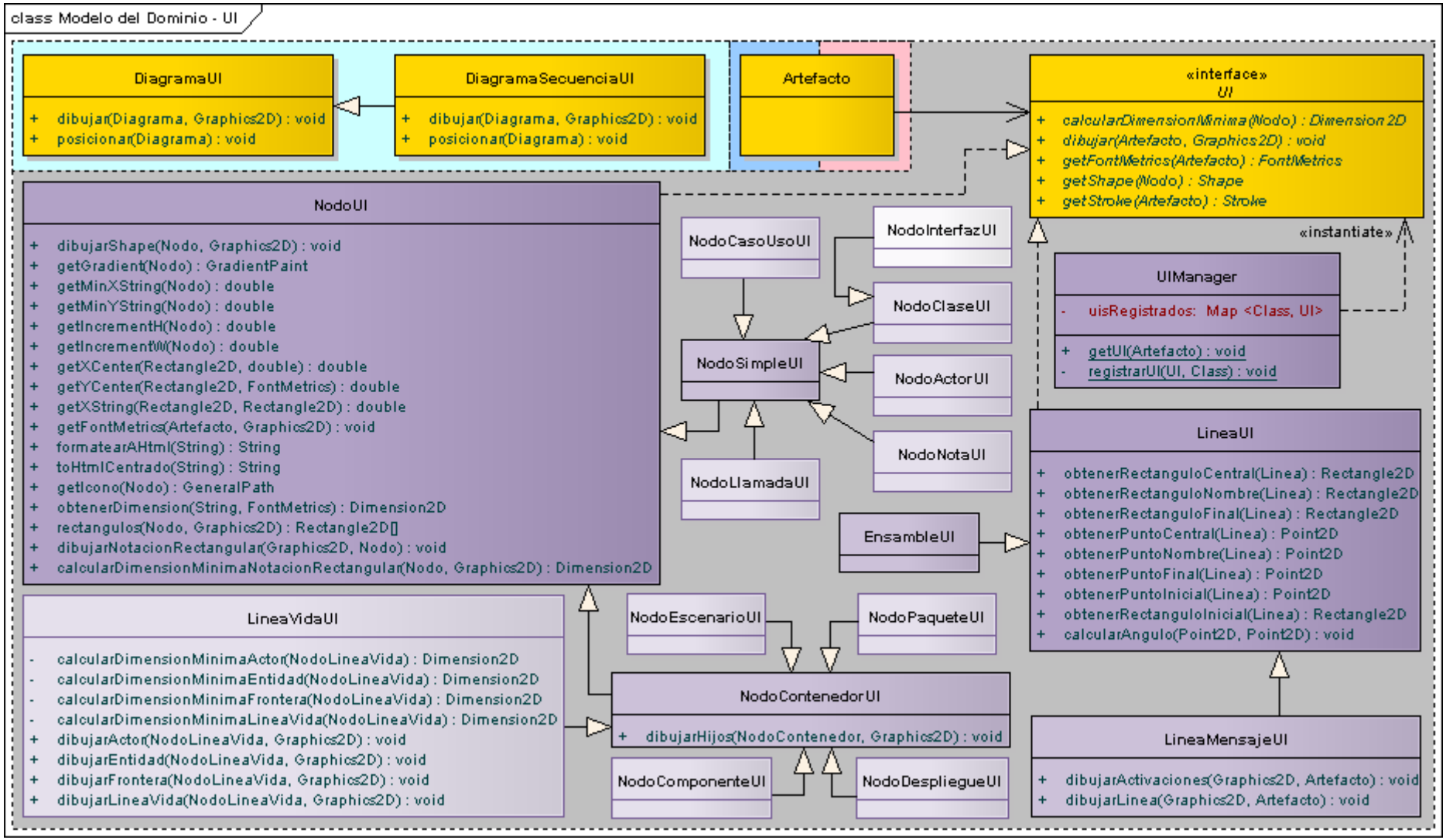


Figura 69: DC UI

8.2.4.2. Diagrama de Componentes

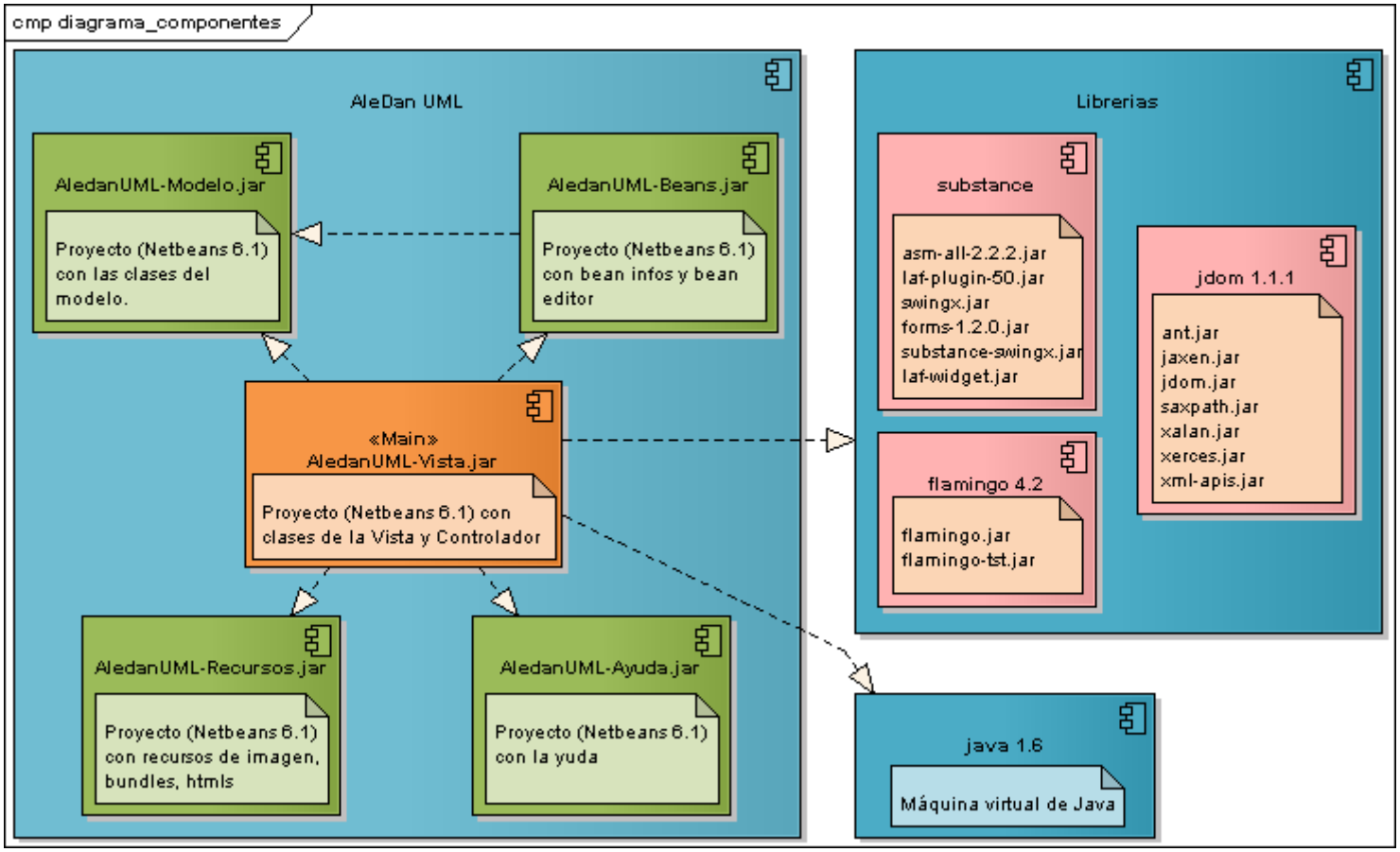


Figura 70: Diagrama de Componentes

8.2.4.3. Diagrama de Despliegue

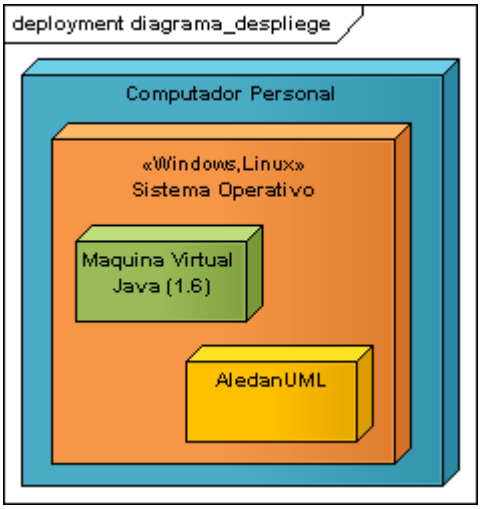


Figura 71: Diagrama de Despliegue

8.2.4.4. Diagrama de Paquetes

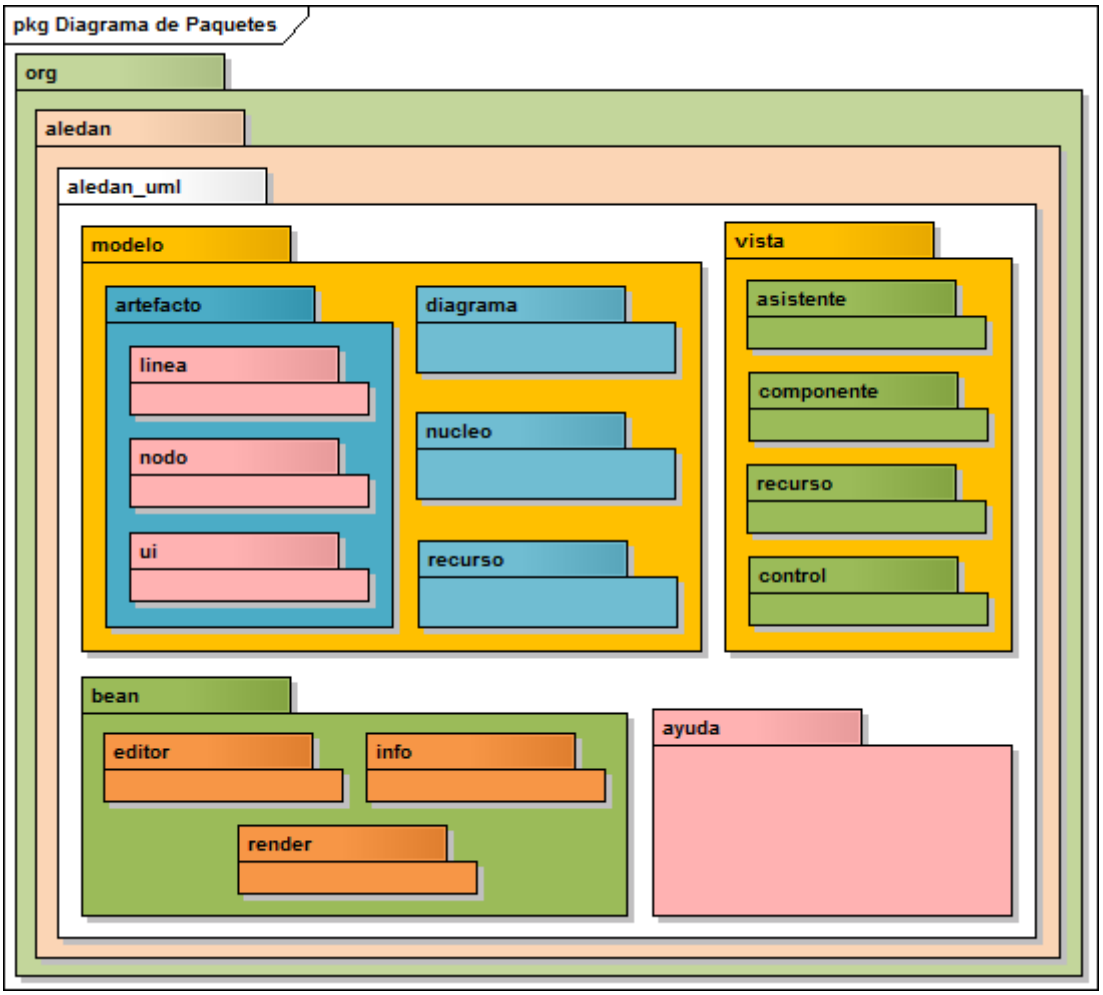


Figura 72: Diagrama de Paquetes

8.2.5. Definición de la arquitectura del sistema

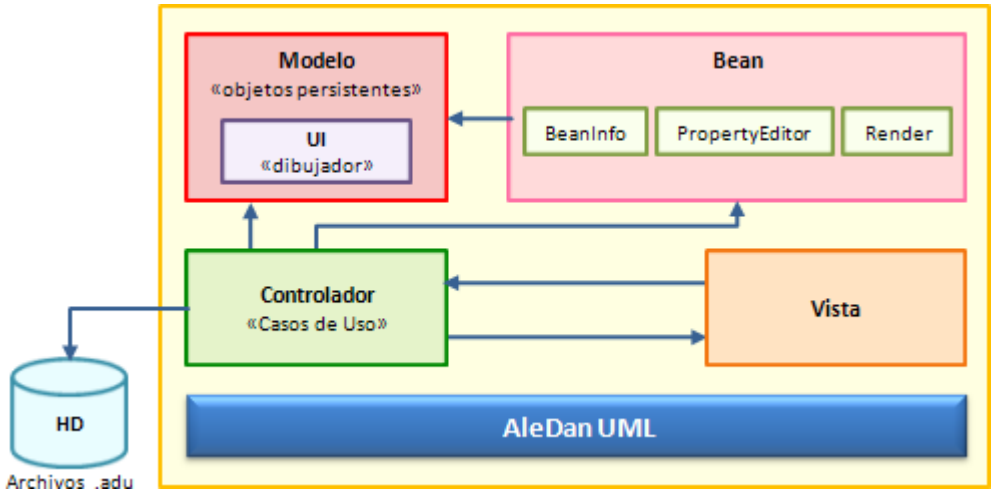


Figura 73: Arquitectura del sistema

La arquitectura establecida para la construcción de la herramienta **AleDan UML** se basa en el desarrollo de software por capas, tal y como se indica en los puntos detallados a continuación:

- Se utiliza una arquitectura de cuatro capas claramente identificadas: Modelo, Bean, Controlador y Vista.
- La capa **Modelo** es la que tiene los objetos persistentes del dominio, es decir, encierra a las entidades. Además de ello cuenta con la capa *UI*, misma que se encarga de dibujar todos los objetos de modelo.
- **Bean** es la capa dirigida a la manipulación o edición de los objetos persistentes del **Modelo**. Esta capa a su vez provee las subcapas *BeanInfo*, *PropertyEditor* y *Render*.
  - *BeanInfo* es la que describe la información de un objeto persistente indicando las propiedades que posee y los métodos de lectura y escritura para dichas propiedades;
  - *PropertyEditor* son editores que pueden ser asociados a una propiedad determinada de un objeto persistente, o bien, a una clase de objetos;
  - *Render* por su parte, es la que permite representar gráficamente un tipo de dato.
- La capa **Controlador**, es la responsable de dar funcionalidad a la **Vista**, interactúa con el **Modelo** los archivos de proyecto *.adu en una unidad de almacenamiento, y conjuntamente con los **Bean**, carga las propiedades en las vistas.

- Finalmente la capa **Vista** permite al usuario la manipulación y edición de un proyecto AleDan UML *.adu.

8.2.6. Glosario de términos

Tabla 68: Glosario de Términos

Término	Descripción
Accede	En un diagrama de Paquetes, relación de dependencia entre paquetes en donde los elementos importados tienen visibilidad privada en el paquete cliente y no pueden usarse fuera de él, incluyendo cualquier otro paquete que los importe.
Adorno Línea	Elemento gráfico que representa el inicio o fin de un tipo de relación para diferenciarla de otra.
Agregación	Tipo de Asociación que representa una relación entre un todo y sus partes.
Artefacto	Elemento gráfico que representa a un nodo dentro de un diagrama.
Asociación	Relación semántica entre dos o más clasificadores que involucra conexiones entre sus instancias.
Atributo	Característica propia que identifica a una entidad u objeto.
Clase	Tipo de Clasificador que representa cualquier objeto o entidad de la realidad que se desea modelar.
Clasificador	Mecanismo de UML que describe las características estructurales y de comportamiento de objetos o entidades.
Composición	Tipo de Asociación que especifica que cada componente pertenece solamente a un todo.
Dependencia	Relación semántica entre dos objetos, en donde un cambio a un objeto puede afectar la semántica del otro objeto.
Diagrama	Representación gráfica de un sistema o subsistema. Es un conjunto de vértices y arcos.
Diagrama Casos de Uso	Muestra un conjunto casos de uso, actores y sus relaciones. Ayuda a documentar el comportamiento que tiene el sistema actual o el comportamiento que va a tener el sistema a desarrollar, a través de la identificación de los actores y escenarios que se van a presentar.
Diagrama Clases	Muestra un conjunto de clases, interfaces, colaboraciones y sus relaciones.
Diagrama	Muestra un conjunto de componentes y sus relaciones, utilizado para

Componentes	modelar código fuente, código ejecutable y bases de datos físicas.
Diagrama Comportamiento	Describen lo que debe suceder en el sistema modelado. Se usan para visualizar, especificar, construir y documentar los aspectos dinámicos de un sistema en un determinado periodo de tiempo.
Diagrama Despliegue	Ayudan a visualizar la configuración física del sistema en tiempo de ejecución, a través de nodos que muestran la disposición tanto en hardware como en software de la aplicación.
Diagrama Estructural	Ayudan a visualizar, especificar, construir y documentar los aspectos estáticos de un sistema, es decir, no indican los cambios que van a tener los objetos durante su ejecución.
Diagrama Paquetes	Indica gráficamente la organización de los paquetes y de sus clases. Muestra cómo un sistema está dividido en agrupaciones lógicas y las dependencias entre esas agrupaciones.
Diagrama Secuencia	Representa la secuencia ordenada y lógica que se da entre los objetos de un caso de uso dentro de un escenario específico. Se enfoca en ordenar los mensajes de acuerdo a su tiempo de envío.
Enumeración	"Tipo especial de estructura en la que los literales de los valores que pueden tomar sus objetos se indican explícitamente al definirla"[27].
Estilo Línea	Representación gráfica del estilo de línea de una relación. Por ejemplo sólida, punteada, etc.
Estilo Trazado Línea	Representación gráfica del trazo de línea de una relación. Por ejemplo horizontal, vertical, etc.
Extensión	En un diagrama de Casos de Uso, tipo de Dependencia que se define como la agregación de pasos a la secuencia de un caso de uso base. Significa que el caso de uso base implícitamente incorpora el comportamiento de otro caso de uso.
Generalización	Tipo de relación en donde el elemento especializado (el hijo) comparte la estructura y la conducta del elemento generalizado (el padre).
Importación	En un diagrama de Paquetes, tipo de Dependencia en donde los elementos importados tienen visibilidad pública en el paquete cliente.
Inclusión	En un diagrama de Casos de Uso, tipo de Dependencia que significa que el caso de uso base usa explícitamente el comportamiento de otro caso de uso en un momento específico.
Interfaz	Colección de operaciones usada para especificar un servicio de una clase o de un componente. UML define dos tipos de interfaz: Provista y

[27] CHARTE OJEDA, Francisco. 2003. Programación C# .NET. España. ANAYA. 165 p.

	Requerida.
Línea	Representación gráfica de una relación. Una relación se define como una conexión entre objetos que muestra la manera como interactúa cada uno de ellos.
Línea Mensaje	Indica la comunicación existente entre dos líneas de vida.
Línea Nota	Asocia una nota a un elemento dentro de un diagrama.
Método	Describe el comportamiento y funcionalidad de una entidad u objeto.
Namespace	Es un elemento que contiene un conjunto de elementos nombrados.
Nodo	Representación gráfica de las diferentes entidades u objetos que conforman a cada tipo de diagrama. Por ejemplo: clase, actor, nota, etc.
Nodo Actor	Artefacto que representa cualquier entidad que puede interactuar con un caso de uso.
Nodo Caso Uso	Artefacto que representa una narración hablada o escrita del comportamiento que tiene o que va a tener un escenario.
Nodo Clase	Artefacto utilizado para representar una clase.
Nodo Componente	Artefacto utilizado para representar un componente.
Nodo Contenedor	Tipo de Nodo que agrupa a todos los artefactos que pueden ser contenedores de otros artefactos.
Nodo Despliegue	Artefacto utilizado para representar despliegues físicos de hardware, software o componentes
Nodo Escenario	Artefacto utilizado para representar un escenario.
Nodo Etiqueta	Artefacto utilizado para representar una etiqueta.
Nodo Interfaz	Artefacto utilizado para representar una interfaz.
Nodo Línea Vida	Artefacto utilizado para representar una línea de vida.
Nodo Línea Vida Actor	Artefacto utilizado para representar una línea de vida especificada como un objeto actor.
Nodo Línea Vida Entidad	Artefacto utilizado para representar una línea de vida especificada como un objeto entidad.
Nodo Línea Vida Frontera	Artefacto utilizado para representar una línea de vida especificada como un objeto frontera.
Nodo Nota	Artefacto utilizado para representar una nota.
Nodo Paquete	Artefacto utilizado para representar un paquete.
Nodo Simple	Tipo de Nodo que agrupa a todos los artefactos que no son contenedores de otros.

Objeto de Modelo	Agrupa clases, interfaces, enumeraciones o tipos de dato, puesto que estos elementos contienen información necesaria para la generación de código fuente.
Parámetro	Información de entrada que ingresa a un método.
Proyecto	Conjunto de diagramas y objetos del modelo.
Realización	Relación semántica entre clasificadores, en donde un clasificador especifica un convenio que otro clasificador garantiza llevar a cabo.
Tipo de Dato	Tipo especial de clasificador similar a una clase, con la diferencia de que sus instancias son identificadas únicamente por su valor. Comúnmente es usado para representar tipos primitivos en los lenguajes de programación (integer, string, etc.).
Tipo Línea Vida	Representa gráficamente los diferentes tipos de objeto que puede adoptar una línea de vida.

8.3. CONSTRUCCIÓN

8.3.1. Definición de la plataforma de desarrollo

Para el desarrollo de la Herramienta AleDan UML se ha establecido que la Plataforma de Desarrollo a usar sea la siguiente:

- **Sistema Operativo.-** Ubuntu Ultimate v1.9 puesto que es la distribución Linux que a nuestro criterio, ofrece las mejores facilidades de uso frente a otros similares. Su utilización permite abaratar los costos de adquisición y producción de nuestro software, pero lo que es más importante, se trata de impulsar el empleo de software libre no sólo en la Universidad Nacional de Loja, sino también en la sociedad en general, para solucionar el problema de licenciamiento del software y desalentar la práctica de la muy conocida “piratería”.

- **Lenguaje y entorno de programación.-** Plataforma Java™, Standard Edition Development Kit (JDK™) v1.6.0_03, debido a que Java es multiplataforma, portable, versátil, eficiente, orientado a objetos y es el lenguaje de programación mejor conocido por los desarrolladores.

- **Entorno de desarrollo.-** IDE de Programación NetBeans v6.7.1, ya que es una herramienta lo suficientemente estable y robusta para la realización de trabajos de este tipo.

- **Control de versiones.-** CVS, pues al darnos la posibilidad de guardar y recuperar las versiones anteriores de nuestro proyecto, minimiza el riesgo de problemas asociados a cambios inapropiados. Además este sistema permite que dos o más desarrolladores trabajen sobre el mismo archivo, facilitando la construcción de software en equipo.

8.3.2. Definición de las políticas de programación

Durante el proceso de codificación de la herramienta AleDan UML se utilizó como referencia las convenciones de escritura para el lenguaje de programación Java:

- **Nombre de paquetes.-** Escrito totalmente en minúsculas, está constituido por varios componentes separados por un punto. El primer componente del nombre del paquete es el nombre de dominio de último nivel: **org**. El segundo componente lo conforma el nombre del grupo de trabajo **aledan.** A continuación se ubica el nombre del proyecto **aledan_uml**, para luego posicionar los paquetes propios del proyecto, entre los que se destacan **bean**, **modelo**, **recurso** y **vista**.

- **Nombre de clases e interfaces.-** Usarán sustantivos simples y descriptivos, en donde la primera letra estará en mayúscula para continuar con minúsculas. En caso de que el nombre de la clase esté compuesta por más de una palabra, éstas serán concatenadas usando mayúsculas para la letra de inicio de cada una de ellas. Cuando se utilicen siglas iniciará con las siglas en mayúsculas siguiendo la política anteriormente señalada.

- **Nombre de métodos.-** Usarán verbos en infinitivo que simbolicen la acción para la que fueron creados. En cuanto a su escritura, seguirán la misma regla establecida para las clases, con la diferencia de que la letra inicial del nombre del método será minúscula.

- **Nombre de variables.-** Usarán palabras simples y significativas escritas en minúscula. En caso de estar compuesto por más de una palabra, éstas serán

concatenadas usando mayúsculas para la letra inicial de cada palabra exceptuando la primera. Para componentes swing se utiliza la siguiente nomenclatura:

- txt[Nombre] para campos de texto
- lbl[Nombre] para etiquetas significativas
- cmb[Nombre] para listas desplegables
- btn[Nombre] para botones
- panel[Nombre] para paneles
- tabla[Nombre] para tablas

- **Nombre de constantes.-** Irán totalmente en mayúsculas separando las palabras con un guión bajo ("_") en caso de que el nombre esté compuesto por más de una palabra.

- **Documentos XML.-** Utilizados para generar el proyecto para el IDE Visual Studio o MonoDevelop.

- **Documentos HTML.-** Utilizados para las ayudas desplegadas dentro de la herramienta.

- **Formato de archivos gráficos.-** Los archivos gráficos utilizados dentro de la aplicación tienen el formato *.png, *.gif, *.jpg y *.svg.

8.3.3. Implementación

El proceso de construcción de la Herramienta AleDan UML se realizó en las 3 iteraciones descritas a continuación:

- **Iteración 1.-** Codificación del módulo de Modelo – módulo de Beans – módulo de Vistas – Pruebas de verificación
- **Iteración 2.-** Mejora y corrección de los módulos antes mencionados en base a los resultados obtenidos en las pruebas unitarias – Pruebas de verificación
- **Iteración 3.-** Refinamiento – construcción de módulo de Ayuda – Manuales – Instalador – Pruebas de verificación – Pruebas de validación

Los documentos resultantes de la elaboración de manuales de usuario y programador, se trabajaron por separado al presente informe.

8.3.4. Pruebas de verificación

Se entiende como Verificación al proceso mediante el cual los desarrolladores prueban el software para intentar encontrar defectos ejecutando el programa en un ambiente de prueba.

- **Pruebas de Unidad.-** Estas pruebas informales consistieron en probar parcialmente los módulos que conforman el programa: cada vez que se hacían modificaciones o que se incluían nuevas funcionalidades, el desarrollador se encargaba de compilar y ejecutar el programa para comprobar su correcto funcionamiento. Los principales resultados obtenidos a lo largo de la ejecución de estas pruebas fueron los siguientes:

Tabla 69: Pruebas de unidad

Meta / Vista	Error	Estado
MT01 Crear proyecto	Cuando se crea un nuevo proyecto agregarlo a la lista de proyectos recientes	OK
MT03 Abrir proyecto MT07 Guardar proyecto como	Cuando un proyecto se guarda como, y se trata de abrir nuevamente, se presenta un mensaje de que ya está abierto	OK
MT03 Abrir proyecto	Presentar en varias líneas el mensaje de error que aparece cuando no se puede encontrar un archivo, porque es demasiado ancho en una sola línea	OK
MT04 Renombrar proyecto	Se puede duplicar el nombre de un proyecto cuando se lo digita letra a letra	OK
MT05 Administrar objeto de modelo	No se debe eliminar el namespace creado por defecto dentro de un proyecto	OK
	Implementar mensaje de aviso cuando no se ha podido renombrar un objeto de modelo porque el campo de texto está vacío	OK
MT06 Guardar proyecto	Un proyecto se guarda a pesar de no haber realizado cambios en él	OK
MT09 Eliminar proyecto	Los proyectos eliminados no se retiran de la Lista Proyectos Recientes	OK
MT10 Crear diagrama	Se puede duplicar el nombre de un diagrama	OK
	No se presenta mensaje de información cuando no se	OK

	ingresa el nombre de un diagrama	
MT11 Imprimir diagrama	Implementar la opción “Imprimir diagrama”	OK
MT13 Editar diagrama	Inconsistencia de valor zoom con lo graficado	OK
	Cuando se incremente el zoom no aparecen barras de desplazamiento para movilizarse por el diagrama	OK
MT14 Exportar diagrama en formato *.png	Implementar mensaje de aviso cuando se duplica el nombre de un archivo *.png al exportar un diagrama	OK
	Se puede duplicar el nombre de un archivo en formato *.png cuando se lo digita letra a letra al exportar un diagrama	OK
	Cuando se exporta un diagrama en formato *.png no asigna nombre por defecto	OK
MT17 Agregar artefacto	No se debe permitir relación de dependencia para los nodos nota y etiqueta. Sólo deben aceptar vínculos de nota.	OK
	Al seleccionar un artefacto propio de un tipo de diagrama en la Paleta y cambiar a otro tipo de diagrama, el artefacto se crea sin ningún problema	OK
MT18 Editar artefacto	Se permite duplicar el nombre de nodo Clase, Interfaz, Enumeración y Tipo de dato	OK
	Se permite ingresar caracteres no permitidos para el nombre de un tipo de dato	OK
	Mejorar evento para seleccionar artefactos creados en un diagrama	OK
	Optimizar repintado del área de diagrama cuando se elimina, pega o edita un artefacto	OK
	El texto de los artefactos se grafica fuera del área de dibujo	OK
	Quitar la pestaña de valores etiquetados en las propiedades de una nota	OK
	Cuando se selecciona un nodo permanecen activas las propiedades para formato de una relación	OK
	Implementar la opción “Cortar”	OK
	Se debería permitir arrastrar las líneas de relación ***Nota.-*** *Las líneas se auto ubican. La personalizada se la puede mover agregando un punto adicional presionando [Ctrl + clic]*	OK
	No se pueden mover varios artefacto a la vez	OK
MT19 Administrar atributo	No se controlan excepciones cuando se trata de eliminar atributos que no existen	OK
	En la administración de atributos agregar campos para declararlos como finales y estáticos	OK
	Limpiar los campos de un atributo en el panel de edición después de eliminarlo	OK
	Se permite duplicar el nombre de atributos	OK
MT20	Limpiar los campos de un método en el panel de edición	OK

Administrar método	después de eliminarlo	
	No se controlan excepciones cuando se trata de eliminar métodos que no existen	OK
MT21 Administrar mecanismo de extensibilidad	Cuando no se llenan los mecanismos quitar las filas vacías	OK
	Validar que los mecanismos de extensibilidad un nombre no quede sin valor o viceversa	OK
MT22 Eliminar artefacto	No se pueden eliminar varios artefacto a la vez	OK
Árbol de proyectos / Panel de edición	Los menús emergentes no se dibujan correctamente en algunas ocasiones	OK
Panel de Inicio	Mostrar únicamente el nombre del archivo y no toda la ruta en la Lista de proyectos recientes del panel de Inicio	OK
Tarea Formato	Quitar icono de sombreado de la Tarea Formato, Banda Texto porque es redundante	OK

8.4. TRANSICIÓN

Al finalizar la etapa de construcción de la versión 1.0 del producto AleDan UML se procedió a crear el instalador compatible con los sistemas operativos Windows y Linux. Este producto se encuentra en la carpeta Instalador del CD adjunto.

8.4.1. Descripción de la herramienta (versión 1.0 al 12 de julio de 2010)

AleDan UML se constituye en una herramienta CASE multiplataforma para el análisis y diseño de sistemas informáticos, que se basa en las especificaciones de UML 2.0 y del Lenguaje de Programación C Sharp.

Esta herramienta ofrece a los desarrolladores de software la posibilidad de utilizar un asistente gráfico para elaborar la documentación de sus aplicaciones a través del diseño de los seis diagramas UML más utilizados en nuestro medio: casos de uso, paquetes, clases, componentes, despliegue y secuencia.

Adicionalmente el usuario puede generar automáticamente el código fuente para el lenguaje de programación C Sharp, esto a partir de los diagramas de clases y secuencia elaborados previamente por el analista.

AleDan UML cuenta también con una interfaz gráfica amigable, interactiva e intuitiva que permite que el trabajo realizado por analistas y desarrolladores de software sea

mucho más rápido y eficiente, puesto que esta herramienta se construyó pensando en controlar al máximo los posibles errores cometidos a la hora de diagramar. Si a ello sumamos los pocos recursos computacionales que consume además de muchas otras excelentes características, estamos hablando de una aplicación lo suficientemente estable para competir con las ya existentes en el mercado.

8.4.2. Etapa de validación

La Validación puede considerarse como el proceso mediante el cual el software final se prueba como un todo para comprobar si cumple los requisitos funcionales y de rendimiento, facilidad de mantenimiento, recuperación de errores, etc., en un ambiente real. Tiene lugar cuando la verificación está completa.

- **Prueba de rendimiento.-** Se probó los tiempos de respuesta, el espacio que ocupa la herramienta en disco, etc.

- **Prueba de robustez.-** Se comprobó la capacidad del programa para soportar entradas incorrectas (controles).

- **Prueba de compatibilidad.-** Se evaluó el desempeño del software en diferentes hardwares y sistemas operativos.

- **Prueba de Aceptación.-** Fue ejecutada en el entorno real de operación del sistema. Esta prueba estuvo encaminada a comprobar que se haya dado completa cobertura a la especificación de requerimientos.

- **Prueba de Usabilidad.-** Su objetivo fue estudiar la usabilidad del programa en su entorno real con usuarios reales. Para ello se evaluaron aspectos tales como la facilidad de uso del sistema, su robustez, su interfaz gráfica, etc.

8.4.2.1. Diseño del plan de pruebas

Como punto de partida se identificó a los usuarios que participarían en el proceso de evaluación. El universo poblacional a considerar en ésta fase del proyecto, lo constituyen los desarrolladores de software de la Universidad Nacional de Loja y de la empresa DIVUSWARE Cía. Ltda.:

Tabla 70: Población

Población (N)		
Carrera de Ingeniería en Sistemas (periodo marzo – julio de 2010)	Segundo módulo	100
	Cuarto módulo	104
	Sexto módulo	56
	Octavo módulo	49
	Décimo módulo	61
DIVUSWARE (a julio de 2010)	–	4
TOTAL		**374**

Fuente: Secretaría General del Área de la Energía, las Industrias y los Recursos Naturales no Renovables y Gerencia general de DIVUSWARE Cía. Ltda.

El tamaño de la muestra se pudo determinar mediante la siguiente fórmula estadística[28]:

$$n = \frac{Z^2 pqN}{NE^2 + Z^2 pq}$$

Donde:

- n es el tamaño de la muestra = ?
- Z es el nivel de confianza = 90% = 1,65
- p es la variabilidad positiva = 0,84
- q es la variabilidad negativa = 0,16
- N es el tamaño de la población = 374
- E es el margen de error = 10% = 0,1

En base a ello se pudo determinar lo siguiente:

$$n = \frac{(1{,}65)^2(0{,}84)(0{,}16)(374)}{(374)(0{,}1)^2 + (1{,}65)^2(0{,}84)(0{,}16)} = \frac{136{,}848}{4{,}106} = 33{,}329 \sim 34$$

Por lo tanto, el tamaño de la muestra es de 34 desarrolladores de software.

Como siguiente paso se procedió a definir el alcance de la prueba y los puntos estratégicos que deberían ser incluidos dentro de la encuesta para obtener información de relevancia e interés que permita mejorar la calidad y funcionalidad de la herramienta y consecuentemente, el nivel de satisfacción de los usuarios. Se

[28] Revisar Tamaño de muestras en el Anexo 1 de la página 191.

estipularon cuáles serían los criterios de suspensión, reanudación y finalización del sondeo, la documentación resultante del mismo y por supuesto, los recursos necesarios para efectuarlo. Luego de ello se determinó el lugar físico en el que la prueba se llevaría a cabo y finalmente, se estableció el calendario para ejecutar el Plan de Validación y su responsable.

Al término de ésta actividad se obtuvo el plan de pruebas descrito en la Tabla 71:

Tabla 71: Plan de Pruebas

Plan de Pruebas	
Identificador:	PP01
Evaluadores:	▪ 30 estudiantes de la Carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja. ▪ 4 desarrolladores de software que se desempeñan profesionalmente en la empresa DIVUSWARE Cía. Ltda. de la ciudad de Loja.
Alcance:	Se probará seguridad, rendimiento, robustez, confiabilidad, integración de módulos, presentación de datos, instalación, compatibilidad, aceptación del usuario y usabilidad.
Focos de Prueba:	▪ Módulo de modelo ▪ Módulo de diagramación ▪ Módulo de generación de código fuente ▪ Módulo de ayuda
Estrategia:	Análisis de entradas y salidas.
Criterios de suspensión:	▪ Falta de computador ▪ Falta de sistema operativo ▪ Falta de la herramienta AleDan UML ▪ Detección de errores que no permitan continuar
Criterios de reanudación:	La prueba se reanudará cuando el o los criterios de suspensión hayan sido superados.
Criterios de finalización:	La prueba no se volverá a ejecutar y se considerará terminada si se cumplen todos los requerimientos y los errores no requieran nueva revisión.
Documentación:	Se realizará un cuadro con los resultados de las pruebas y las correcciones realizadas.
Recursos:	▪ Computador ▪ Sistema operativo Windows / Linux ▪ Instalador de AleDan UML ▪ Encuesta ▪ Personal
Calendario:	12 de julio de 2010
Local:	▪ Centro de cómputo 1.3 del Área de la Energía, las Industrias y los Recursos Naturales no Renovables ▪ Instalaciones de DIVUSWARE Cía. Ltda.

Responsable:	Alexandra Maurad

8.4.2.2. Ejecución del plan de pruebas

La etapa de validación de la herramienta AleDan UML fue realizada de acuerdo a lo establecido en el Plan de Pruebas, de la siguiente manera:

En primer lugar se proporcionó a los usuarios el instalador de AleDan UML compatible con los sistemas operativos Windows y Linux para que procedieran a su instalación en los equipos respectivos. Paralelamente se distribuyó a cada participante un documento con la Encuesta 02[29] para que documenten los resultados y apreciaciones respecto a la herramienta, durante y después de utilizar el sistema.

Al término de la prueba, los evaluadores entregaron al equipo de desarrollo los resultados alcanzados, mismos que incluían deficiencias encontradas y algunas observaciones y sugerencias para mejorar al máximo la herramienta antes de ser aprobada definitivamente. De ésta manera se pudo establecer una retroalimentación entre los usuarios y el equipo de desarrollo.

8.4.2.3. Análisis de Resultados del plan de pruebas

Luego del análisis de la información recolectada en la etapa de pruebas de la herramienta AleDan UML, los resultados obtenidos fueron los siguientes:

Tabla 72: Resultados alcanzados en la etapa de pruebas y validación de la Herramienta AleDan UML

Funcionalidad	Rango			
	Excelente	Bueno	Regular	Malo
Interfaz gráfica de usuario amigable	20	15	0	0
Administración de proyectos	18	17	0	0
Organización y distribución de “Navegador de Proyectos”	17	17	1	0
Administración de diagramas UML	24	11	0	0
Administración de artefactos	20	13	2	0
Edición y presentación de propiedades gráficas y estructurales de un artefacto	18	17	0	0

[29] El modelo utilizado para la Encuesta 02 se encuentra en el Anexo 2 de la página 193.

Control de entradas erróneas	17	13	5	0
Presentación de mensajes informativos, de control, de error, etc.	14	20	1	0
Exportación de diagramas en formato *.png	17	15	3	0
Impresión de diagramas	20	13	2	0
Ayuda de la herramienta	23	11	1	0
Coherencia entre diseño de diagramas y código generado	18	16	0	1
Tiempo de respuesta de la herramienta en la ejecución de tareas	25	9	1	0
Desempeño de la herramienta en cuanto al uso de recursos computacionales	21	13	1	0
Satisfacción del usuario	17	18	0	0

Como se puede apreciar en la Tabla 72 la gran mayoría de resultados obtenidos estuvieron encaminados a los rangos “Excelente” y “Bueno”, puesto que los usuarios encuestados mostraron una gran acogida por la herramienta probada. Y es precisamente gracias a esos resultados, que se pudo determinar que la herramienta de análisis y diseño AleDan UML cumple satisfactoriamente con cada uno de los requerimientos del sistema establecidos en la etapa inicial del proceso investigativo.

9. VALORACIÓN TÉCNICO – ECONÓMICA – AMBIENTAL

Al culminar con el proyecto investigativo denominado **"Desarrollo de una Herramienta de análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp (ECMA – 334)"**, podemos determinar que efectivamente se cumplieron los objetivos establecidos al inicio de la investigación, dando como resultado el producto software **AleDan UML**, mismo que cumple satisfactoriamente los requerimientos funcionales y no funcionales planteados por los desarrolladores de software de nuestro medio.

El desarrollo de esta aplicación fue posible gracias a la utilización de recursos hardware y software de actualidad, mismos que fueron conseguidos a través de la inversión personal de los tesistas. Sin embargo, es importante destacar que la mayoría de herramientas empleadas son de libre distribución, lo que hizo posible obtenerlas a través de la Internet sin necesidad de incurrir en gastos excesivos.

Los costos reales asumidos por los aspirantes en el desarrollo de la herramienta **AleDan UML** son los especificados a continuación:

9.1. RECURSOS MATERIALES

Tabla 73: Recursos materiales

Recursos materiales			
Material	**Cantidad**	**Unidad**	**Valor Total**
Anillado	10	$ 2,00	$ 20,00
Carpetas de perfil	3	$ 0,60	$ 1,80
Empastado	3	$ 10,00	$ 30,00
Material de oficina	-	$ 50,00	$ 50,00
Resma de papel Inen A4	10	$ 4,00	$ 40,00
Recarga genérica de tinta a color para impresora	4	$ 12,50	$ 50,00
Recarga genérica de tinta negra para impresora	8	$ 6,00	$ 48,00
Infocus	2	$ 15,00	$ 30,00
SUBTOTAL			$ 269,80

9.2. SERVICIOS BÁSICOS

Tabla 74: Servicios básicos

Servicios básicos		
Servicio	**Valor por mes**	**Valor Total**
Luz	$ 20,00	$ 260,00
Teléfono	$ 10,00	$ 130,00
Transporte	$ 25,00	$ 325,00
SUBTOTAL		$ 715,00

9.3. RECURSOS TÉCNICOS Y TECNOLÓGICOS

Tabla 75: Hardware

Hardware					
Equipo	**Costo**	**Costo Residual**	**Vida Útil (años)**	**Uso (días)**	**Valor**
Computadora de escritorio	$ 1000,00	$ 100,00	10 años	297	$ 74,23
Computadora de escritorio	$ 1000,00	$ 100,00	10 años	297	$ 74,23
Flash Memory	$ 18,00	$ 5,00	2 años	150	$ 2,72
Impresora	$ 60,00	$ 20,00	3 años	30	$ 1,11
SUBTOTAL					$ 152,29

Tabla 76: Software

Software	
Aplicación	**Valor Total**
SO: Windows XP Professional Versión 2002 compilación 6200	$ 143,77
SO: Ubuntu Ultimate Edition 8.1	$ 000,00
Herramienta para la Planificación: Microsoft Office Project 2003	$ 120,00
Herramienta para la Documentación: OpenOffice 3.0	$ 000,00
Lenguaje de Programación: Java	$ 000,00
Lenguaje de Programación: C#	$ 000,00
Herramienta para codificación: NetBeans 6.7.1	$ 000,00
Herramienta para diseño de arquitectura: Enterprise Architect 7.1	$ 135,00
Herramienta para diseño gráfico: Gimp	$ 000,00
SUBTOTAL	$ 398,77

Tabla 77: Comunicaciones

Comunicaciones			
Medio	**Cantidad**	**Valor Unitario**	**Valor Total**
Internet	60	$ 0,80	$ 48,00
SUBTOTAL			$ 48,00

Tabla 78: Recursos técnicos y tecnológicos

Recursos técnicos y tecnológicos	
Recurso	**Valor Total**
Hardware	$ 152,29
Software	$ 398,77
Comunicaciones	$ 48,00
SUBTOTAL	$ 599,06

9.4. RESUMEN DE COSTOS

Tabla 79: Resumen de costos

Resumen de costos	
Recurso	**Valor Total**
Recursos Humanos	$ 000,00
Recursos Materiales	$ 269,80
Servicios Básicos	$ 715,00
Recursos Técnicos y Tecnológicos	$ 599,06
SUBTOTAL	$ 1.583,86
IMPREVISTOS (5 %)	$ 79,19
TOTAL	$ 1.663,05

10. CONCLUSIONES

Una vez finalizado el desarrollo de la herramienta de análisis y diseño AleDan UML se han podido establecer las siguientes conclusiones:

- El estudio de las herramientas CASE de análisis y diseño de sistemas más utilizadas por los desarrolladores de software, permitió capturar los requerimientos que la herramienta AleDan UML debería satisfacer.

- Se desarrolló un módulo de modelado que permite realizar diagramas de clases, paquetes, componentes, despliegue, casos de uso y secuencia, para documentar software bajo las especificaciones de UML 2.0 con soporte para el lenguaje de programación C Sharp.

- Se desarrolló un módulo de generación de código fuente con soporte para el Lenguaje de Programación C Sharp a partir de los modelos previamente diseñados.

- La aplicación de la metodología RUP y de las herramientas seleccionadas para la construcción de la aplicación, permitió agilizar el proceso de desarrollo del proyecto para alcanzar los objetivos planteados.

- La ejecución de pruebas de validación nos permitió confirmar que AleDan UML cumple satisfactoriamente los requerimientos y objetivos establecidos al inicio del proyecto.

11. RECOMENDACIONES

Las recomendaciones planteadas al término del presente proyecto son las siguientes:

- Para el desarrollo de software, por más pequeño que éste sea, se recomienda realizar el análisis y diseño con una herramienta que ofrezca un entorno amigable, interactivo e intuitivo, de manera que el proceso no sea demasiado tedioso.

- Incentivar a los estudiantes de la Carrera de Ingeniería en Sistemas de nuestra alma mater, el uso de la herramienta AleDan UML para el desarrollo de sus proyectos de software, porque es un producto obtenido en base a los requerimientos de nuestro entorno, bajo los estándares de UML 2.0 y ECMA – 334.

- Se recomienda que AleDan UML, a más de constituirse en una herramienta que ayude en el proceso de enseñanza – aprendizaje, se convierta en un punto de partida no sólo para el mejoramiento de sus funcionalidades, sino también, para el desarrollo de nuevos y mejores proyectos de investigación.

- Se recomienda leer detenidamente el manual de usuario, de tal manera que se logre aprovechar al máximo todas las ventajas que la herramienta AleDan UML ofrece.

- Para el desarrollo de éste tipo de aplicaciones se requiere trabajar en equipo, por lo que se recomienda la utilización de herramientas que soporten el uso y edición compartido de archivos como subversión, conexión de red, impresoras compartidas, entre otras.

- Para mejorar la apariencia de las aplicaciones java se recomienda la investigación y uso de nuevas librerías libres y multiplataforma como “substance look and feel” y “falmingo”.

12. BIBLIOGRAFÍA

Libros

- CHARTE OJEDA, Francisco. 2003. Programación C#.NET. España. Anaya.

Sitios web:

- SOTO MADRIGAL, Alejandro. 2007. Colecciones de Datos. http://alejandrocr.blogspot.com/2007/06/colecciones-de-datos.html, 03 de febrero del 2009.
- BARROS JUSTO, José Luis. 2006. Herramientas CASE. http://creaweb.ei.uvigo.es/creaweb/Asignaturas/DAS/apuntes/Teoria%20semana%209.pdf, 27 de noviembre del 2008.
- Herramientas CASE. 2008. Wikipedia. http://es.wikipedia.org/wiki/CASE, 27 de noviembre del 2008.
- Lenguaje Unificado de Modelado. 2008. Wikipedia. http://es.wikipedia.org/wiki/Lenguaje_Unificado_de_Modelado, 28 de noviembre del 2008.
- NetBeans UML. 2007. NetBeans org. http://plugins.netbeans.org/PluginPortal/faces/PluginDetailPage.jsp?pluginid=1801, 07 de febrero de 2009.
- Oracle JDeveloper. 2008. Proiektutaldea-ren Weblog. http://proiektutaldea.wordpress.com/2008/04/16/oracle-jdeveloper/, 07 de febrero de 2009.
- HERNÁNDEZ ORALLO, Enrique. 2002. El Lenguaje Unificado de Modelado (UML). http://www.disca.upv.es/enheror/pdf/ActaUML.PDF, 28 de noviembre del 2008.
- Empresa Arquitecto para UML 2.1 (Enterprise Architect for UML 2.1) v6.5. 2006. Free Download Manager. http://www.freedownloadmanager.org/es/downloads/Arquitecto_de_Empresa_para_UML_2.1_3280_p, 04 de febrero de 2009.
- ALARCÓN, José. 2004. Disponible Eclipse UML 2.0. PC World Digital. http://www.idg.es/pcworld/Disponible-Eclipse-UML-2.0/art163162.htm, 04 de febrero de 2009.
- REYNOSO, Carlos. 2006. Métodos Heterodoxos en Desarrollo de Software. Universidad de Buenos Aires, Argentina, MSDN Library.

http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/heterodox.mspx, 30 de octubre del 2008.

- BARRIENTOS ENRÍQUEZ, Aleida Mirian. 2003. Bolivia. El desarrollo de sistemas de información empleando el lenguaje de modelado unificado UML. Monografías.com. http://www.monografias.com/trabajos16/lenguaje-modelado-unificado/lenguaje-modelado-unificado.shtml, 28 de noviembre del 2008.
- Una lista de herramientas CASE. 2006. Navegapolis.net, anual 1 (1). http://www.navegapolis.net/content/view/406/, 27 de noviembre del 2008.
- ArgoUML 0.24. 2008. Osalt.com. http://www.osalt.com/es/argouml, 04 de febrero de 2009.
- StarUML 5.0. 2008. Taringa. http://www.taringa.net/posts/downloads/1677819/StarUML-5_0-Modelador-Diagramas-UML-Excelente!.html, 04 de febrero de 2009.
- Unidad 5 Teoría de muestreo. 1999. Universidad Autónoma de Querétaro México. http://www.uaq.mx/matematicas/estadisticas/xu5.html, 23 de julio de 2010.
- NAVARRO, Juan. 2005. UMLet 1.5. http://www.versioncero.com/noticia/19/umlet-15, 07 de febrero de 2009.

13. ANEXOS

ANEXO 1

Tamaño de muestras[30]

Para calcular el tamaño de una muestra hay que tomar en cuenta tres factores:

- El porcentaje de confianza con el cual se quiere generalizar los datos desde la muestra hacia la población total.
- El porcentaje de error que se pretende aceptar al momento de hacer la generalización.
- El nivel de variabilidad que se calcula para comprobar la hipótesis.

El **porcentaje de confianza** es el porcentaje de seguridad que existe para generalizar los resultados obtenidos. Esto quiere decir que un porcentaje del 100% equivale a decir que no existe ninguna duda para generalizar tales resultados, pero también implica estudiar a la totalidad de los casos de la población. Comúnmente en las investigaciones se busca un 90 a 95%.

El **porcentaje de error** equivale a elegir una probabilidad de aceptar una hipótesis que sea falsa como si fuera verdadera, o la inversa. Al igual que en el caso de la confianza, si se quiere eliminar el riesgo del error y considerarlo como 0%, entonces la muestra es del mismo tamaño que la población, por lo que conviene correr un cierto riesgo de equivocarse. Comúnmente se aceptan entre el 4% y el 10% como error.

La **variabilidad** es la probabilidad con el que se aceptó y se rechazó la hipótesis que se quiere investigar en alguna investigación anterior o en un ensayo previo a la investigación actual. El porcentaje con que se aceptó tal hipótesis se denomina **variabilidad positiva** y se denota por *p*, y el porcentaje con el que se rechazó se la hipótesis es la **variabilidad negativa**, denotada por *q*.

[30] LARIOS, Víctor. Unidad 5 Teoría de muestreo. http://www.uaq.mx/matematicas/estadisticas/xu5.html, 23 de julio de 2010

Hay que considerar que *p* y *q* son complementarios, es decir, que su suma es igual a la unidad: $p + q = 1$. Además, cuando se habla de la máxima variabilidad, en el caso de no existir antecedentes sobre la investigación, entonces el valor de variabilidad es $p = q = 0.5$.

En el caso de que **no se conozca con precisión el tamaño de la población** se utiliza la siguiente fórmula:

$$n = \frac{Z^2 pq}{E^2}$$

De lo contrario, la fórmula a usar es:

$$n = \frac{Z^2 pqN}{NE^2 + Z^2 pq}$$

En cualquiera de los dos casos:

- n es el tamaño de la muestra
- Z es el nivel de confianza
- p es la variabilidad positiva
- q es la variabilidad negativa
- N es el tamaño de la población
- E es el margen de error

El nivel de confianza se puede establecer de acuerdo a la siguiente tabla:

Tabla 1: Niveles de confianza (95% al 90%)

Confianza	95%	94%	93%	92%	91%	90%
Z	1.96	1.88	1.81	1.75	1.69	1.65
Z^2	3.84	3.53	3.28	3.06	2.86	2.72

ANEXO 2

Encuesta 01 dirigida a desarrolladores de software de nuestra localidad para determinar características, ventajas y desventajas de las herramientas UML más usadas, así como también, los requerimientos de AleDan UML.

UNIVERSIDAD NACIONAL DE LOJA

Área de la Energía, las Industrias y los Recursos Naturales no Renovables

Carrera de Ingeniería en Sistemas

1. **¿Cuáles son los lenguajes de programación con los que trabaja?**

   ……

2. **¿Qué metodología utiliza para el desarrollo de software?**

   ……

   ¿Por qué? ……

3. **¿Cree que es necesario realizar el análisis y diseño de una aplicación antes de construirla?**

   SI ( ) NO ( )

   ¿Por qué? ……

4. **¿Documenta el software bajo las especificaciones UML 2.0?**

   SI ( ) NO ( )

   ¿Por qué? ……

5. **Indique los diagramas UML que normalmente utiliza para el desarrollo de software**

   a) ( ) Diagrama de Clases
   b) ( ) Diagrama de Objetos
   c) ( ) Diagrama de Paquetes
   d) ( ) Diagrama de Despliegue
   e) ( ) Diagrama de Actividad
   h) ( ) Diagrama de Casos de Uso
   i) ( ) Diagrama de Estados
   j) ( ) Diagrama de Secuencia
   k) ( ) Diagrama de Comunicación
   l) ( ) Diagrama de Tiempos

f) ( ) Diagrama de Estructura Compuesta

g) ( ) Diagrama de Componentes

m) ( ) Diagrama de Vista de Interacción

Otros: …..……...……

6. **¿Hace uso de dichos diagramas a la hora de codificar su aplicación?**

   SI ( ) NO ( )

   ¿Por qué? ……..……

7. **¿Cómo selecciona la herramienta UML más adecuada para un determinado proyecto de software? Indique las ventajas y desventajas**

   ……..……

   Ventajas:…….……

   Desventajas:…….……

8. **¿Qué herramienta UML es la que más utiliza para el modelado de sistemas? Mencione algunas de sus características.**

   Nombre: …..……

   a) ……

   b) ……

9. **Dicha herramienta ¿permite la generación de código fuente para el lenguaje de programación que usted utiliza?**

   SI ( ) NO ( )

10. **Mencione algunas características que a su criterio, deberían ser incorporadas en una herramienta UML para ser más eficiente.**

    a) ……

    b) ……

11. **¿Considera usted que sería necesario el desarrollo de una herramienta para análisis y diseño de sistemas que permita la documentación de software basándose en UML 2.0 y la generación de código fuente para C#?**

    SI ( ) NO ( )

    ¿Por qué? ……..……

GRACIAS POR SU COLABORACIÓN!!!

ANEXO 3

Encuesta 02 dirigida a desarrolladores de software de nuestra localidad para validar la Herramienta AleDan UML.

UNIVERSIDAD NACIONAL DE LOJA

Área de la Energía, las Industrias y los Recursos Naturales no Renovables

Carrera de Ingeniería en Sistemas

Como desarrolladores de la Herramienta AleDan UML, le solicitamos comedidamente se digne contestar la presente encuesta, misma que nos permitirá obtener información respecto al funcionamiento de la aplicación.

Datos Informativos	
Tipo de usuario:	*Desarrollador ( ) Docente ( ) Estudiante ( )*
Nombre:	
Ocupación:	
Organización:	
Características del equipo:	
Sistema operativo:	
Herramienta UML que más usa:	

1. ¿Qué tan amigable es la interfaz gráfica de usuario en cuanto a la distribución y organización de componentes?

Excelente ( ) Bueno ( ) Regular ( ) Malo ( )

¿Por qué?.....

2. ¿Qué calificativo asigna a la apertura, creación, eliminación, guardado y cierre de proyectos?

Excelente ( ) Bueno ( ) Regular ( ) Malo ( )

¿Por qué?.....

3. ¿Cómo califica la organización y distribución de un proyecto en el árbol "Navegador de Proyectos"?

Excelente ( ) Bueno ( ) Regular ( ) Malo ( )

¿Por qué?.....

4. ¿Cuál es su apreciación sobre la apertura, creación, eliminación y cierre de diagramas UML?

Excelente ( ) Bueno ( ) Regular ( ) Malo ( )

¿Por qué?.....

5. ¿Cómo califica la administración de elementos gráficos dentro de un diagrama en edición?

Excelente ( ) Bueno ( ) Regular ( ) Malo ( )

¿Por qué?.....

6. La edición y presentación de las propiedades gráficas y estructurales de un artefacto UML dentro de un diagrama en edición es:

Excelente ( ) Bueno ( ) Regular ( ) Malo ( )

¿Por qué?.....

7. ¿Cómo considera el control de entradas erróneas al momento de ingresar datos?

Excelente ( ) Bueno ( ) Regular ( ) Malo ( )

¿Por qué?.....

8. ¿Cree usted que la presentación de mensajes informativos, de control, de error, etc. es?

Excelente ( ) Bueno ( ) Regular ( ) Malo ( )

¿Por qué?.....

9. ¿Cuál es su valoración para el proceso de exportación de diagramas en formato *.png?

Excelente ( ) Bueno ( ) Regular ( ) Malo ( )

¿Por qué?.....

10. El proceso de impresión de diagramas es:

Excelente ( ) Bueno ( ) Regular ( ) Malo ( )

¿Por qué?.....

11. ¿Qué calificativo daría usted al módulo de ayuda incluido dentro de la herramienta?

Excelente ( ) Bueno ( ) Regular ( ) Malo ( )

¿Por qué?.....

12. ¿Cómo considera el grado de coherencia entre el código fuente generado y los diagramas diseñados?

Excelente ( ) Bueno ( ) Regular ( ) Malo ( )

¿Por qué?.....

13. El tiempo de respuesta de la herramienta en la realización de tareas (dibujado de artefactos y diagramas, exportación de diagramas, generación de código e impresión) es:

Excelente ( ) Bueno ( ) Regular ( ) Malo ( )

¿Por qué?.....

14. El desempeño de la herramienta en cuanto al uso de recursos computacionales (memoria RAM, espacio en disco y procesador) es:

Excelente ( ) Bueno ( ) Regular ( ) Malo ( )

¿Por qué?.....

15. En comparación con la herramienta UML más utilizada por usted, AleDan UML es:

Excelente ( ) Bueno ( ) Regular ( ) Malo ( )

¿Por qué?.....

OBSERVACIONES:

.....

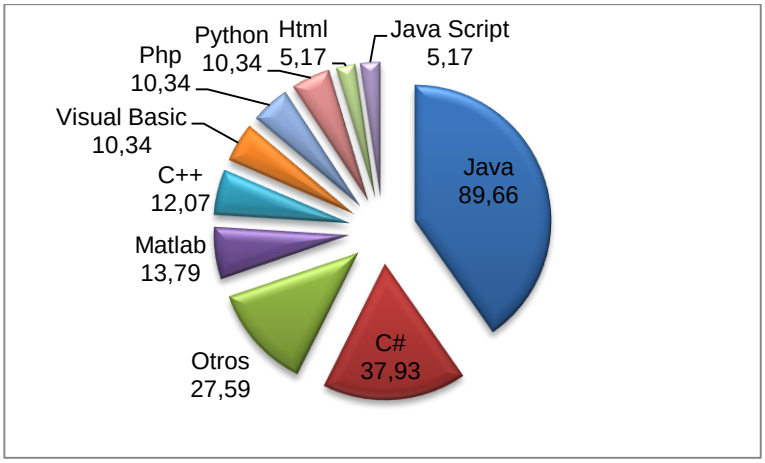
.....

ANEXO 4

Tabulación de Encuesta dirigida a desarrolladores de software de nuestra localidad para determinar características, ventajas y desventajas de las herramientas UML más usadas, así como también, los requerimientos de AleDan UML.

1. ¿Cuáles son los lenguajes de programación con los que trabaja?

Indicadores	Frecuencia	Porcentaje
Java	52	89,66
C#	22	37,93
Otros	16	27,59
Matlab	8	13,79
C++	7	12,07
Visual Basic	6	10,34
Php	6	10,34
Python	6	10,34
Html	3	5,17
Java Script	3	5,17

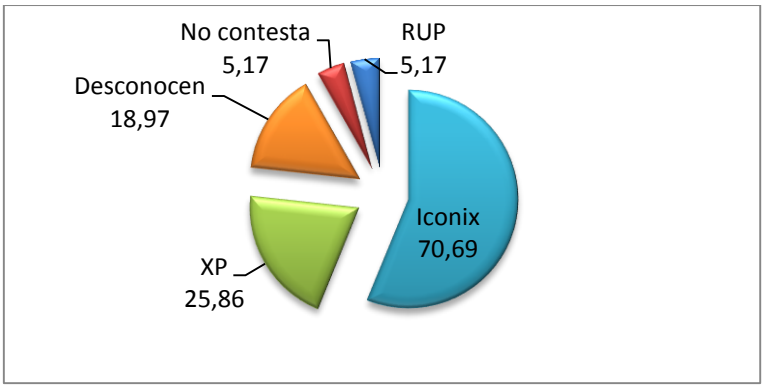


Como puede observarse en el cuadro estadístico y en el gráfico correspondiente a la pregunta 1, los dos lenguajes de programación más utilizados por los desarrolladores de software en nuestro medio son Java y C#, con el 89,66% y 37,93% respectivamente. El tercer lugar con un 27,59% es para otros lenguajes como Spring, Seam, etc., mismos que por su baja utilización no fueron mencionados individualmente. La cuarta y quinta posición la ocupan Matlab con un 13,79% y C++

con el 12,07%. A continuación con un 10,34% se encuentran igualados los lenguajes Visual Basic, Php y Python, para finalmente ubicar a Html y Java Script con 5,17% cada uno.

2. ¿Qué metodología utiliza para el desarrollo de software?

Indicadores	Frecuencia	Porcentaje
Iconix	41	70,69
XP	15	25,86
Desconocen	11	18,97
RUP	3	5,17
No contesta	3	5,17



El 70,69% de los encuestados prefieren Iconix como metodología para el desarrollo de software, el 25,86% se inclinan por XP y el 5,17% por RUP. Un 18,97% desconocen el tema y otro 5,17% no contesta.

Quienes optan por **Iconix** lo hacen fundamentándose en que es una metodología de desarrollo ordenada, iterativa e incremental que permite actualizar constantemente la aplicación a construir. Además se añade que dicha metodología se basa en el manejo de diagramas, es más fácil de entender, permite alcanzar mayor dominio del problema, es flexible, eficiente, robusta y se adapta fácilmente al desarrollo de proyectos grandes, medianos y pequeños. Finalmente hacen prevalecer el hecho de que tienen mayor experiencia en el uso de Iconix.

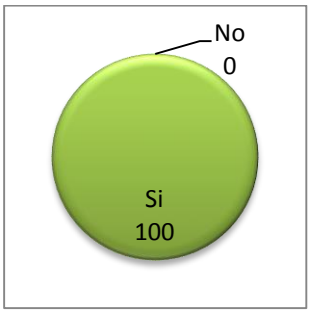
Por su parte la metodología **XP** es preferida ya que en ocasiones no se dispone del tiempo suficiente para aplicar otras metodologías que así lo demandan: XP supone un análisis corto y no necesita mucha documentación (diagramas).

Los desarrolladores que prefieren **RUP** afirman que aporta en gran medida al desarrollo rápido, especialmente en proyectos académicos, mismos que deben ser realizados en lapsos comprendidos entre tres y cuatro meses.

Desconocen incluye a los desarrolladores que señalaron indicadores que no corresponden a metodologías de desarrollo: modelo vista – controlador, estructurada, incremental, lineal – secuencial, ciclo de desarrollo clásico de sistemas y cascada.

3. ¿Cree que es necesario realizar el análisis y diseño de una aplicación antes de construirla?

Indicadores	Frecuencia	Porcentaje
Si	58	100
No	00	000
TOTAL	**58**	**100**

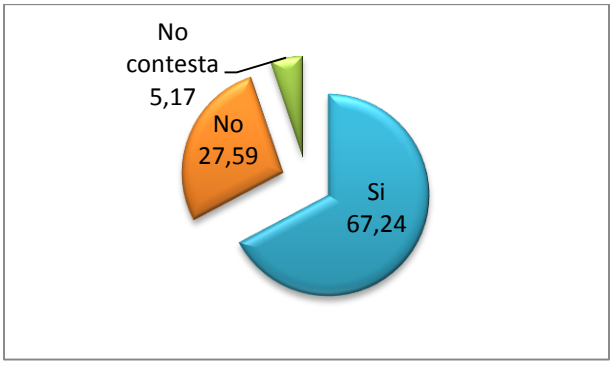


El 100% de los encuestados afirman que es completamente necesario realizar el análisis y diseño de una aplicación antes de construirla, puesto que permite involucrar al desarrollador con el usuario para determinar la magnitud y complejidad del proyecto, obtener sus requerimientos y establecer si su construcción es factible. De éste modo se puede conocer la realidad del problema para planificar paso a paso su solución y asegurar que el producto final sea un éxito. Añaden también que este estudio es la base para saber qué hacer, cómo hacer, cuándo hacer y qué se necesita para lograrlo, evitando cualquier riesgo o problema durante la construcción del software en estudio.

4. ¿Documenta el software bajo las especificaciones UML 2.0?

Indicadores	Frecuencia	Porcentaje
Si	39	67,24

No	16	27,59
No contesta	3	5,17
TOTAL	**58**	**100**



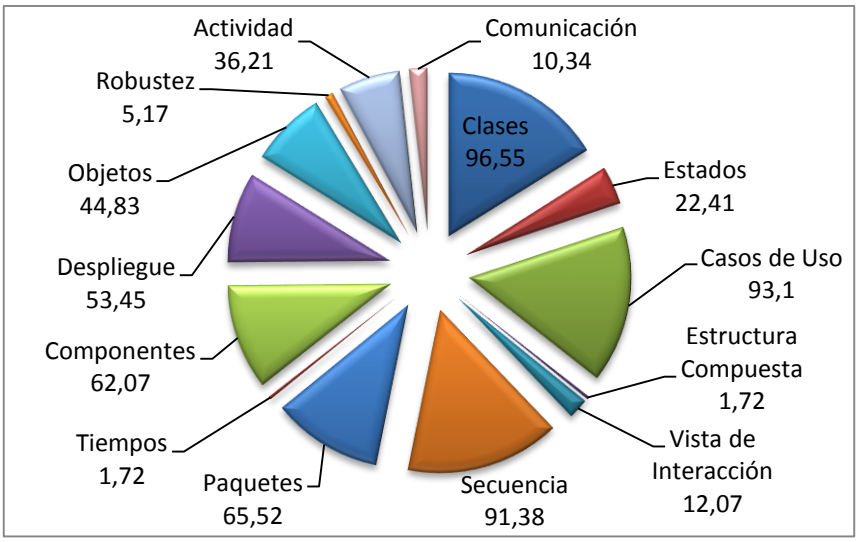
Quienes responden afirmativamente, un 67,24% de los desarrolladores encuestados, aseguran que los diagramas dados en ésta versión de UML son de gran ayuda para el desarrollador, pues permiten documentar y entender la arquitectura de un sistema, de tal modo que los programadores que no intervinieron en la construcción de un determinado producto, puedan empaparse fácilmente de su funcionamiento, haciendo más sencillo, eficaz y eficiente su manejo y mantenimiento. Es más, al representar gráficamente la arquitectura de una aplicación se depura en un gran porcentaje el sistema a desarrollar.

Por su parte el 27,59% de los encuestados que responden negativamente a ésta pregunta, señalan que siempre hay mejores especificaciones para documentar software, o en su defecto, que no tienen mayor conocimiento acerca de ésta especificación. Finalmente el 5,17% no contesta a ésta interrogante.

5. Indique los diagramas UML que normalmente utiliza para el desarrollo de software

Indicadores	Frecuencia	Porcentaje
Diagrama de Clases	56	96,55
Diagrama de Casos de Uso	54	93,10
Diagrama de Secuencia	53	91,38
Diagrama de Paquetes	38	65,52
Diagrama de Componentes	36	62,07
Diagrama de Despliegue	31	53,45
Diagrama de Objetos	26	44,83

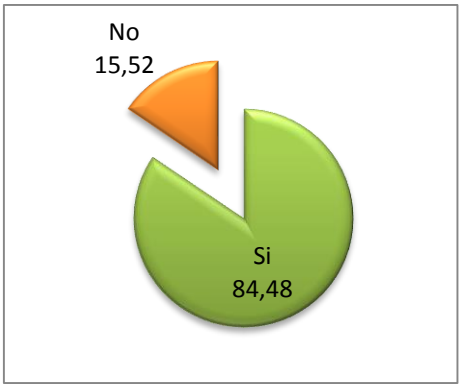
Diagrama de Actividad	21	36,21
Diagrama de Estados	13	22,41
Diagrama de Vista de Interacción	7	12,07
Diagrama de Comunicación	6	10,34
Diagrama de Robustez	3	5,17
Diagrama de Estructura Compuesta	1	1,72
Diagrama de Tiempos	1	1,72



Los desarrolladores utilizan con mayor frecuencia los Diagramas de Clases, Casos de Uso y Secuencia con un 96,55%, 93,1% y 91,38% respectivamente. El cuarto lugar lo ocupa el Diagrama de Paquetes con el 65,52% seguido por el Diagrama de Componentes con un 62,07%. Luego se ubica al Diagrama de Despliegue con el 53,45%, al de Objetos con un 44,83% y al Diagrama de Actividad con el 36,21%. Los resultados de las encuestas indican que un 22,41% de los desarrolladores usan el Diagrama de Estados, un 12,07% el Diagrama de Vista de Interacción y el 10,34% el Diagrama de Comunicación. El 1,72% utilizan tanto el Diagrama de Estructura Compuesta como el Diagrama de Tiempos. Finalmente, dentro de otros diagramas que son elaborados para documentar el software, el 5,17% de los encuestados coincidieron en que utilizan el Diagrama de Robustez.

6. ¿Hace uso de dichos diagramas a la hora de codificar su aplicación?

Indicadores	Frecuencia	Porcentaje
Si	49	84,48
No	9	15,52
TOTAL	**58**	**100**



Un total de 49 encuestados que corresponden al 84,48% manifiestan que al comenzar con la fase de codificación, hacen uso de los diagramas UML elaborados previamente en la fase de diseño. Por su parte, el 15,52% no lo hace.

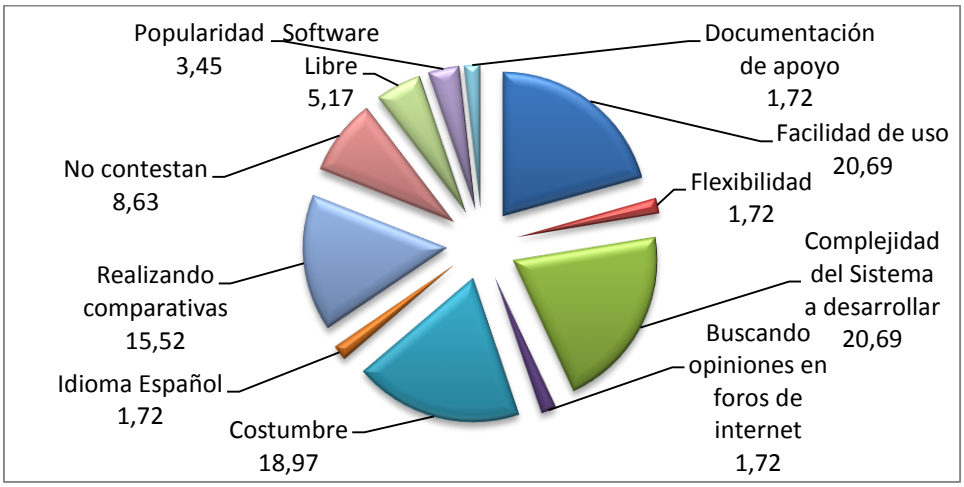
Los primeros alegan que con ello se facilita la codificación del software puesto que dichos diagramas contienen toda la estructura o esqueleto de la aplicación. De ésta manera dichos diagramas se constituyen en una verdadera guía para no salirse del ámbito del problema, definiendo de forma sencilla y estructurada las entidades, relaciones, actividades y atributos de un sistema, que permitirán construir un software rápido, de buena calidad y acorde tanto a las necesidades del cliente como a las especificaciones del análisis y diseño realizados previamente.

Por otro lado, quienes no se apoyan en los diagramas realizados en la etapa de diseño indican que no lo hacen porque no es su costumbre, porque no quieren perder tiempo de desarrollo, o bien, porque prefieren primero codificar la aplicación para que en base al código fuente, la herramienta genere automáticamente los diagramas.

7. ¿Cómo selecciona la herramienta UML más adecuada para un determinado proyecto de software? Indique las ventajas y desventajas

Indicadores	Frecuencia	Porcentaje
Facilidad de uso	12	20,69
Complejidad del sistema a desarrollar	12	20,69
Costumbre	11	18,97
Realizando comparativas	9	15,52
Software Libre	3	5,17
Popularidad	2	3,45
Flexibilidad	1	1,72

Documentación de apoyo	1	1,72
Idioma Español	1	1,72
Buscando opiniones en foros de internet	1	1,72
No contestan	5	8,63
TOTAL	**58**	**100**



La facilidad de uso así como la complejidad que tenga el sistema que se va a construir, son los criterios más tomados en cuenta por los desarrolladores para seleccionar una herramienta UML; así lo demuestra el 20,69% que tiene cada uno de éstos criterios. Otro 18,97% de los encuestados ocupa una determinada herramienta para cualquier proyecto que realicen puesto que ya tienen experiencia en su manejo. Un 15,52% señalan que realizan comparativas entre herramientas de tal manera que se pueda escoger aquella que más se apegue a sus necesidades. En quinto lugar, el 5,17% de encuestados afirman que para trabajar con UML optan por aquellas de software libre. Existe un 3,45% de personas que se basan en la popularidad de dichas herramientas para utilizarlas en sus proyectos. La flexibilidad, la documentación de apoyo, el idioma español y la búsqueda de opiniones en foros de internet son los últimos cuatro criterios de selección, con el 1,72% para cada uno de ellos. Finalmente el 8,63% de los encuestados no contestan a ésta interrogante.

De acuerdo a los desarrolladores, reducir el tiempo de selección, mejor comprensión de la herramienta y una interfaz gráfica amigable son las ventajas del criterio **Facilidad de Uso**. Como desventajas se menciona que la mayoría de estas herramientas son muy pesadas para el ordenador, se encuentran en inglés, tienen licencia comercial, no poseen todas las librerías o complementos necesarios y siempre existen herramientas con más y mejores ventajas que al no ser fáciles de utilizar son descartadas.

Las ventajas del indicador **Complejidad del sistema a desarrollar** son que facilita el modelado, mejora la definición de la relación entre componentes del sistema y ahorra tiempo. No obstante se debe tener en consideración los elevados precios de licencia, la complejidad de manejo y que no se encuentran versiones disponibles en español.

Por su parte la **Costumbre** al utilizar una determinada herramienta facilita la documentación, permite adquirir cada vez mayor experiencia en su uso y finalmente, agilita el desarrollo de proyectos. En contraparte, la mayoría de dichas herramientas son software privativo, tienen poca documentación que ayude en su uso y no se conoce otras herramientas con mejores prestaciones.

El aspecto positivo del criterio **Realizando comparativas** es que siempre se selecciona la herramienta más rápida, conveniente, confiable, sencilla, eficaz, eficiente y la de mayor rendimiento. Sin embargo generalmente aquellas herramientas que reúnen éstas características son de licencia comercial, su uso es complicado y por lo tanto consumen mucho tiempo para aprender a manejarlas.

Según los encuestados, que las herramientas UML sean **Software Libre** trae la ventaja de no tener que pagar por licencias, pero no existe apoyo técnico para los problemas de funcionamiento que se presenten. Así mismo se sostiene el hecho de que las mejores herramientas tienen licencia propietaria.

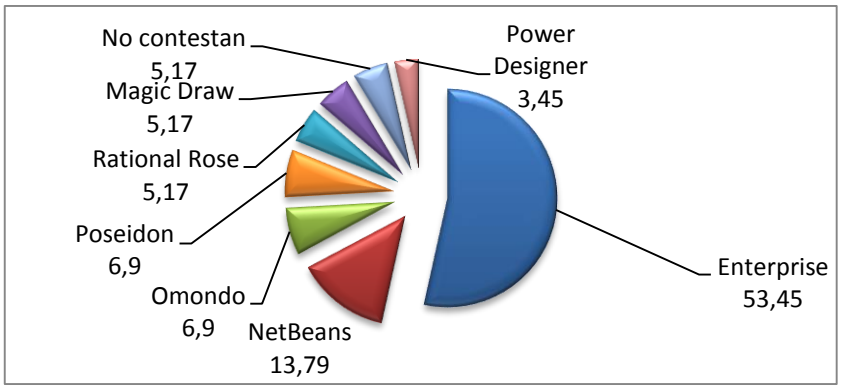
Los criterios **Popularidad**, **Flexibilidad** e **Idioma Español** no tienen comentario alguno en ventajas o desventajas. Por su parte, la **Documentación de apoyo** permite a sus usuarios obtener un mayor entendimiento de la herramienta UML.

Finalmente el criterio **Buscando opiniones en foros de internet** garantiza facilidad de uso de la herramienta seleccionada y versatilidad de la misma. A pesar de ello, se debe considerar el alto costo de estas herramientas así como también su soporte.

8. ¿Qué herramienta UML es la que más utiliza para el modelado de sistemas? Mencione algunas de sus características

Indicadores	Frecuencia	Porcentaje
Enterprise	31	53,45
NetBeans	8	13,79

Omondo	4	6,9
Poseidon	4	6,9
Rational Rose	3	5,17
Magic Draw	3	5,17
Power Designer	2	3,45
No contestan	3	5,17
TOTAL	**58**	**100**



Como se puede observar en el cuadro y en la representación gráfica correspondiente a ésta pregunta, la herramienta Enterprise Architect es la más utilizada por los desarrolladores con el 53,45% de aceptación. El 13,79% se inclinan por el workbench que incluye el IDE NetBeans, mientras que el 6,9% optan por Omondo, plugin del IDE Eclipse. Otro 6,9% eligen la herramienta Poseidon para elaborar los diagramas UML que sean necesarios. Rational Rose y Magic Draw son utilizadas por el 5,17% cada una, mientras que Power Designer es elegida por el 3,45%. Por último el 5,17% de los desarrolladores no contestan a ésta interrogante.

Enterprise presenta facilidad de uso, generación de código en diferentes lenguajes de programación, organización completa del proyecto, variedad de utilerías en barras bien organizadas, soporte a Iconix, multiplataforma, permite la reingeniería, potente, buena interfaz, licencia comercial, no es pesado y su eficiencia.

En cuanto a las características que presenta **NetBeans**, se señalan su facilidad de uso, generación de código fuente, muy potente, permite la ingeniería inversa, permite realizar la mayoría de diagramas, buen GUI y que brinda todas las herramientas necesarias para editar los diagramas UML.

Por su parte quienes trabajan con **Omondo** destacan que existen versiones libres de ésta herramienta, su generación de código fuente, gran cantidad de paquetes para la

elaboración de diagramas, su facilidad de uso, exportación en muchos formatos y que permite realizar la gran mayoría de los diagramas que se necesitan.

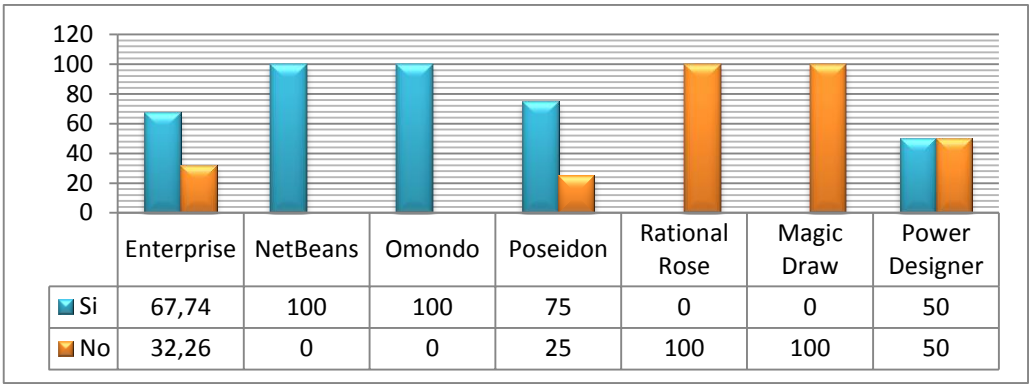
Quienes se deciden por **Poseidon** afirman que es fácil de usar, posee buena interfaz gráfica, más completo en cuanto a herencias, permiten editar fácilmente key's, generación de código fuente, existen versiones en español, y de igual manera, ciertas versiones son libres.

Rational Rose permite elaborar de manera mucho más sencilla los diagramas UML, es fácil de utilizar y permite hacer la mayoría de diagramas. De igual manera la herramienta **Magic Draw** es libre, se encuentra disponible en español, tiene buena interfaz gráfica y facilidad de uso.

Los encuestados que prefieren **Power Designer** se fundamentan en que genera código fuente, es fácil de manejar, apoya en el diseño y permite tanto la ingeniería inversa como la conectividad con bases de datos. Sin embargo, es importante recalcar que ésta no es una herramienta UML, puesto que está orientada al desarrollo de bases de datos.

9. Dicha herramienta ¿permite la generación de código fuente para el lenguaje de programación que usted utiliza?

Herramienta UML	Indicadores	Frecuencia	Porcentaje
NetBeans	**Si**	8	100
	No	0	00
Rational Rose	**Si**	0	000
	No	3	100
Omondo	**Si**	4	100
	No	0	000
Enterprise	**Si**	21	67,74
	No	10	32,26
Power Designer	**Si**	1	50
	No	1	50
Magic Draw	**Si**	0	000
	No	3	100
Poseidon	**Si**	3	75
	No	1	25



	Enterprise	NetBeans	Omondo	Poseidon	Rational Rose	Magic Draw	Power Designer
Si	67,74	100	100	75	0	0	50
No	32,26	0	0	25	100	100	50

El 67,74% de quienes utilizan **Enterprise** afirman que sí genera código fuente, mientras que el restante 32,26% responde que no tiene ésta función. El 100% de los desarrolladores que utilizan **NetBeans** y **Omondo** aseveran que efectivamente estas herramientas generan código fuente para el lenguaje de programación que utilizan. Asimismo el 75% de los encuestados que usan **Poseidon** responden afirmativamente a ésta pregunta, mientras que el 25% lo hacen negativamente. Para la totalidad de quienes hacen uso de las herramientas **Rational Rose** y **Magic Draw**, éstas no generan código fuente. Los usuarios de **Power Designer** que fueron encuestados se encuentran divididos en partes iguales, puesto que el 50% contesta que si genera código, mientras que el otro 50% manifiestan que no lo hace.

10. Mencione algunas características que a su criterio, deberían ser incorporadas en una herramienta UML para ser más eficiente

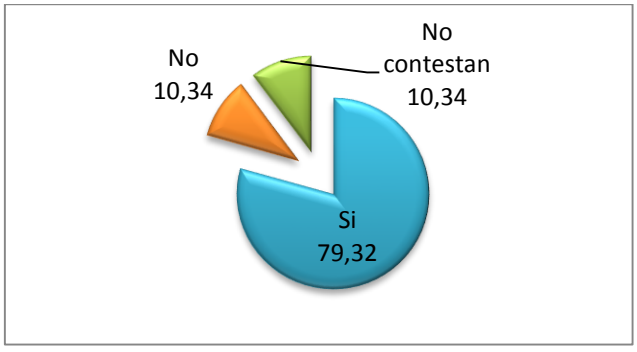
De acuerdo a quienes fueron encuestados las características a tomar en consideración para el desarrollo de una herramienta UML más eficiente y apegada a sus condiciones de trabajo son las siguientes:

- Que sea desarrollado como software libre
- Que esté disponible en idioma español
- Mejorar y ordenar la interfaz gráfica de tal manera que sea más intuitiva para el usuario
- Que sea portable, flexible, rápido, multiplataforma, versátil, potente y fácil de usar
- Que sea orientado a objetos
- Soporte Iconix
- Generación de código fuente para por lo menos, los lenguajes de programación más utilizados en nuestro medio

- Detallar dicha generación de código
- Que permita la elaboración de todos los diagramas UML y que disponga de herramientas completas para la elaboración de cada uno de ellos
- Que permita ingresar casos de uso en forma de texto
- Que permita generar diagramas a partir de los ya creados
- Que detecte errores en el momento de modelar
- Que permita la edición de sus componentes
- Que incluya tutoriales dinámicos acerca de la herramienta
- Que permita el empaquetamiento de clases
- Soporte para importar y exportar en XMI
- Presentación de plantillas en sistemas estándar
- Plantillas patrones de procesos personalizables
- Que permita la ingeniería inversa
- Que permita realizar estimaciones

11. ¿Considera usted que sería necesario el desarrollo de una herramienta para análisis y diseño de sistemas que permita la documentación de software basándose en UML 2.0 y la generación de código fuente para C#?

Indicadores	Frecuencia	Porcentaje
Si	46	79,32
No	6	10,34
No contestan	6	10,34
TOTAL	**58**	**100**



El 79,32% de los encuestados que corresponden a 46 desarrolladores, indican que sí sería conveniente el desarrollo de una herramienta que reúna éstas características, mientras que un 10,34% no lo cree así. Otro 10,34% no contesta a ésta interrogante.

Quienes contestaron afirmativamente manifiestan que de ésta forma se facilitaría el desarrollo de software, ahorrando tiempo tanto a analistas como a desarrolladores. Así mismo creen que con la generación de código fuente se incrementaría la posibilidad de que los programadores se apoyen en los diagramas realizados en la fase de diseño a la hora de realizar la codificación del software. También se destaca que existen muy pocas herramientas UML que tomen en cuenta a C#, e independientemente del lenguaje al que esté dirigido, la mayoría tienen licencia comercial y son complejas de utilizar. Por último opinan que sería conveniente que la generación de código fuente no sea únicamente para un lenguaje de programación, sino también para los más utilizados por los desarrolladores de software.

Quienes optaron por el indicador **No**, se basan principalmente en que en la actualidad existe una gran variedad de herramientas que brindan éste servicio y en que no conocen el lenguaje de programación C#, razón por la cual recomiendan que la generación de código fuente debería ser para Java, lenguaje de programación más utilizado en nuestro medio.

ANEXO 5

Análisis de las herramientas basadas en UML 2.0 más utilizadas por los desarrolladores de software

1. Listado

Los resultados obtenidos a partir de la encuesta realizada a desarrolladores de software de nuestra localidad, demostraron que las herramientas UML más utilizadas son las siguientes:

- Enterprise
- NetBeans
- Omondo
- Poseidon
- Rational Rose
- Magic Draw

Existe también un pequeño porcentaje de encuestados que sugirió Power Designer como su herramienta predilecta, pero al estar orientada al desarrollo de bases de datos, no se tomará en cuenta durante el presente análisis.

2. Prestaciones

Enterprise 7.0

- Software comercial.
- Facilidad de uso.
- Interfaz gráfica de usuario amigable.
- Basado en la especificación UML 2.0.
- Soporte para el desarrollo de Diagramas de Paquetes, Clases, Objetos, Estructura Compuesta, Componentes, Despliegue, Casos de Uso, Actividad, Estado, Comunicación, Secuencia, Tiempos y Vista de Interacción.
- Paleta con los elementos y utilerías UML que se carga de acuerdo al tipo de diagrama seleccionado.

- Generación de código fuente para los lenguajes de programación ActionScript, C, C#, C++, Delphi, Java, PHP, Python, Visual Basic Net, y Visual Basic.
- Generación de Documentación a partir de los diagramas UML, en formato RTF y HTML.
- Ingeniería inversa.
- Permite al usuario organizar su proyecto mediante la creación de vistas.
- Presenta asistentes para la configuración de nodos y relaciones al momento de crearlos o editarlos.
- Las líneas de relación pueden ser modificadas en varios puntos, tantos como requiera el usuario.
- Distribución manual o preestablecida de los elementos utilizados en un diagrama.
- Alineación de los componentes del diagrama.
- Permite modificar las propiedades gráficas de los elementos UML.
- Propiedades para Zoom In y Zoom Out.
- Propiedades para Hacer y Deshacer.
- Impresión de diagramas.
- Exportación de diagramas en los formatos *.bmp, *.png, *.jpg, *.tga, *.gif, *.wmf y *.emf.
- Importación y Exportación XMI.
- Ayuda para el manejo de la herramienta (Inglés).

NetBeans 6.1

- Software libre.
- Facilidad de uso.
- Interfaz gráfica de usuario amigable.
- Basado en la especificación UML 2.0.
- Multiplataforma.
- Soporte para el desarrollo de Diagramas de Actividad, Clases, Colaboración, Componentes, Despliegue, Secuencia, Estado y Casos de Uso.
- Paleta con los elementos y utilerías UML que se carga de acuerdo al tipo de diagrama seleccionado.
- Generación de código fuente para el lenguaje de programación Java.
- Generación de Documentación a partir de los diagramas UML, en formato HTML.
- Ingeniería inversa.
- Organización automática de cada diagrama de acuerdo al tipo al que pertenezca.

- Presenta las propiedades para configuración de los elementos UML al seleccionarlos.
- Las líneas de relación pueden ser modificadas en varios puntos, tantos como requiera el usuario.
- Distribución manual o preestablecida de los elementos utilizados en un diagrama.
- Permite modificar las propiedades gráficas de los elementos UML.
- Propiedades para Zoom In y Zoom Out.
- Propiedades para Hacer y Deshacer.
- Impresión de diagramas.
- Exportación de diagramas en los formatos *.jpg, *.svg, y *.png.
- Ayuda para el manejo de la herramienta (Inglés).

Omondo

- Software libre y comercial.
- Facilidad de uso.
- Interfaz gráfica de usuario amigable.
- Basado en la especificación UML 2.0.
- Multiplataforma.
- Soporte para el desarrollo de Diagramas de Clases, Objetos, Componentes, Despliegue, Casos de Uso, Actividad, Estado, Comunicación, Secuencia, y Robustez.
- Sencilla paleta con los elementos y utilerías UML que se carga de acuerdo al tipo de diagrama seleccionado.
- Generación de código fuente para el lenguaje de programación Java.
- Ingeniería inversa.
- Organización en el mismo proyecto del código fuente.
- Presenta asistentes para la configuración de nodos y relaciones al momento de crearlos o editarlos.
- Las líneas de relación pueden ser modificadas en varios puntos, tantos como requiera el usuario.
- Alineación de los componentes del diagrama.
- Permite modificar las propiedades gráficas de los elementos UML.
- Propiedades para Zoom In y Zoom Out.
- Visualización de diagramas a escala.
- Propiedades para Hacer y Deshacer.

- Impresión de diagramas.
- Exportación de diagramas en los formatos *.svg, *.gif, *.wmf y *.jpg.
- Exportación XMI.
- Ayuda para el manejo de la herramienta (Inglés).

Poseidon 6.0.2 – 0

- Software libre y comercial.
- Facilidad de uso.
- Interfaz gráfica de usuario amigable.
- Basado en la especificación UML 2.0.
- Multiplataforma.
- Soporte para el desarrollo de Diagramas de Clases, Casos de Uso, Estado, Actividad, Comunicación, Secuencia, Componentes y Despliegue.
- Sencilla paleta con los elementos y utilerías UML que se carga de acuerdo al tipo de diagrama seleccionado.
- Permite agregar de manera rápida nodos, relaciones, atributos y métodos a partir de un elemento con sólo seleccionarlo.
- Visualización del código fuente de un elemento seleccionado.
- Permite al usuario ingresar la documentación en forma de texto, de un elemento seleccionado.
- Generación de código fuente para el lenguaje de programación Java.
- Generación de Documentación a partir de los diagramas UML, en formato HTML y DOC.
- Ingeniería inversa.
- Organización automática de cada diagrama de acuerdo al tipo al que pertenezca.
- Presenta las propiedades para configuración de los elementos UML al seleccionarlos.
- Las líneas de relación pueden ser modificadas en varios puntos, tantos como requiera el usuario.
- Distribución manual o preestablecida de los elementos utilizados en un diagrama.
- Alineación de los componentes del diagrama.
- Permite modificar las propiedades gráficas de los elementos UML.
- Propiedades para Zoom In y Zoom Out.
- Visualización de diagramas a escala.
- Propiedades para Hacer y Deshacer.

- Impresión de diagramas.
- Exportación de diagramas en todos los formatos gráficos estándares.
- Exportación XMI.
- Ayuda para el manejo de la herramienta (Inglés).

Rational Rose 2003

- Software comercial.
- Facilidad de uso.
- Basado en la especificación UML 2.0.
- Multiplataforma.
- Permite escoger el lenguaje de programación con el que se va a trabajar, antes de iniciar un proyecto.
- Soporte para el desarrollo de Diagramas de Clases, Casos de Uso, Comunicación, Componentes, Despliegue, Actividad, Estado y Secuencia.
- Sencilla paleta con los elementos y utilerías UML que se carga de acuerdo al tipo de diagrama seleccionado.
- Cuenta con un asistente que contiene las clases estándar del lenguaje de programación con el que se esté trabajando.
- Generación de código fuente para los lenguajes de programación ANSI C++, Ada 83, Ada 95, CORBA, Java, Visual C++ y Visual Basic.
- Ingeniería inversa.
- Las líneas de relación pueden ser modificadas en varios puntos, tantos como requiera el usuario.
- Permite modificar las propiedades gráficas de los elementos UML.
- Propiedades para Zoom In y Zoom Out.
- Impresión de diagramas.
- Ayuda para el manejo de la herramienta (Inglés).

Magic Draw 16.5

- Software comercial.
- Facilidad de uso.
- Interfaz gráfica de usuario amigable.
- Basado en la especificación UML 2.0.
- Multiplataforma.

- Soporte para el desarrollo de Diagramas de Clases, Casos de Uso, Comunicación, Secuencia, Estado, Actividad, Estructura Compuesta, Componentes, Despliegue, Tiempos, Robustez y Vista de Interacción. Además incluye otro tipo de diagramas.
- Paleta muy completa con los elementos y utilerías UML del diagrama seleccionado y de los relacionados con éste.
- Permite agregar de manera rápida nodos, relaciones, atributos y métodos a partir de un elemento con sólo seleccionarlo.
- Permite al usuario ingresar la documentación en forma de texto, de un elemento seleccionado.
- Cuenta con un asistente que contiene las clases estándar del lenguaje de programación con el que se esté trabajando.
- Generación de código fuente para los lenguajes de programación C#, C++, Java, CORBA IDL, DDL, EJB 2.0, XML Schema y WSDL.
- Generación de Documentación a partir de los diagramas UML, en formato HTML.
- Ingeniería inversa.
- Presenta las propiedades para configuración de los elementos UML al seleccionarlos.
- Las líneas de relación pueden ser modificadas en varios puntos, tantos como requiera el usuario.
- Distribución manual o preestablecida de los elementos utilizados en un diagrama.
- Alineación de los componentes del diagrama.
- Permite modificar las propiedades gráficas de los elementos UML.
- Propiedades para Zoom In y Zoom Out.
- Visualización de diagramas a escala.
- Propiedades para Hacer y Deshacer.
- Impresión de diagramas.
- Exportación de diagramas en los formatos *.eps, *.png, *.jpg, *.svg, *.tif, *.tiff, *.wmf y *.emf.
- Importación y Exportación XMI.
- Ayuda para el manejo de la herramienta (Inglés).

3. Fortalezas y debilidades

Fortalezas	Debilidades
Enterprise	
▪ Soporta los 13 tipos de diagramas de UML 2.0. ▪ Disponibilidad de elementos y utilerías UML para el desarrollo de diagramas. ▪ Generación de código fuente. ▪ Generación de documentación. ▪ Ingeniería inversa. ▪ Asistentes para la configuración de nodos y relaciones. ▪ Distribución y alineación de los elementos que conforman un diagrama. ▪ Exportación de diagramas en varios formatos gráficos. ▪ Importación y exportación XMI.	▪ Software comercial. ▪ Permite modelar herencia múltiple en lenguajes de programación que no la admiten. ▪ La generación de código fuente es incorrecta cuando se modela herencia múltiple para lenguajes que no la admiten. ▪ Al arrastrar clases a paquetes, dentro del diagrama, el modelo no actualiza las dependencias creadas, razón por la que se presentan errores durante la generación de código fuente. ▪ No se presentan mensajes cuando el usuario comete errores en el diseño de diagramas. ▪ El manual de usuario se encuentra en idioma inglés.
NetBeans	
▪ Software libre. ▪ Soporta 8 tipos de diagramas UML 2.0. ▪ Facilidad de uso. ▪ Multiplataforma. ▪ Interfaz gráfica de usuario amigable. ▪ Disponibilidad de elementos y utilerías UML para el desarrollo de diagramas. ▪ Generación de código fuente. ▪ Generación de documentación. ▪ Ingeniería inversa. ▪ Distribución de los elementos que conforman un diagrama (layout). ▪ Exportación de diagramas en los 3 principales formatos gráficos.	▪ Necesita un equipo con gran capacidad de procesamiento. ▪ La generación de código fuente es únicamente para Java ▪ La generación de documentación es únicamente en formato HTML. ▪ El manual de usuario se encuentra en idioma inglés.
Omondo	
▪ Software libre. ▪ Soporta 10 tipos de diagramas UML 2.0. ▪ Facilidad de uso. ▪ Multiplataforma. ▪ Interfaz gráfica de usuario amigable. ▪ Disponibilidad de elementos y utilerías UML para el desarrollo de diagramas.	▪ Las mejores prestaciones se encuentran en la versión comercial. ▪ La generación de código fuente es únicamente para Java. ▪ El manual de usuario se encuentra en idioma inglés.

▪ Generación de código fuente. ▪ Ingeniería inversa. ▪ Asistentes para la configuración de nodos y relaciones al crear o editar. ▪ Alineación de los elementos que conforman un diagrama. ▪ Exportación de diagramas en 4 formatos gráficos. ▪ Importación y exportación XMI.	
Poseidon	
▪ Software libre. ▪ Soporta 8 tipos de diagramas UML 2.0. ▪ Facilidad de uso. ▪ Multiplataforma. ▪ Interfaz gráfica de usuario amigable. ▪ Disponibilidad de elementos y utilerías UML para el desarrollo de diagramas. ▪ Rápida creación de nodos, relaciones, atributos y métodos a partir de un elemento seleccionado. ▪ Visualización del código fuente de un elemento seleccionado. ▪ Generación de código fuente. ▪ Generación de documentación. ▪ Ingeniería inversa. ▪ Distribución y alineación de los elementos que conforman un diagrama. ▪ Exportación de diagramas en todos los formatos gráficos estándares. ▪ Importación y exportación XMI.	▪ Las mejores prestaciones se encuentran en la versión comercial. ▪ Permite modelar herencia múltiple. ▪ La creación rápida de nodos, relaciones, atributos y métodos a partir de un elemento seleccionado no está totalmente controlada. ▪ El código fuente que se visualiza es inexacto. ▪ La generación de código fuente es únicamente para Java. ▪ La generación de código fuente es incorrecta cuando se modela herencia múltiple, para lenguajes que no la admiten. ▪ El manual de usuario se encuentra en idioma inglés.
Rational Rose	
▪ Permite trabajar con varios lenguajes de programación. ▪ Soporta 8 tipos de diagramas UML 2.0. ▪ Multiplataforma. ▪ Disponibilidad de elementos y utilerías UML para el desarrollo de diagramas. ▪ Asistente que contiene las clases estándar del lenguaje de programación con el que se esté trabajando. ▪ Generación de código fuente. ▪ Ingeniería inversa.	▪ Software comercial. ▪ La creación rápida de nodos, relaciones, atributos y métodos a partir de un elemento seleccionado no está totalmente controlada. ▪ El manual de usuario se encuentra en idioma inglés.
Magic Draw	
▪ Soporta los 13 tipos de diagramas UML 2.0 y más. ▪ Facilidad de uso. ▪ Multiplataforma.	▪ La creación rápida de nodos, relaciones, atributos y métodos a partir de un elemento seleccionado no está totalmente controlada.

▪ Completa disponibilidad de elementos y utilerías UML para el desarrollo de diagramas. ▪ Rápida creación de nodos, relaciones, atributos y métodos a partir de un elemento seleccionado. ▪ Asistente que contiene las clases estándar del lenguaje de programación con el que se esté trabajando. ▪ Generación de código fuente. ▪ Generación de documentación. ▪ Ingeniería inversa. ▪ Distribución y alineación de los elementos que conforman un diagrama. ▪ Exportación de diagramas en varios formatos gráficos. ▪ Importación y Exportación XMI.	▪ Software comercial. ▪ Permite modelar herencia múltiple. ▪ La documentación generada se encuentra únicamente en formato HTML. ▪ El manual de usuario se encuentra en idioma inglés.

ANEXO 6

Tabulación de Encuesta dirigida a desarrolladores de software de nuestra localidad para validar la Herramienta AleDan UML.

1. ¿Qué tan amigable es la interfaz gráfica de usuario en cuanto a la distribución y organización de componentes?

Tabla 1: Resultados a la pregunta Nº 1

Indicadores	Frecuencia	Porcentaje
Excelente	20	57,1 %
Bueno	15	42,9 %
Regular	0	0,0 %
Malo	0	0,0 %
TOTAL	**35**	**100 %**

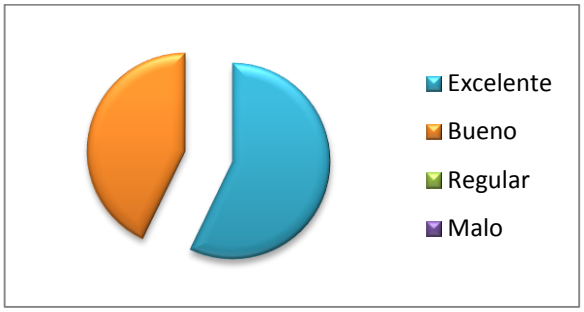


Figura 1: Resultados a la pregunta Nº 1

Como puede observarse en el cuadro estadístico y en el gráfico correspondiente a la pregunta 1, 20 usuarios que corresponden al 57,1 % consideran que la interfaz gráfica de usuario es “Excelente”, mientras que los restantes 15 usuarios equivalentes al 42,9 % la catalogan como “Bueno”.

En sentido general, se destaca que ésta herramienta posee una buena distribución de iconos, que es interactiva e intuitiva y que posee más facilidades que otros programas para la elaboración de diagramas UML. El estar en español, es una característica que le permite ser mucho más comprensible, y su ambiente de trabajo organizado y colorido, sencillamente llaman la atención del usuario.

2. ¿Qué calificativo asigna a la apertura, creación, eliminación, guardado y cierre de proyectos?

Tabla 2: Resultados a la pregunta Nº 2

Indicadores	Frecuencia	Porcentaje
Excelente	18	51,4 %
Bueno	17	48,6 %
Regular	0	0,0 %
Malo	0	0,0 %
TOTAL	**35**	**100 %**

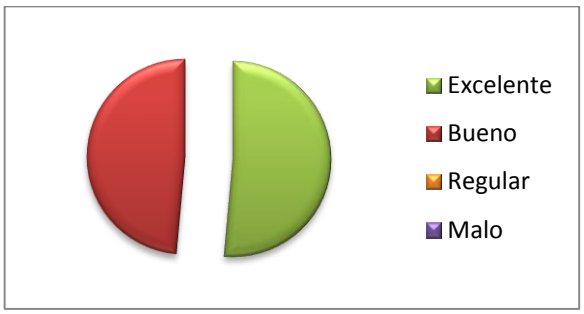


Figura 2: Resultados a la pregunta Nº 2

Quienes optan por “Excelente”, un 51,4 %, lo hacen fundamentándose en que según ellos, la herramienta es una gran ayuda en el modelado de sus proyectos por su facilidad de uso, porque es óptimo, rápido, comprensible y porque posee una interfaz práctica para el usuario. Al hacer pruebas no se presentan inconvenientes con estas acciones.

La categoría “Bueno” fue seleccionada por un 48,6 %, debido a que la herramienta no permite importar proyectos realizados en otras aplicaciones. Pese a ello indican que es una herramienta eficiente que facilita la organización de sus proyectos.

3. ¿Cómo califica la organización y distribución de un proyecto en el árbol “Navegador de Proyectos”?

Tabla 3: Resultados a la pregunta Nº 3

Indicadores	Frecuencia	Porcentaje
Excelente	17	48,6 %
Bueno	17	48,6 %
Regular	1	2,8 %
Malo	0	0,0 %
TOTAL	**35**	**100 %**

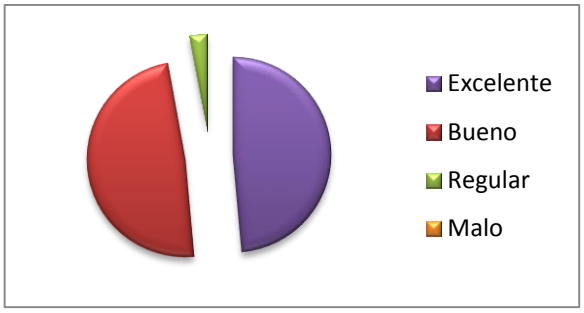


Figura 3: Resultados a la pregunta Nº 3

El 48,6 % de los encuestados afirman que la organización y distribución de un proyecto es “Excelente” ya que gracias a su disposición de componentes se encuentran fácil y rápidamente los elementos creados dentro de un proyecto.

Por otra parte, 17 usuarios correspondientes a otro 48,6 % creen que la organización del “Navegador de Proyectos” es “Buena” por permitir desglosar secuencialmente cada una de las partes de los proyectos creados. Se recomienda añadir iconos más representativos.

Finalmente el 2,8 % que se inclina por la opción “Regular” no explica sus razones.

4. ¿Cuál es su apreciación sobre la apertura, creación, eliminación y cierre de diagramas UML?

Tabla 4: Resultados a la pregunta Nº 4

Indicadores	Frecuencia	Porcentaje
Excelente	24	68,6 %
Bueno	11	31,4 %
Regular	0	0,0 %
Malo	0	0,0 %
TOTAL	**35**	**100 %**

Figura 4: Resultados a la pregunta Nº 4

Un 68,6 % de los usuarios encuestados creen que la administración de diagramas UML es “Excelente” y un 31,4 % cree que es “Buena”.

Según los usuarios la administración de diagramas es rápida, práctica, dinámica, pero sobre todo didáctica. Estas son características que sin duda alguna permiten optimizar en gran medida el tiempo invertido en el modelado de diagramas.

5. ¿Cómo califica la administración de elementos gráficos dentro de un diagrama en edición?

Tabla 5: Resultados a la pregunta Nº 5

Indicadores	Frecuencia	Porcentaje
Excelente	20	57,1 %
Bueno	13	37,1 %
Regular	2	5,8 %
Malo	0	0,0 %
TOTAL	**35**	**100 %**

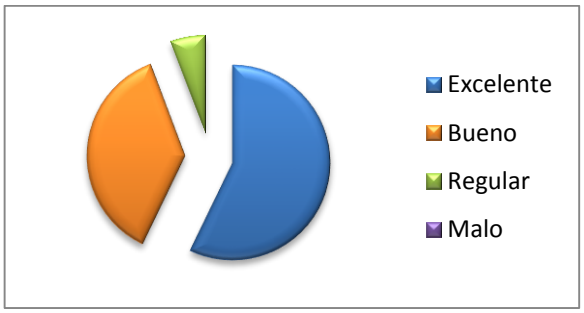


Figura 5: Resultados a la pregunta Nº 5

20 de los encuestados afirman que el manejo de elementos UML dentro de diagramas es “Excelente” puesto que la herramienta brinda un entorno de trabajo agradable, interactivo y entretenido en la edición de diagramas, y consecuentemente, de sus artefactos gráficos.

El 37,1 % de los usuarios que representa a 13 encuestados, contestan a la interrogante con “Bueno”, señalando que se debe incrementar la opción de deshacer acciones.

Finalmente 2 encuestados que representan al 5,8 % manifiestan que los íconos de la Paleta deben ser más grandes y visibles, y principalmente, que AleDan UML no

permite deshacer las últimas acciones realizadas. Por lo tanto, creen que la administración de elementos en un diagrama en edición es “Regular”.

6. La edición y presentación de las propiedades gráficas y estructurales de un artefacto UML dentro de un diagrama en edición es:

Tabla 6: Resultados a la pregunta Nº 6

Indicadores	Frecuencia	Porcentaje
Excelente	18	51,4 %
Bueno	17	48,6 %
Regular	0	0,0 %
Malo	0	0,0 %
TOTAL	**35**	**100 %**

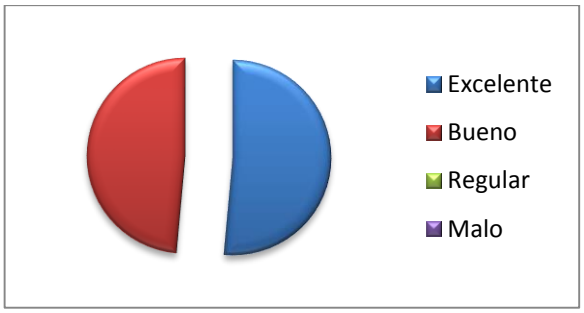


Figura 6: Resultados a la pregunta Nº 6

Un total de 18 encuestados que corresponden al 51,4 % manifiestan que es “Excelente” principalmente por la facilidad para la edición de las propiedades de un artefacto. Indican además que los colores empleados hacen más llamativa esta tarea y en contexto general, hacen más interesante a la herramienta.

Otro 48,6 % perteneciente a 17 encuestados califican esta tarea como “Bueno”, pues aseguran que hace falta implementar algunos componentes en la Paleta.

7. ¿Cómo considera el control de entradas erróneas al momento de ingresar datos?

Tabla 7: Resultados a la pregunta Nº 7

Indicadores	Frecuencia	Porcentaje
Excelente	17	48,6 %
Bueno	13	37,1 %
Regular	5	14,3 %

Malo	0	0,0 %
TOTAL	**35**	**100 %**

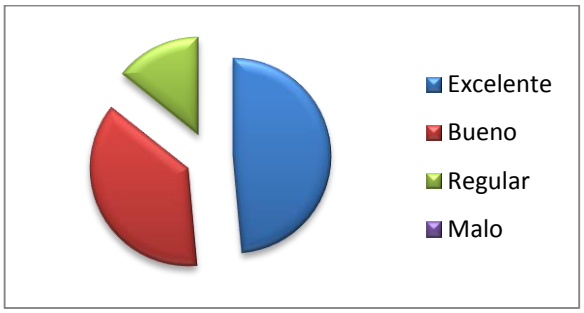


Figura 7: Resultados a la pregunta Nº 7

Puesto que AleDan UML es una herramienta bastante intuitiva, el usuario sabe lo que va a hacer, pero en caso de cometer errores, los controles facilitan saber donde se los ha cometido. Específicamente en el caso de diagramas de clases, la herramienta controla bien el nombre de clases, métodos y atributos, razones por la que un total de 17 encuestados que representan a un 48,6 % califican al control de entradas erróneas como “Excelente”.

Otros 13 usuarios correspondientes al 37,1 % estiman que esta actividad es “Bueno” destacando que los mensajes desplegados en el panel Salida ayudan a realizar bien su trabajo. Además se cree que falta explicación de porqué no se acepta un dato determinado.

Finalmente un 14,3 % representado por 5 encuestados, advierte que no hay mucho control pues se permite ingresar cualquier dato. Esto se puede deber principalmente a la falta de experiencia en el uso y manejo de la herramienta. Otros por su parte, creen conveniente mostrar mensajes más amigables y descriptivos. Por ello en ésta parte de la prueba, se ha calificado a AleDan UML con la categoría “Regular”.

8. ¿Cree usted que la presentación de mensajes informativos, de control, de error, etc. es?

Tabla 8: Resultados a la pregunta Nº 8

Indicadores	Frecuencia	Porcentaje
Excelente	14	40 %

Bueno	20	57,1 %
Regular	1	2,9 %
Malo	0	0,0 %
TOTAL	**35**	**100 %**

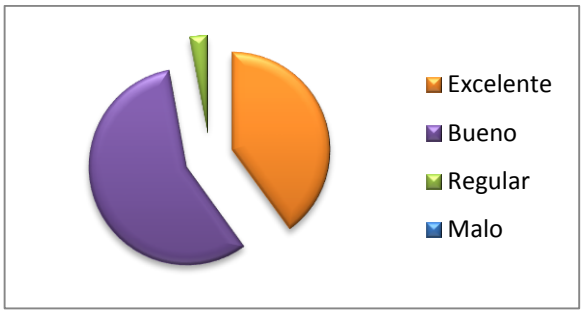


Figura 8: Resultados a la pregunta Nº 8

Como se puede observar en el cuadro y en la representación gráfica correspondiente, 14 encuestados creen que es “Excelente” puesto que de una manera dinámica permite identificar errores cometidos en el diseño de diagramas, y de igual manera, se dan las pautas necesarias para el manejo de la herramienta.

El 57,1 % creen que aunque se ayuda en la corrección de errores y facilita el desarrollo del trabajo, hace falta mejorar aún más en éste campo, para obtener mejores resultados y mayor satisfacción del usuario. En definitiva se sugiere incluir mayor explicación en los mensajes presentados al usuario. Es por ello, que asignan el calificativo “Bueno” a la presentación de mensajes.

El 2,9 % que se inclinó por la opción “Regular” lo hace afirmando que los mensajes deben ser más amigables y descriptivos.

9. ¿Cuál es su valoración para el proceso de exportación de diagramas en formato *.png?

Tabla 9: Resultados a la pregunta Nº 9

Indicadores	Frecuencia	Porcentaje
Excelente	17	48,6 %
Bueno	15	42,9 %
Regular	3	8,5 %
Malo	0	0,0 %
TOTAL	**35**	**100 %**

Figura 9: Resultados a la pregunta Nº 9

El 48,6 % asegura que es “Excelente” puesto que se trata de un proceso rápido y fácil de realizar. Otro 42,9 % considera que es “Bueno” con las mismas consideraciones, añadiendo además que falta un mensaje indicando que la imagen ha sido exportada correctamente o si ha existido algún error. Finalmente un 8,5 % califica este proceso como “Regular” debido a que es necesario que dicha exportación sea realizada en la gran mayoría de formatos gráficos existentes.

10. El proceso de impresión de diagramas es:

Tabla 10: Resultados a la pregunta Nº 10

Indicadores	Frecuencia	Porcentaje
Excelente	20	57,1 %
Bueno	13	37,1 %
Regular	2	5,8 %
Malo	0	0,0 %
TOTAL	**35**	**100 %**

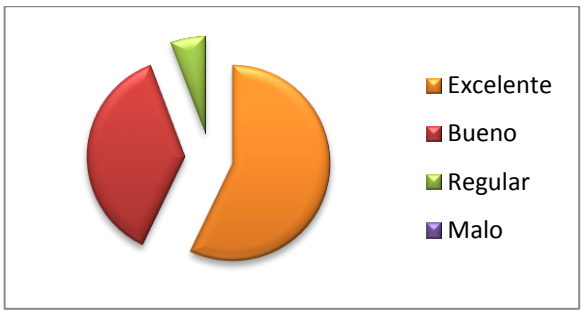


Figura 10: Resultados a la pregunta Nº 10

De acuerdo a 20 usuarios que representan al 57,1 % de la población encuestada, la impresión es “Excelente” puesto que se obtiene una imagen clara, con buena resolución, y más aún, porque se permite configurar el tipo de imagen que se desea obtener.

Otros 13 encuestados que corresponden al 37,1 % aseguran que el proceso es “Bueno” con similares apreciaciones que en la categoría anterior, añadiendo que en ocasiones disminuye la calidad en la impresión, situación que de acuerdo a los mismos encuestados, puede darse por las características de la impresora.

Por último, 2 de los usuarios encuestados creen que la impresión de diagramas es “Regular” aunque no se hayan especificado las razones por las que se lo hizo.

11. ¿Qué calificativo daría usted al módulo de ayuda incluido dentro de la herramienta?

Tabla 11: Resultados a la pregunta Nº 11

Indicadores	Frecuencia	Porcentaje
Excelente	23	65,7 %
Bueno	11	31,4 %
Regular	1	2,9 %
Malo	0	0,0 %
TOTAL	**35**	**100 %**

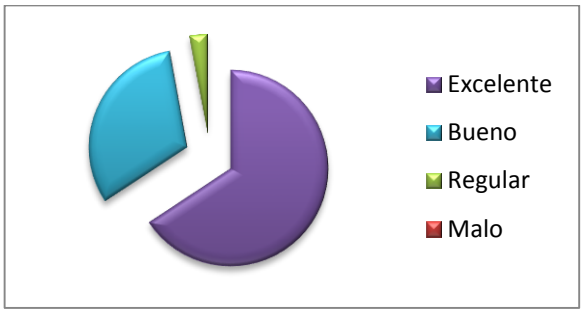


Figura 11: Resultados a la pregunta Nº 11

El 65,7 % que representa a 23 encuestados califican la Ayuda como “Excelente” pues aseguran que es muy explicativa, permite conocer la herramienta en sí, sus características de funcionamiento y su utilidad, está totalmente en español, da mayor comodidad al usuario y a que es muy informativa.

Quienes contestaron con la categoría “Bueno”, el 31,4 %, afirman que la ayuda incluida da las pautas necesarias para el manejo de AleDan UML y apuntan a que se debe mejorar su presentación. De igual manera hay quienes opinan que hace falta explicar más a profundidad la generación de código fuente, o también, lo relacionado a cuándo se puede utilizar asociaciones, herencia, etc.

Aquellos que prefirieron la opción “Regular”, un 2,9%, afirman que se debe mostrar ejemplos prácticos en el uso de la herramienta con cada uno de los diagramas.

12. ¿Cómo considera el grado de coherencia entre el código fuente generado y los diagramas diseñados?

Tabla 12: Resultados a la pregunta Nº 12

Indicadores	Frecuencia	Porcentaje
Excelente	18	51,4 %
Bueno	16	45,7 %
Regular	0	0,0 %
Malo	1	2,9 %
TOTAL	**35**	**100 %**

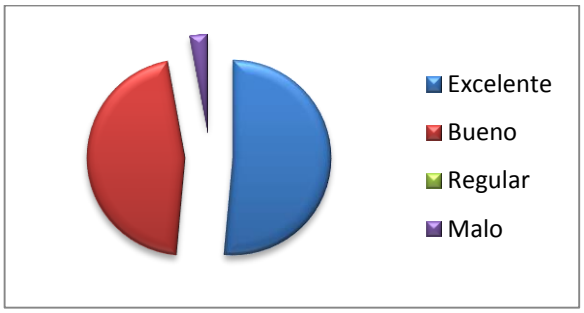


Figura 12: Resultados a la pregunta Nº 12

Como puede observarse en el cuadro estadístico y en el gráfico correspondiente a esta interrogante, 18 usuarios con el 51,4 % afirman que el grado de coherencia es “Excelente” puesto que el código fuente generado refleja lo diseñado en un diagrama, y además, se genera clases con sus respectivas dependencias.

Un porcentaje del 45,7 % señala que es “Bueno” porque el código se genera paralelamente al diseño de diagramas, de tal manera que se pueden percatar inmediatamente del resultado obtenido.

Un 2,9 % que representa a un encuestado, indica por su parte que al no conocer este lenguaje de programación, no puede dar un veredicto real, motivo por el que prefiere asignar como “Malo” el proceso de generación de código fuente.

13. El tiempo de respuesta de la herramienta en la realización de tareas (dibujado de artefactos y diagramas, exportación de diagramas, generación de código e impresión) es:

Tabla 13: Resultados a la pregunta Nº 13

Indicadores	Frecuencia	Porcentaje
Excelente	25	71,4 %
Bueno	9	25,7 %
Regular	1	2,9 %
Malo	0	0,0 %
TOTAL	**35**	**100 %**

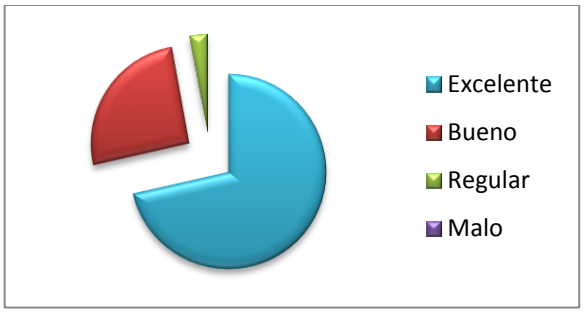


Figura 13: Resultados a la pregunta Nº 13

El 71,4 % de los encuestados califican como “Excelente” el tiempo de respuesta de la aplicación al comprobar que cada una de las funcionalidades que la herramienta ofrece a los usuarios se ejecuta de una manera rápida, eficaz y eficiente.

Quienes optan por el calificativo “Bueno”, un 25,7 %, dan las mismas observaciones anteriores confirmando una vez más que el uso de AleDan UML ahorra tiempo de trabajo.

Otro 2,9 % cree se puede mejorar la rapidez de la aplicación, motivo por el cual se inclina por la opción “Regular”.

14. El desempeño de la herramienta en cuanto al uso de recursos computacionales (memoria RAM, espacio en disco y procesador) es:

Tabla 14: Resultados a la pregunta Nº 14

Indicadores	Frecuencia	Porcentaje
Excelente	21	60 %

Bueno	13	37,1 %
Regular	1	2,9 %
Malo	0	0,0 %
TOTAL	**35**	**100 %**

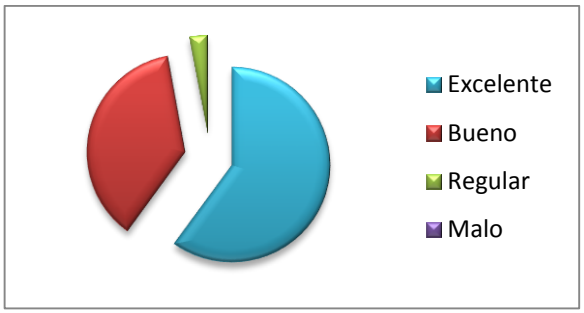


Figura 14: Resultados a la pregunta Nº 14

Según los propios encuestados, en la actualidad la gran mayoría de equipos superan o al menos igualan los requerimientos mínimos de la herramienta AleDan UML. En éste aspecto, un total de 21 encuestados que representan al 60 % supieron responder que AleDan UML es un software bastante liviano, que no ocupa mucho espacio en disco y rápido en su ejecución a pesar de las múltiples ventajas ofrecidas. Indiscutiblemente es “Excelente”.

Por su parte 13 usuarios que simbolizan un 37,1 %, ante esta interrogante responden “Bueno” aunque hacen las mismas apreciaciones señaladas inicialmente. Sin embargo, uno de los usuarios que se inclinó por ésta categoría, indica que al trabajar sobre el sistema operativo Windows XP, AleDan UML consume procesador casi en su totalidad al igual que la memoria RAM.

Por último, un 2,9 % de los desarrolladores encuestados optaron por el calificativo “Regular” afirmando que la herramienta ocupó del 90 al 100 % de su procesador.

15. En comparación con la herramienta UML más utilizada por usted, AleDan UML es:

Tabla 15: Resultados a la pregunta Nº 15

Indicadores	Frecuencia	Porcentaje
Excelente	17	48,6 %
Bueno	18	51,4 %
Regular	0	0,0 %
Malo	0	0,0 %

TOTAL	35	100 %

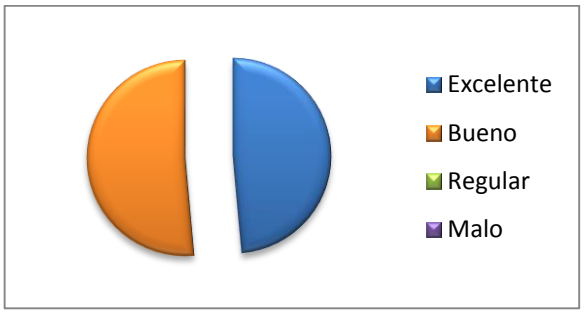


Figura 15: Resultados a la pregunta Nº 15

Un total de 17 encuestados que corresponden al 48,6 % manifiestan que al hacer ésta comparación, AleDan UML es “Excelente” debido a que su interfaz amigable e interactiva, la convierte en una herramienta más fácil de entender y usar, su rapidez ayuda a ahorrar recursos en la etapa de diseño, y en sí, se constituye en una aplicación UML bastante original.

Los 18 usuarios restantes, que simbolizan el 51,4 % de los encuestados, otorgan el calificativo “Bueno” pues creen que se debe implementar la exportación de proyectos realizados en otras aplicaciones, mejorar en el aspecto de las ayudas y optimizar el uso de los recursos computacionales. Creen conveniente además, que se debe dar un mayor tiempo de prueba de la herramienta, de tal manera que se adquiera mayor experiencia en su uso. Sin embargo, se confirma una vez más que es una herramienta totalmente orientada al usuario.

ANEXO 7

ANEXO 8

Análisis FODA de la Herramienta AleDan UML

Fortalezas:

- Interfaz gráfica amigable, interactiva e intuitiva
- Facilidad de uso
- Idioma español
- Administración de proyectos, diagramas y artefactos UML es rápida, práctica, dinámica y didáctica
- Buen control de entradas de datos erróneos
- Exportación de diagramas en formato *.png rápida y fácil de realizar
- Excelente impresión de diagramas, pues se obtiene una imagen clara, con buena resolución, y más aún, porque se permite configurar el tipo de imagen que se desea obtener
- Ayuda explicativa
- Generación de código fuente para el lenguaje de programación C# coherente con lo diseñado en un diagrama
- Los procesos se realizan de una manera rápida, eficaz y eficiente
- Software liviano que no consume muchos recursos computacionales
- Software libre

Oportunidades:

- Implementar mejoras en el módulo de diagramación para permitir la elaboración de los 7 tipos de diagramas restantes: objetos, estructura compuesta, actividad, estados, comunicación, vista de interacción y tiempos
- Implementar módulos para la generación de código fuente en otros lenguajes de programación
- Adaptar la herramienta AleDan UML a las nuevas versiones de la especificación UML
- Mejorar la presentación de mensajes informativos, de control, de error, etc., de tal manera que sean más amigables y explicativos

Debilidades:

- No permite importar proyectos realizados en otras herramientas UML
- No permite deshacer últimas acciones
- Exportar diagramas en la mayoría de formatos gráficos
- El módulo Ayuda no ofrece explicaciones en cuanto a las especificaciones UML 2.0 (uso de relaciones)
- El módulo Ayuda no contiene ejemplos prácticos en el uso de la herramienta con cada uno de los diagramas
- La generación de código fuente se limita al lenguaje de programación C#

Amenazas:

- Falta de interés de desarrolladores de software en mejorar las prestaciones ofrecidas por la herramienta AleDan UML
- Desactualización de la herramienta
- Falta de difusión de la herramienta AleDan UML en la comunidad de desarrolladores de software
- Falta de interés de desarrolladores de software en utilizar AleDan UML

UNIVERSIDAD NACIONAL DE LOJA

ÁREA DE LA ENERGÍA, LAS INDUSTRIAS Y LOS RECURSOS NATURALES NO RENOVABLES

Ingeniería en Sistemas

DESARROLLO DE UNA HERRAMIENTA DE ANÁLISIS Y DISEÑO DE SISTEMAS QUE PERMITA LA DOCUMENTACIÓN DE SOFTWARE BASADA EN UML 2.0 Y LA GENERACIÓN DE CÓDIGO FUENTE BAJO LA ESPECIFICACIÓN DEL LENGUAJE DE PROGRAMACIÓN C SHARP (ECMA – 334)

Autores:

Alexandra Elizabeth Maurad Córdova
Danny Emanuel Muñoz Flores

LOJA – ECUADOR
2008 – 2009

1. Título:

Desarrollo de una Herramienta de análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp (ECMA – 334) .

2. Problemática:

2.1 Situación Problémica

Los continuos avances tecnológicos que se han presentado en las últimas décadas a raíz de la aparición de los computadores de escritorio, han desembocado en que la mayoría de procesos de resolución de problemas sean precisamente a través de un computador, lo que conduce a la escritura de un programa, y a su ejecución en el mismo.

Para la escritura de un programa, a lo largo de la historia se han desarrollado distintos lenguajes de programación basados en diversos paradigmas o formas de pensar. Es así que a comienzos de la década de los 80 surge el paradigma de "Orientación a Objetos", el cual concibe que todo lo que existe en el mundo real son entidades (objetos) que interactúan y se relacionan entre sí. Esto hace que el software sea más fácil de programar, comprender y mantener.

Actualmente la orientación a objetos es sin lugar a duda el paradigma más usado por desarrolladores de software de todo el mundo, y es a partir de éste que nace el lenguaje visual de modelado y documentación de sistemas UML (Lenguaje Unificado de Modelado), con la finalidad de definir estándares de diseño que permitan a todos los desarrolladores hablar en un mismo lenguaje.

Ante ésta situación, la compañía Microsoft desarrolla la plataforma Microsoft .NET, la misma que adopta el paradigma orientado a objetos y componentes, y nace a partir de los lenguajes más aceptados en el mundo tecnológico, como Java, Visual Basic, C++, etc. Es más, también se crea un nuevo lenguaje de programación denominado C Sharp (C#), basándose en la sintaxis de C, C++, Delphi y Java. Posteriormente Microsoft decide liberar la plataforma .NET con la finalidad de que pueda ser adaptada a los diferentes sistemas operativos y lenguajes de programación, es decir, con la finalidad de extender su uso. Gracias a ésta oportunidad surge el proyecto MonoDevelop, el mismo que se constituye en un equivalente de la plataforma .NET de Microsoft, con la diferencia que es de software libre y multiplataforma, es decir, puede ser soportado en diferentes sistemas operativos.

Los desarrolladores de software en general, tratan de crear sus productos con alta calidad y bajo costo, de acuerdo a la planificación inicial y en el tiempo mínimo. Una de las formas de lograr éste objetivo, es formular un correcto diseño del proyecto a realizar, de tal manera que el proceso de construcción sea rápido, ágil, flexible y lo más importante, eficaz. Para ello, dichas empresas deben seleccionar una herramienta de diseño que presente facilidades de uso, que sea eficiente y que se adapte a los requerimientos tanto de los desarrolladores, como del proyecto en marcha, de tal manera que lo enmarque dentro un proceso ordenado y sistemático que asegure el éxito del mismo.

Para el desarrollo de software en el lenguaje de programación C Sharp existen dos alternativas a considerar: por una parte el IDE Visual Studio .NET ofrecido por Microsoft de uso comercial, y por otro, su equivalente en software libre, MonoDevelop. A pesar de que C Sharp es un lenguaje de programación 100% orientado a objetos y a componentes, lamentablemente no existe una herramienta gratuita y open source[31] que ayude a diseñar una arquitectura enfocada a éste paradigma, circunstancia que sin duda puede llevar al desarrollador a caer en una programación estructurada a pesar de trabajar con un lenguaje orientado a objetos: el IDE Visual Studio .NET únicamente ofrece la posibilidad de realizar un diagramado de clases, dejando de lado otros diagramas que son de igual importancia para el ingeniero en sistemas especializado en el desarrollo de software; por su parte el IDE MonoDevelop no cuenta con un módulo de diagramación. Aunque algunas empresas han desarrollado plugin's (módulos) con soporte UML, ya sea para Visual Studio o MonoDevelop, lo que persiguen es una utilidad comercial y privada.

A pesar de que algunas herramientas permiten la construcción de la mayoría del diseño de una aplicación software, no poseen una interfaz amigable, por lo que su uso puede tornarse demasiado complejo y confuso para los usuarios; en otros casos, el alto costo que supone la adquisición de una herramienta de éste tipo, puede convertirla hasta cierto punto en inalcanzable para los desarrolladores, razón por la que son utilizadas mediante licencias ilegales, o en su defecto sin licencia, mediante "demos" que tienen poca duración; adicionalmente, es importante que la Universidad Nacional de Loja disponga de una herramienta de diseño propia de carácter libre,

[31] Las herramientas open source o de código abierto permiten modificar el código fuente para implantar mejoras o nuevas funcionalidades a las mismas.

dando a otros desarrolladores la posibilidad de hacer mejoras significativas que aporten positivamente al funcionamiento y facilidad de uso de la misma.

A esto se suma el hecho de que los desarrolladores de software, especialmente aquellos que están iniciándose en ésta actividad, no saben por qué herramienta decidirse a la hora de realizar su trabajo, independientemente de que se cuente o no con una versión legal. Ciertas herramientas si bien soportan el lenguaje de programación C Sharp, no permiten trabajar con los diagramas UML más utilizados (clases, paquetes, componentes, despliegue, casos de uso y secuencia) o viceversa, es decir que a pesar de trabajar con dichos diagramas, no soportan la generación de código fuente para C Sharp, lo que conlleva a escribir manualmente el modelado.

Esta falta de herramientas incrementa el riesgo en los desarrolladores de adquirir malos hábitos durante la construcción de software, pues en lugar de seguir un proceso ordenado y metodológico, optan por codificar el proyecto para luego proceder a ejecutar la etapa de análisis y diseño; mientras que en el peor de los casos, dejan de lado la documentación de la arquitectura del sistema informático. El único resultado esperado de ésta situación es la pérdida de tiempo debido a un mal diseño y, consecuentemente, retrasos significativos en la planificación, ya que si en una etapa posterior se detectan errores en el diseño, se deberá volver a realizar el mismo trabajo dos o hasta más veces.

Enmarcados en éste contexto es que proponemos como tema de proyecto de tesis el **“Desarrollo de una Herramienta de análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp (ECMA – 334)”**, con el cual pretendemos crear una herramienta de diseño capaz de satisfacer los requerimientos de los desarrolladores que se orientan a realizar su trabajo en el lenguaje de programación C Sharp, ayudando a minimizar los riesgos de retrasos en la planificación asociados al mal diseño. De igual manera, se pretende aportar significativamente a que los desarrolladores de software, y en especial los estudiantes de la Carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja, adopten buenos hábitos en el desarrollo de software, dejando de lado la denominada “ingeniería inversa”.

2.2 Problema General de Investigación

Falta de una Herramienta para el análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp.

2.3 Delimitación

2.3.1 Problemas específicos de Investigación

Luego del análisis de la problemática planteada, hemos podido detectar algunos inconvenientes, los mismos que a nuestro criterio, han sido definidos como problemas específicos de investigación:

- La falta de conocimiento por parte de los desarrolladores de software, acerca de las herramientas para análisis y diseño de sistemas que soportan C Sharp y UML 2.0.
- La falta de una herramienta de análisis y diseño de sistemas que soporte el lenguaje de programación C Sharp y que permita diagramar la arquitectura de un producto software mediante la especificación UML 2.0 para la documentación de diagramas de clases, paquetes, componentes, despliegue, casos de uso y secuencia, causa que los desarrolladores adquieran malos hábitos en el proceso de construcción de software.
- La falta de una herramienta que permita la generación de código fuente para el lenguaje de programación C Sharp a partir del diseño de diagramas de clases, paquetes, componentes, despliegue, casos de uso y secuencia, basados en UML 2.0.

2.3.2 Espacio

El proceso investigativo denominado **"Desarrollo de una Herramienta de análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp (ECMA – 334)"**, será realizado en la Carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja tomando como base los diagramas UML

2.0 más utilizados en la misma (diagrama de clases, diagrama de paquetes, diagrama de componentes, diagrama de despliegue, diagrama de casos de uso y diagrama de secuencia), y estará dirigido a todos los desarrolladores de software que deseen utilizarla, especialmente a quienes pertenecen a dicha entidad, ya sean docentes o futuros profesionales de ésta rama.

2.3.3 Tiempo

El desarrollo de la presente investigación estará comprendido durante el período Noviembre del 2008 hasta Agosto del 2009.

2.3.4 Unidades de Observación

- Herramientas CASE
- Herramientas CASE para el diseño de la arquitectura de un producto software
- Lenguaje Unificado de Modelado UML 2.0
- Especificación del Lenguaje de Programación C Sharp (ECMA – 334)

3. Justificación

La Universidad Nacional de Loja en su afán de formar profesionales capaces de dar solución a las problemáticas presentadas en nuestra sociedad, impulsa en cada una de las carreras que oferta el desarrollo de investigaciones de carácter científico – técnico que coadyuven al desarrollo sustentable de nuestro entorno y contribuyan a su adelanto. Es así que en la carrera de Ingeniería en Sistemas, se capacita a los estudiantes de tal forma que a través de conceptos propios de sistemas, proporcionen alternativas válidas para dar solución a diversas dificultades que pueden llegar a presentarse en empresas tanto públicas como privadas, ya sea mediante la implantación de sistemas informáticos o mediante el planteamiento de propuestas para el mejoramiento de la situación problemáticas existentes.

Los constantes avances tecnológicos que se han dado en las últimas décadas han contribuido a que empresas tanto públicas como privadas incrementen su productividad, ya sea mediante la adquisición de nuevos equipos (hardware) o mediante la adquisición e implantación de tecnologías de información (software). Pos su parte las empresas desarrolladoras de software incrementan su competitividad mediante la utilización de herramientas que permiten optimizar recursos y obtener productos de calidad. Las empresas más sobresalientes de desarrollo de software son las que utilizan buenas herramientas (herramientas CASE) que apoyen a una etapa específica o a todo el ciclo de desarrollo de un producto.

Ante la necesidad de mejorar el rendimiento en todos los sectores productivos, dependemos cada vez más del software en nuestras actividades cotidianas, razón por la que la fiabilidad del software es vital. Sin embargo, y pese a los recursos y esfuerzos invertidos, gran parte de los proyectos de desarrollo de software no llegan a cumplir los objetivos planteados, debido a errores de diseño detectados en etapas posteriores a éste. Una de las posibles alternativas de solución a éste inconveniente, es el correcto diseño de la arquitectura del software a desarrollar, para lo que se requiere el apoyo de una herramienta eficiente capaz de satisfacer, en éste caso específico, las necesidades de los desarrolladores que deban o quieran realizar su trabajo en el lenguaje de programación C Sharp.

La investigación denominada **“Desarrollo de una Herramienta de análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la**

generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp (ECMA – 334)”, fue propuesta fundamentalmente con el ánimo de proporcionar a los estudiantes que forman parte de la carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja y a los desarrolladores de software en general, una herramienta de diseño de software que contenga y brinde a los usuarios los instrumentos necesarios para diseñar la arquitectura de una aplicación software, además de generar automáticamente el código fuente del diseño realizado previamente en el lenguaje C Sharp. Con esta herramienta, se espera minimizar el riesgo de que los desarrolladores adopten el mal hábito de la “ingeniería inversa”, es decir, primero codificar y luego realizar el análisis y diseño del software. Además, ya que la herramienta a desarrollar será libre, queda abierta la posibilidad de que a futuro se integren nuevos módulos que le permitan la generación de código para la gran diversidad de lenguajes existentes, o para realizar modificaciones que mejoren la funcionalidad y operatividad de ésta herramienta.

Para conseguirlo, nos apoyaremos en los conocimientos adquiridos dentro y fuera de las aulas a lo largo del curso de nuestra carrera, así como también en los que se han adquirido en el desarrollo de proyectos investigativos de carácter científico – técnico realizado en los diferentes módulos como parte del cumplimiento del Sistema Académico Modular por Objetos de Transformación (SAMOT). Asimismo, se continuará investigando constantemente con la finalidad de recopilar la información necesaria para abordar la temática eficientemente.

Con ésta herramienta de diseño, los desarrolladores que la utilicen podrán realizar su trabajo en base a un diseño adecuado que minimice el riesgo de retrasos y pérdida de tiempo, y por lo tanto, se incrementarán las posibilidades de obtener un producto exitoso, en el tiempo y con los recursos estimados en la planificación establecida inicialmente, lo que consecuentemente se refleja en la satisfacción del cliente al constatar el incremento en la productividad de su empresa. Además es importante que la Universidad Nacional de Loja cuente con una herramienta de diseño libre, de tal manera que los desarrolladores de software puedan realizar las modificaciones necesarias para obtener una mejor funcionalidad en la misma.

Viabilidad Económica

Se trata de un proyecto viable puesto que para la construcción de ésta herramienta no se necesita la adquisición de tecnología avanzada en cuanto a hardware y software, es decir, el desarrollo se lo realizará con la ayuda de equipos propios y con herramientas de software libre. Los recursos económicos que deberán ser invertidos a lo largo del desarrollo del proyecto, serán financiados por nuestro grupo de trabajo, de tal manera que no se correrá el riesgo de paralizar el desarrollo de la herramienta debido a limitaciones presupuestarias.

Viabilidad Técnica

Nuestro grupo de trabajo puede realizar el presente proyecto puesto que contamos con los conocimientos necesarios para el manejo de los distintos recursos a aplicar, esto es, hardware, software, Lenguaje Unificado para el Modelado (UML) y Lenguaje de Programación C Sharp. De igual manera, disponemos de los recursos técnicos (hardware y software) para el desarrollo de la herramienta.

Viabilidad Operativa

Al ser estudiantes de la Carrera de Ingeniería en Sistemas, la investigación será operativamente factible, pues tenemos fácil acceso tanto a la Internet como a las diversas fuentes de información existentes en la Biblioteca del Área de la Energía, las Industrias y los Recursos Naturales no Renovales.

Asimismo contamos con el apoyo de los estudiantes de los últimos módulos de la carrera y de los docentes que laboran en la misma, cuyos valiosos aportes facilitarán el proceso de investigación. Por otra parte, los desarrolladores de software, especialmente los futuros Ingenieros en Sistemas de la Universidad Nacional de Loja, estarán en capacidad de manejar adecuadamente la nueva herramienta puesto que poseen los conocimientos necesarios tanto del diseño de aplicaciones software como de los lenguajes C Sharp y UML 2.0.

La herramienta podrá ser construida en el tiempo estimado, ya que como desarrolladores de la tesis, estamos comprometidos con la misma, disponemos del tiempo planificado para el efecto y lo dedicaremos en su totalidad a la ejecución de la

investigación. Así mismo, al ser un proyecto que no involucra a una empresa en particular, podremos realizar las tareas en el momento apropiado, ya sea en nuestra dependencia o como trabajo de campo, minimizando así la presencia de retrasos significativos. Para el desarrollo de la herramienta vamos a tomar en consideración las ventajas y fortalezas de las mejores herramientas existentes en el medio para el diseño y análisis de sistemas.

Finalmente es importante señalar que ésta herramienta será multiplataforma, es decir, podrá ser ejecutada en los sistemas operativos Windows o Linux luego de ser instalada en el ordenador.

4. Objetivos

4.1 General

- Desarrollar una Herramienta de análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp (ECMA – 334).

4.2 Específicos

- Analizar las herramientas de análisis y diseño de sistemas basadas en la especificación UML 2.0 más utilizadas por los desarrolladores de software.

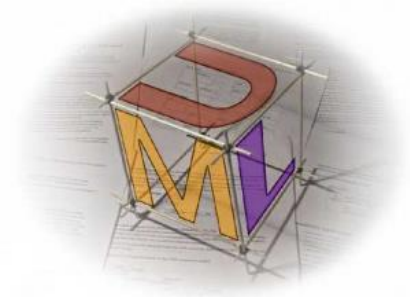
- Desarrollar un módulo de diagramación basado en el Lenguaje de Programación C Sharp, y en la especificación UML 2.0 para modelar diagramas de clases, paquetes, componentes, despliegue, casos de uso y secuencia.

- Desarrollar un módulo de generación de código fuente para el Lenguaje de Programación C Sharp, a partir de los diagramas de clases y secuencia.

5. Marco Teórico

CAPÍTULO I

Lenguaje Unificado de Modelado UML



5.1.1 Definiciones

Un modelo es la simplificación de la realidad observada. En el campo del desarrollo de software, el objetivo que persigue el modelado de un sistema es obtener las piezas más importantes del mismo. Para ello se procede a realizar una abstracción de esa realidad y se la plasma con una notación gráfica.

A ésta representación se la conoce como modelo visual y lo que permite es manejar la complejidad de los sistemas a analizar o diseñar. De ahí es que nace la necesidad de contar con una herramienta que permita visualizar, comprender y entender que será y cómo se hará un sistema, antes de ponerlo en marcha.

El **Lenguaje Unificado de Modelado UML (Unified Modeling Language)** nace cuando Rumbaugh, Booch y Jacobson unifican sus estudios basándose en una semántica y notación estándares, con el objetivo de lograr compatibilidad en el análisis y diseño orientado a objetos. De ésta manera se consigue que los proyectos sean realizados en un lenguaje de modelado maduro, dando a los desarrolladores la oportunidad de enfocarse en producir software de calidad.

Es un lenguaje gráfico estandarizado y orientado a objetos, que ayuda a escribir los "planos" del software, con la finalidad de visualizar, especificar, construir y documentar los artefactos de un sistema de información. "Incluye conceptos conceptuales tales como procesos de negocio y funciones del sistema, y algunos aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables".[32]

[32] WIKIPEDIA. 2008 [en línea]. Lenguaje Unificado de Modelado. [http://es.wikipedia.org/wiki/UML], [Consulta: 28 de noviembre del 2008]

UML está conformado por varios elementos, artefactos o figuras que tienen un significado propio. Mediante la agrupación de algunos de estos elementos (según sea necesario), se pueden obtener diagramas que modelan diferentes aspectos del sistema a desarrollar, desde sus vistas lógicas y físicas, hasta los aspectos dinámicos, estáticos y funcionales del mismo.

Esta notación gráfica muy expresiva, también permite representar en mayor o menor grado todas las fases que un proyecto informático debería seguir para ser exitoso: por ejemplo, apoyarnos en los casos de uso durante la etapa de análisis; en diagramas de clases, objetos, etc., en la fase de diseño; o definir la implementación y configuración del sistema mediante diagramas de despliegue.

UML es totalmente independiente del lenguaje de programación, es decir, los diseños realizados utilizando éste estándar pueden ser implementados en cualquier lenguaje, principalmente los orientados a objetos.

5.1.2 Historia

UML tuvo sus inicios en octubre de 1994, cuando dos investigadores de la metodología del software, Rumbaugh y Booch, decidieron unificar los métodos que cada uno de ellos había desarrollado por separado: el método OMT (Object Modelling Tool) y el método Booch.

En octubre de 1995 suceden dos acontecimientos importantes: por una parte se presenta el primer borrador del lenguaje UML, y por otra, Jacobson, otro entusiasta del área de la metodología del software, se une al equipo para aportar algunas de sus ideas al lenguaje en desarrollo. Con el avance del proyecto, se solicitó la colaboración de varias empresas para que aportaran ideas, recomendaciones o sugerencias, con el propósito de enriquecer UML.

Con todas estas colaboraciones de por medio, se pudo concretar la primera versión del lenguaje gráfico para el modelado UML, la misma que fue entregada a un grupo de trabajo del OMG (Object Management Group). Este grupo formuló algunas modificaciones para que sean incrementadas en una nueva versión de UML (la 1.1), la misma que sería finalmente adoptada por el OMG como estándar en noviembre de

1997. En la actualidad se cuenta con la versión 2.1 de UML, la misma que fue lanzada en el año 2007.

5.1.3 Funciones de UML

- Permite que los desarrolladores representen gráficamente un sistema, de tal manera que otro u otros lo puedan leer.
- Permite especificar las características de un sistema antes de su construcción.
- A partir de los modelos realizados en éste lenguaje, se pueden construir los sistemas diseñados.
- Permite documentar el sistema desarrollado a través de sus elementos gráficos.

5.1.4 Estructura de un Modelo

Un modelo está compuesto por tres diferentes bloques de construcción:

- **Elementos,** que son abstracciones de los objetos o entidades que se desea representar.
- **Relaciones,** que permiten relacionar entre sí a los objetos o entidades representados.
- **Diagramas,** que son un conjunto de elementos con sus respectivas relaciones.

5.1.5 Diagramas

Un diagrama es la representación gráfica de un sistema o subsistema, que brinda la posibilidad de comprender de una mejor manera cómo interactúan sus elementos en la realidad (sistema actual) y cómo se desea que interactúen en el sistema a desarrollar.

Lo realmente valioso de un diagrama es su significado y no su imagen gráfica.

Con el propósito de realizar una representación correcta de un sistema, la versión 2.0 de UML ofrece trece diferentes tipos de diagramas para visualizar nuestro sistema desde varios puntos de vista. Estos diagramas se organizan jerárquicamente tal como se muestra en la figura 1.1:

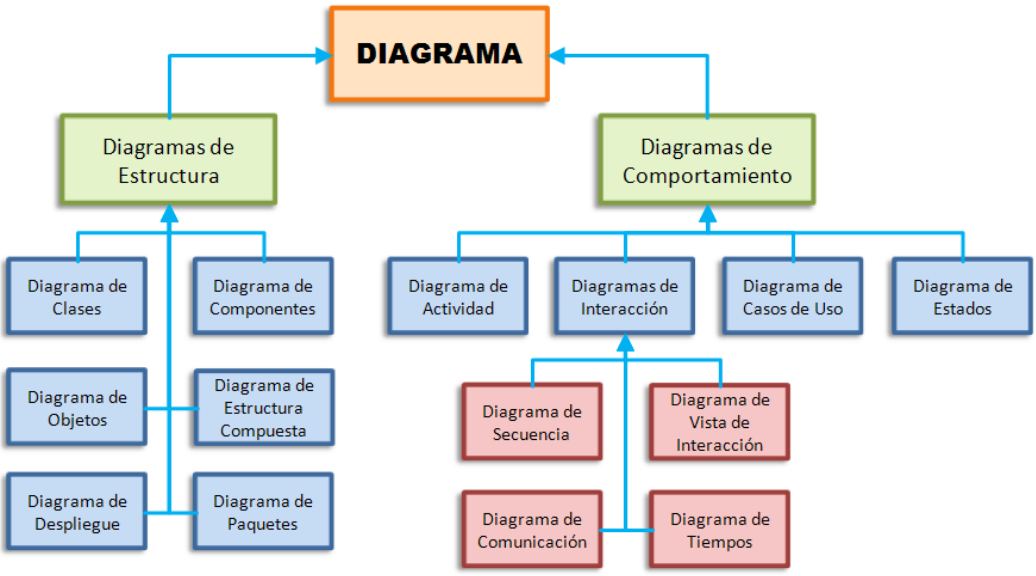


Figura 1.1: Tipos de Diagramas en UML 2.0

5.1.5.1 Diagramas de Estructura

Los Diagramas de Estructura se centran en enfatizar los elementos que deben existir en el sistema modelado, y son los siguientes:

5.1.5.1.1 Diagrama de Clases

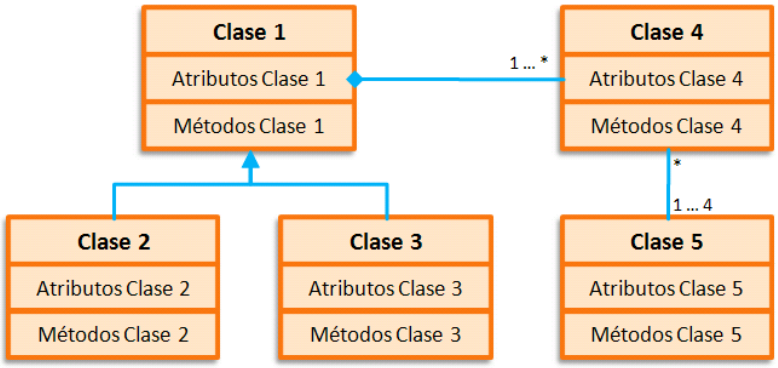


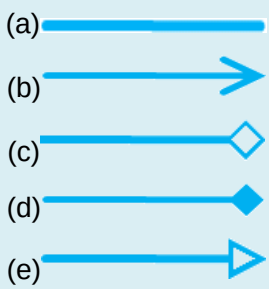
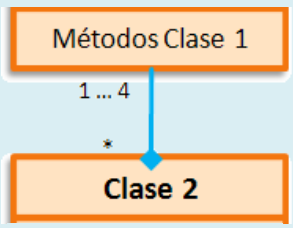
Figura1.2: Diagrama de Clases

Éste diagrama (ver Figura 1.2) nos proporcionan una vista estática del sistema, puesto que representa gráficamente cómo los objetos o entidades interactúan entre sí (sus relaciones). Para ello, el analista deberá definir las características de cada una de las clases y sus relaciones.

Se dice que se trata de un modelado estático porque no se indican los cambios que van a tener los objetos durante su ejecución.

Los elementos de un Diagrama de Clases son los mostrados en la tabla 1.1:

Tabla 1.1: Elementos de un Diagrama de Clases

Artefacto	Descripción
	**Clase.-** Éste artefacto representa cualquier objeto o entidad de la realidad que se desea modelar. En el primer casillero se deberá hacer constar el nombre de la clase; en el segundo, los atributos; y, en el tercer casillero se indicarán los métodos que posee la clase.
	**Relación.-** Es la manera cómo interactúa cada una de las clases. Éstas pueden ser representadas de 5 diferentes maneras: asociación simple (a y b), agregación (c), composición (d) y herencia (e). Adicionalmente, se puede presentar el nombre de la relación (descripción clara y precisa de la relación que existe entre sus objetos) y la navegabilidad (grado de conocimiento que tiene un objeto respecto a otro, en la relación) que puede ser unidireccional o bidireccional.
	**Multiplicidad.-** Permite identificar cuántos objetos intervienen en una asociación. Ésta puede ser de los siguientes tipos: **1.** 0 … 1 → 0 a exactamente 1 **2.** 1 → exactamente 1 **3.** 0 … * → 0 a muchos **4.** 1 … * → 1 a muchos **5.** n → exactamente n objetos, donde n > 0 **6.** 0 … n → 0 a n, donde n > 0 **7.** 1 … n → 1 a exactamente n, donde n > 1

5.1.5.1.2 Diagrama de Componentes

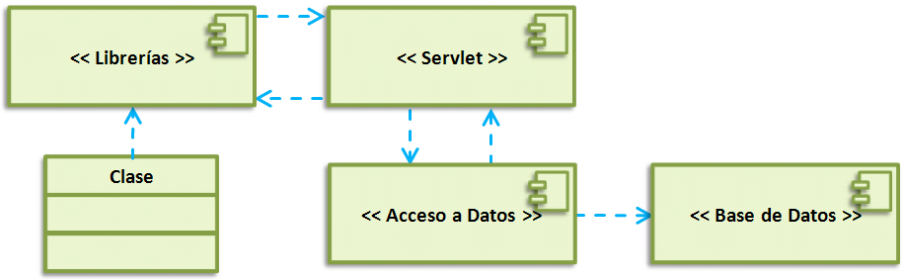


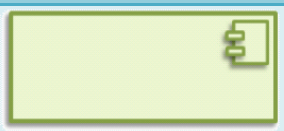
Figura 1.3: Diagrama de Componentes

Permite a los analistas realizar el diseño de cómo quedará un sistema en su parte física. Para lograr éste objetivo se deben incluir las librerías utilizadas, tablas, archivos, clases, paquetes, etc., de tal manera que se represente cómo se encuentran organizados los componentes del sistema. Normalmente los diagramas de componentes (ver Figura 1.3) se utilizan para modelar código fuente, código ejecutable y bases de datos físicas.

Para mayor comprensión de quienes se apoyen en éste diagrama, cada componente deberá ser identificado por medio de cualquiera de éstos estereotipos: ejecutable <<executable>>, tabla <<table>>, librería <<library>>, archivo <<file>>, documento <<document>>, etc.

Los elementos que se pueden utilizar en éste tipo de diagrama son los enunciados en la tabla 1.2:

Tabla 1.2: Elementos de un Diagrama de Componentes

Artefacto	Descripción
	**Componente.-** Representa las librerías, clases, paquetes, archivos java, etc., que formarán parte de la aplicación.
	**Relación.-** Indica que un componente se refiere a los servicios ofrecidos por otro componente.

5.1.5.1.3 Diagrama de Objetos

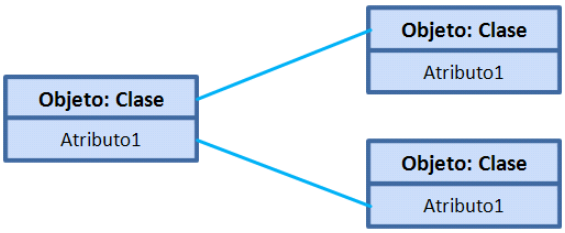
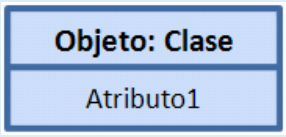


Figura 1.4: Diagrama de Objetos

Es un diagrama que muestra la relación existente entre objetos en un momento determinado de la aplicación (ver Figura 1.4). Para realizar éste tipo de diagrama, el analista deberá decidir previamente cuál es la situación del sistema que se desea representar.

Los elementos gráficos que se deben tomar en consideración para elaborar un diagrama de objetos son los descritos en la tabla 1.3:

Tabla 1.3: Elementos de un Diagrama de Objetos

Artefacto	Descripción
	**Objeto.-** Representa las instancias creadas en un momento determinado de la ejecución. En el casillero superior se coloca el comportamiento en la forma [Nombre de objeto: Nombre de clase].
	Relación.- Muestra la interacción que se da entre los diferentes objetos existentes en un momento dado.
	**Clase.-** Éste artefacto puede ser utilizado opcionalmente cuando se desee mostrar explícitamente la clase a partir de la cual se extiende uno o varios objetos.

5.1.5.1.4 Diagrama de Estructura Compuesta

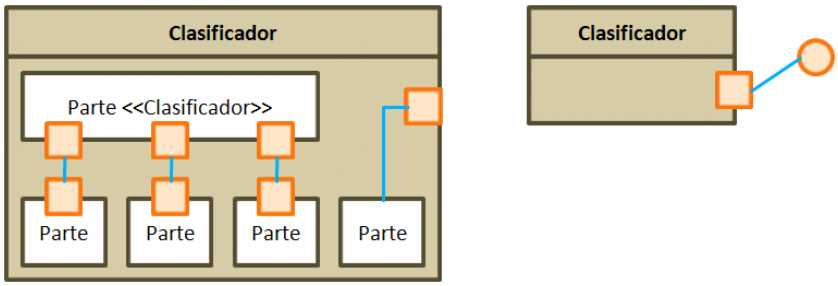
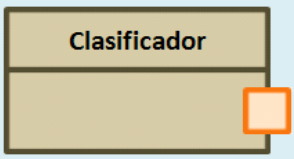


Figura 1.5: Diagrama de Estructura Compuesta

El diagrama descrito en la figura 1.5 muestra la estructura interna de una clase y las colaboraciones que hace posible. Una estructura compuesta es una colección de elementos (con un papel bien definido) que interactúan entre sí para lograr el propósito para el cual fueron creados.

Los artefactos utilizados son los que se describen en la tabla 1.4:

Tabla 1.4: Elementos de un Diagrama de Estructura Compuesta

Artefacto	Descripción
	**Parte.-** Representa el papel que cumple un objeto en un momento dado. Puede incluir un factor de multiplicidad (cardinalidad), por ejemplo [0…*].
	**Puerta.-** Es un punto de interacción que conecta clasificadores con sus partes y con el ambiente. Pueden describir los servicios que proveen y los que necesitan de otras partes del sistema. Son de dos tipos: públicas (mostradas sobre el borde) o protegidas (mostradas dentro del borde).
	Conector.- Une dos o más entidades que interactúan en un momento determinado.
	**Colaboración.-** Contiene los papeles que los objetos pueden cumplir en la colaboración.
	**Clasificador Estructurado.-** Representa una clase, generalmente abstracta, cuyo comportamiento puede ser completa o parcialmente descrito mediante interacciones entre partes. Su principal tipo es el Clasificador Encapsulado el cual se caracteriza por poseer puertas.

5.1.5.1.5 Diagrama de Despliegue

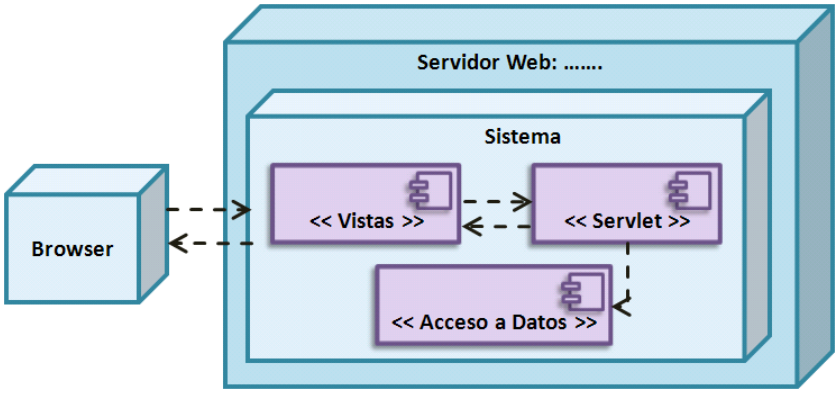
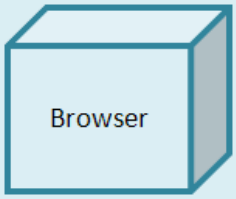


Figura 1.6: Diagrama de Despliegue

Ayuda a visualizar la distribución física de la aplicación, a través de nodos que muestran la disposición tanto en hardware como en software de nuestro sistema (ver figura 1.6). Antes de realizar éste diagrama, el analista deberá previamente haber realizado el diagrama de componentes respectivo.

Un diagrama de despliegue está conformado por los elementos mostrados en la tabla 1.5:

Tabla 1.5: Elementos de un Diagrama de Despliegue

Artefacto	Descripción
	**Nodo.-** Representa un recurso computacional que tiene algo de memoria y capacidad de procesamiento o un dispositivo sobre el que se pueden desplegar y ejecutar los componentes del sistema. Los estereotipos que usan son: procesadores (capacidad de procesamiento) o dispositivos (nodo sin capacidad de procesamiento).
	**Componente.-** Representa los elementos que participan en la ejecución del sistema, y el empaquetamiento físico de los mismos.
	**Relación.-** Puede ser unidireccional o bidireccional, y se muestra como una relación de dependencia o, listando los nodos desplegados en un compartimiento adicional dentro del nodo.

5.1.5.1.6 Diagrama de Paquetes

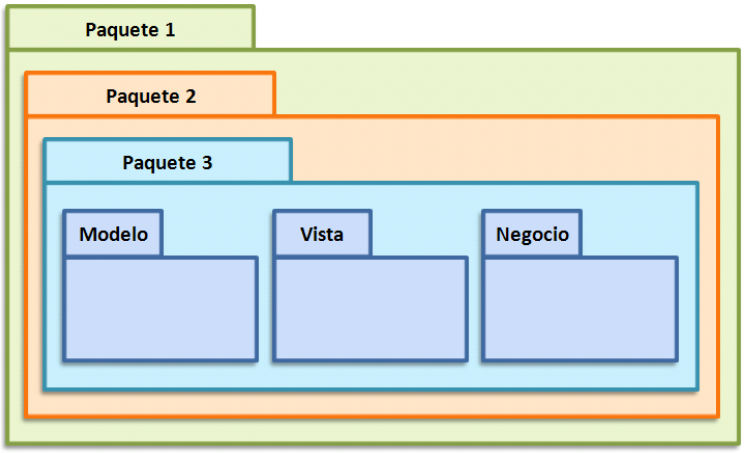
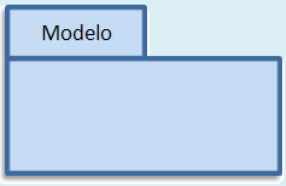


Figura 1.7: Diagrama de Paquetes

Es un tipo de diagrama que indica gráficamente la organización de los paquetes y de sus clases, puesto que muestra cómo un sistema está dividido en agrupaciones lógicas y las dependencias entre esas agrupaciones (Figura 1.7).

Pese a que varios autores incluyen dentro de éste diagrama las clases y relaciones que contienen los paquetes involucrados en el sistema, para una mayor comprensión, hemos creído conveniente incluir únicamente a los paquetes como artefacto gráfico, y es en la tabla 1.6, donde se hace una breve descripción de éste componente gráfico:

Tabla 1.6: Elementos de un Diagrama de Objetos

Artefacto	Descripción
	**Paquete.-** Permite agrupar clases o componentes que forman parte de un subsistema en particular. Generalmente, dentro de todo un conjunto de paquetes, se encuentran tres principales: modelo (agrupa las clases relacionadas con el dominio del sistema real), vista (contiene la interfaz gráfica de usuario) y negocio (contiene las reglas del negocio).

5.1.5.2 Diagramas de Comportamiento

Los **Diagramas de Comportamiento** por su parte, describen lo que debe suceder en el sistema modelado. En ésta categoría constan los siguientes:

5.1.5.2.1 Diagrama de Actividades

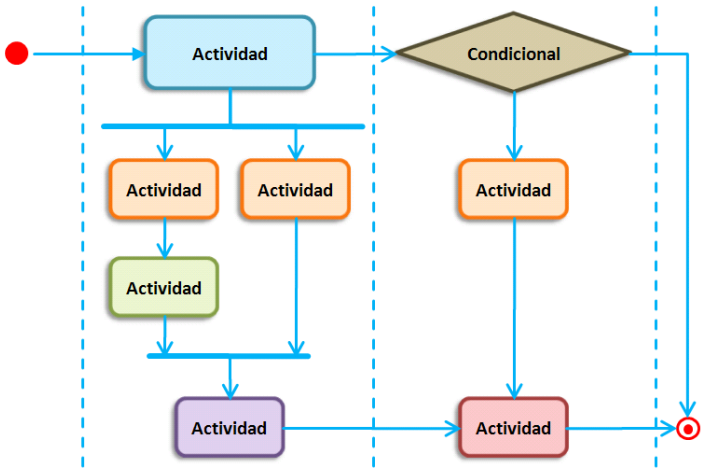
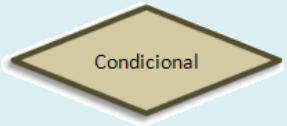


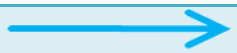
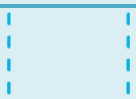
Figura 1.8: Diagrama de Actividades

Como se puede apreciar en la Figura 1.8, éste tipo de diagrama muestra gráficamente el orden en el que se van a ejecutar las actividades dentro de un evento determinado. Permite modelar un sistema, un escenario o un proceso.

Los elementos utilizados son los que se muestran a continuación, en la tabla 1.7:

Tabla 1.7: Elementos de un Diagrama de Actividades

Artefacto	Descripción
	**Estado Inicial.-** Señala el estado inicial de la actividad que se va a representar.
	**Actividad.-** Representa la operación que se debe ejecutar. Pueden llegar a formar jerarquías, es decir, puede estar formada a su vez por varias actividades.
	**Ramificación (Condicional).-** Ocurre cuando existe la posibilidad de que ocurra más de una transición al terminar la ejecución de una actividad determinada. El cambio podrá darse a una actividad o a la otra.
	División (Fork).- Al igual que la anterior, ocurre cuando existe la posibilidad de que ocurra más de una transición al terminar una actividad, pero con la diferencia de que el cambio deberá darse obligatoriamente por todos los caminos existentes.
	Unión.- Representa la fusión de dos o más actividades.

	**Transición.-** Indica el cambio de una actividad a otra.
	**Pasarelas o Calles.-** Agrupan las actividades de acuerdo a su responsabilidad, en regiones distintas.
	**Estado Terminal.-** Éste artefacto representa el final del diagrama de actividades, es decir, en donde finaliza la actividad representada.

5.1.5.2.2 Diagrama de Casos de Uso

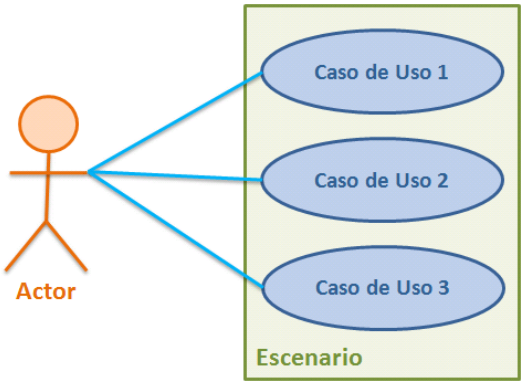


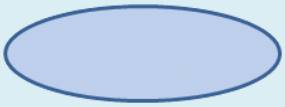
Figura 1.9: Diagrama de Casos de Uso

Un caso de uso es la interacción con el sistema a desarrollar, donde se representan los requisitos funcionales del mismo. Un diagrama de Casos de Uso (ver Figura 1.9) representa gráficamente los casos de uso que tiene un sistema, y nos ayuda a documentar el comportamiento que tiene el sistema actual (análisis) o el comportamiento que va a tener el sistema a desarrollar (diseño), a través de la identificación de sus actores y los escenarios que se van a presentar.

Los artefactos existentes dentro de un Diagrama de Casos de Uso se describen en la tabla 1.8:

Tabla 1.8: Elementos de un Diagrama de Casos de Uso

Artefacto	Descripción
	**Actor.-** Representa cualquier entidad que puede interactuar con los casos de uso. El artefacto con el cual se representa, se denomina “stick”, y su nombre deberá identificar claramente el rol que desempeña dentro del sistema. Puede ser de tres

	diferentes tipos: 4. **Primario o Principal,** que actúa directamente sobre el sistema y tiene metas completamente definidas sobre él a través de los servicios. Éste maneja completamente el caso de uso. 5. **De soporte,** que provee información externa que tiene relación con el sistema. 6. **Externo o Silencioso,** que tiene cierto interés por el comportamiento del caso de uso, por lo tanto no es primario ni de soporte.
	Relación.- Es la manera cómo interactúa el actor con los diferentes casos de uso identificados en el sistema.
	**Escenario.-** Es una secuencia de acciones que se dan entre el actor y el sistema. Define claramente el ámbito del sistema, es decir, delimita hasta dónde va a llegar.
	**Caso de Uso.-** Es una narración hablada o escrita del comportamiento que tiene o que va a tener el escenario. Describe una secuencia de acciones que proveen una medida de valor para un actor, y es más, obligatoriamente deben dejar algo de valor para el sistema.

5.1.5.2.3 Diagrama de Estados

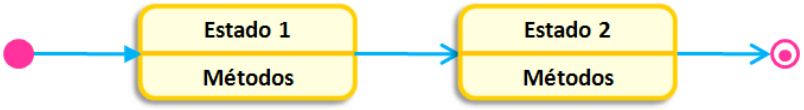
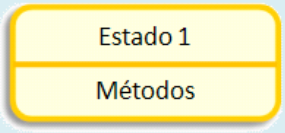
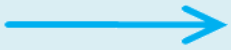


Figura 1.10: Diagrama de Estados

Capturan el ciclo de vida de uno o de más objetos y expresan los estados que va a tener el objeto durante su tiempo de ejecución, es decir, muestra cómo se va a comportar. Para realizar un diagrama de objetos (ver Figura 1.10), previamente se deberán identificar claramente el o los objetos que van a participar en el diagrama de estado.

Los elementos a utilizar al elaborar un diagrama de estados, son los descritos dentro de la tabla 1.9:

Tabla 1.9: Elementos de un Diagrama de Objetos

Artefacto	Descripción
	**Estado Inicial.-** Éste círculo indica en dónde inicia el diagrama, es decir, el comienzo del ciclo de vida del objeto representado.
	**Estado.-** Representa el estado del objeto en un momento determinado de su ejecución o ciclo de vida.
	**Estado Terminal.-** Éste artefacto representa el final del diagrama de estados, es decir, en donde finaliza el ciclo de vida del o de los objetos.
	**Relación.-** Representan el cambio de un estado a otro, es decir, la forma cómo interactúan entre sí.

5.1.5.2.4 Diagramas de Interacción

Los **Diagramas de Interacción** son una subdivisión de los diagramas de comportamiento, que se encargan de mostrar el flujo de control y de datos entre los elementos del sistema modelado. Estos diagramas se clasifican en:

5.1.5.2.4.1 Diagrama de Secuencia

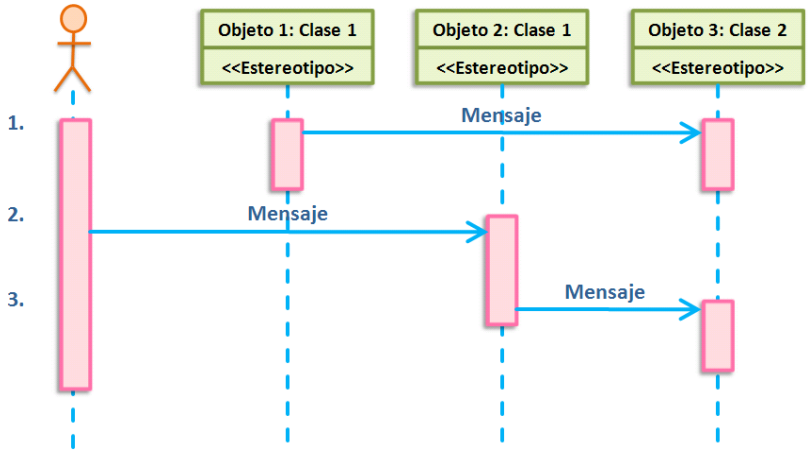


Figura 1.11: Diagrama de Secuencia

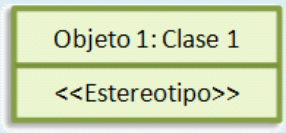
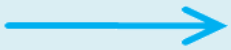
Como puede apreciarse en la Figura 1.11, este tipo de diagrama representa gráficamente la secuencia ordenada y lógica que se da entre los objetos de un caso de uso dentro de un escenario específico.

Los diagramas de secuencia constituyen el mayor trabajo que debe realizarse durante la etapa de diseño, puesto que marcan el comportamiento que tendrá el sistema en tiempo de ejecución. Para lograrlo, se deberá incluir con gran detalle, el cómo los objetos van a realizar todos y cada uno de los procesos.

Un diagrama de secuencia debe contener los cursos normales y cursos alternos de eventos o casos de uso, por lo tanto, existirá un diagrama de secuencia por cada caso de uso identificado.

Los elementos identificados en éste diagrama UML son los que se presentan en la tabla 1.10:

Tabla 1.10: Elementos de un Diagrama de Secuencia

Artefacto	Descripción
	**Actor.-** Representa a la entidad que interactúa con el caso de uso, y que generalmente, es quien inicia una acción.
	**Objeto.-** Representa a las instancias que intervienen en el caso de uso que se está representando. En el casillero superior se coloca el nombre del objeto (no obligatorio) y de la clase a partir de la cual fue instanciado (obligatorio), en un formato “nombre objeto: nombre clase”. En el casillero inferior se indica el estereotipo del objeto: <<boundary>>, <<entity>> o <<control>>.
	**Mensaje.-** Indican la comunicación que existe entre dos objetos. Pueden ser colocados entre las líneas de vida de los objetos, o entre los métodos. Sobre éste artefacto se deberá colocar la acción que será realizada, los valores necesarios para ello, y los valores de retorno que se obtendrán al finalizar dicha tarea.
	**Método.-** Representa una acción a ser ejecutada por el objeto al cual pertenece. Se encuentra unido al objeto, a través de una línea entrecortada (línea de vida).

5.1.5.2.4.2 Diagrama de Comunicación

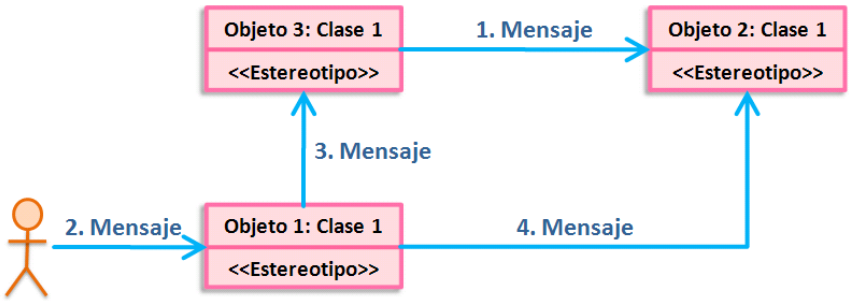


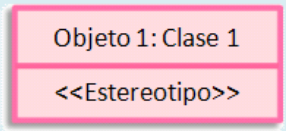
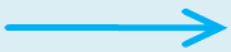
Figura 1.12: Diagrama de Comunicación

Antiguamente conocidos como diagramas de colaboración, son los encargados de explicar la distribución que los objetos van a tener en el sistema (ver Figura 1.12).

Guardan una estrecha relación con los diagramas de secuencia, puesto que los dos muestran cómo interactúan los objetos entre sí, a través del intercambio de mensajes. Sin embargo, la diferencia radica en que el diagrama de secuencia hace énfasis en seguir ordenadamente la narrativa del flujo de un caso de uso en el tiempo, mientras que un diagrama de comunicación hace énfasis en la distribución de los objetos, mostrando cómo colaboran en un caso de uso.

Un diagrama de comunicación está conformado por los elementos de la tabla 1.11:

Tabla 1.11: Elementos de un Diagrama de Comunicación

Artefacto	Descripción
	**Actor.-** Representa a quien generalmente inicia la acción representada en el diagrama.
	**Objeto.-** Representa a las instancias que intervienen en el caso de uso representando. En el casillero superior se coloca el nombre del objeto (no obligatorio) y el de la clase a partir de la cual fue instanciado (obligatorio), en el formato "nombre objeto: nombre clase". En el casillero inferior se deberá indicar el estereotipo del objeto: <<boundary>>, <<entity>> o <<control>>.
	**Mensaje.-** Indica la comunicación que existe entre los objetos. Sobre ésta línea se deberá colocar la acción que será

	realizada (con los valores necesarios de entrada y los valores de retorno al finalizar dicha tarea), acompañado de un número que indica el orden dentro de la secuencia.

5.1.5.2.4.3 Diagrama de Tiempos

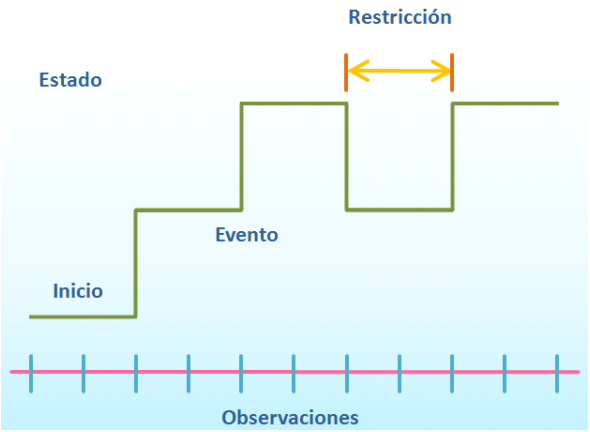


Figura 1.13: Diagrama de Tiempos

Este diagrama mostrado en la Figura 1.13, se constituye en un nuevo tipo incorporado en la versión 2.0 de UML. Se encarga de mostrar los cambios en el estado de un objeto (a lo largo del tiempo y de forma lineal) en respuesta a los eventos ocurridos.

En él se pueden apreciar detalladamente las restricciones que tienen los objetos en el tiempo, por lo que son utilizados con mayor frecuencia en sistemas de tiempo real.

Los artefactos utilizados en éste diagrama son detallados en la tabla 1.12.

Tabla 1.12: Elementos de un Diagrama de Tiempos

Artefacto	Descripción
	**Línea de vida.-** Representa el ciclo de vida del objeto representado. Las alteraciones que pueda presentar, serán el resultado de los cambios del objeto, ante los diferentes eventos producidos. El estado del objeto puede ser colocado en la parte superior izquierda del diagrama, mientras que los eventos y restricciones, deberán ser colocados junto a la línea de vida tomando en consideración el momento en el que se produce.

	**Línea de Tiempo.-** Representa el tiempo de vida de un objeto, y estará dividida en varias unidades, de tal manera que se pueda ejercer un mayor control sobre los cambios sufridos por el objeto. En la parte inferior de éste artefacto se podrán colocar las observaciones necesarias.

5.1.5.2.4.4 Diagrama de Vista de Interacción

Éste diagrama muestra cierta vista de los aspectos dinámicos del sistema que se debe modelar. Cuando se tenga entre manos sistemas complejos, lo más aconsejable es subdividir en partes más pequeñas las interacciones, de tal manera que se pueda realizar un mejor trabajo en el diseño del sistema.

CAPÍTULO II

Lenguaje de Programación C Sharp

5.2.1 Definiciones

C# o C Sharp es un lenguaje de programación diseñado específicamente para utilizarse en la plataforma .NET, motivo por el cual programar utilizando C# es mucho más fácil, que hacerlo con cualquiera de los otros lenguajes que ésta plataforma soporta.

Este lenguaje fue desarrollado por Microsoft (principalmente por Scott Wiltamuth y Anders Hejlsberg), tomando como base las mejores características de C++, Java y Visual Basic, con la idea no sólo de crear un lenguaje orientado a objetos, sino también, con el objetivo de contar con el primer lenguaje orientado a componentes.

5.2.2 Características

- No soporta herencia múltiple: únicamente se maneja el runtime para la herencia múltiple en la forma de interfaces.
- No maneja apuntadores: usa los llamados delegados (delegates) para proveer del modelo de eventos.
- Gestión automática de memoria, puesto que proporciona un colector de basura para la administración de memoria en los programas C#.
- Es completamente orientado a objetos (encapsulamiento, herencia y polimorfismo).
- A los modificadores private, public y protected, se incrementa el internal.
- Es un lenguaje orientado a componentes.
- Posee un sistema de tipos unificado, puesto que todos los tipos de datos se derivan de una clase base común llamada System.Object.
- Se puede insertar en código fuente C#, fragmentos de código escrito en cualquiera de los lenguajes soportados.

5.2.3 Tipos de Datos

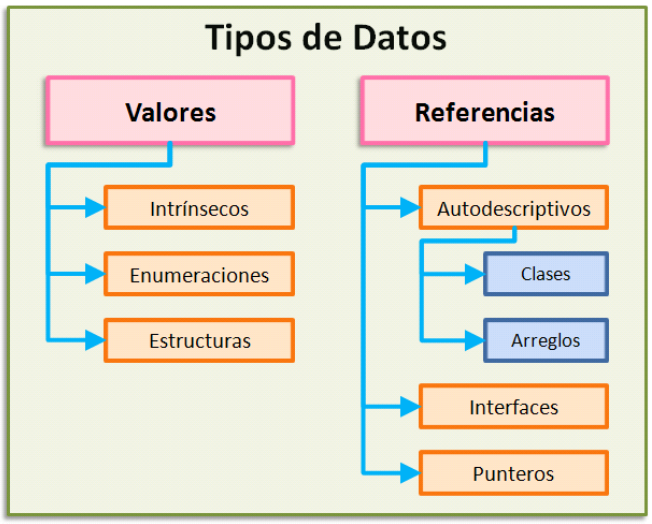


Figura 2.1: Tipos de Datos[33]

C# cuenta con un sistema de tipos unificado que establece que todos los tipos de datos parten de una raíz común denominada System.Object (ver Figura 2.2), misma que se subdivide en dos grandes grupos (ver Figura 2.1): por una parte los que almacenan valores (estáticos), y por otro, los tipos que almacenan referencias (dinámicos).

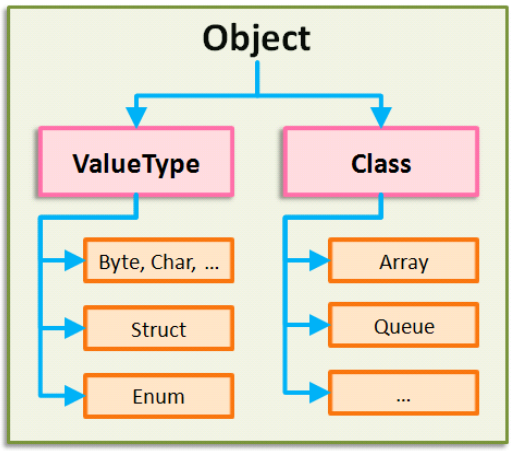


Figura 2.2: Sistema de tipos unificado[34]

[33] CHARTE OJEDA, Francisco. 2003. Programación C# .NET. España. ANAYA. 104 p.
[34] CHARTE OJEDA, Francisco. 2003. Programación C# .NET. España. ANAYA. 105 p.

5.2.3.1 Tipos de Datos Nativos en C#

Como Tipos Intrínsecos se conoce a los tipos de datos fundamentales para cualquier lenguaje: números, cadenas de caracteres, lógicos, etc.

El Sistema Común de Tipos (CTS) define como tipos de datos para C#, a los que se muestran a continuación, en la tabla 2.1:

Tabla 2.1: Tipos de Datos Intrínsecos[35]

Tipo	Dato que puede contener
byte	Número entero de 8 bits sin signo.
sbyte	Número entero de 8 bits con signo.
short	Número entero de 16 bits sin signo.
ushort	Número entero de 16 bits con signo.
int	Número entero de 32 bits sin signo.
uint	Número entero de 32 bits con signo.
long	Número entero de 64 bits sin signo.
ulong	Número entero de 64 bits con signo.
float	Número con parte decimal de precisión simple.
double	Número con parte decimal de precisión doble.
decimal	Número con parte decimal fija para cálculos financieros.
bool	Valor de verdad verdadero (true) o falso (false).
char	Un carácter del conjunto Unicode.
string	Cadena de caracteres.
object	Referencia a cualquier objeto.

5.2.4 Declaración de Variables

En el lenguaje de programación C# se define a una variable como un identificador utilizado para almacenar datos generados durante el proceso de ejecución de un programa.

La declaración de una variable (en cuanto a su nombre) debe seguir ciertas normas establecidas:

[35] CHARTE OJEDA, Francisco. 2003. Programación C# .NET. España. ANAYA. 107 – 108 pp.

- Su nombre debe ser claro y referente al problema.
- No deben contener espacios en blanco ni símbolos extraños entre ellas.
- No deben ser palabras de C#.

En lo referente a su visibilidad se deberá tomar en consideración los siguientes modificadores:

- Public, si se desea que la variable sea visible en todo el namespace.
- Private, en cuyo caso la variable será visible únicamente en la clase dentro de la que fuera declarada.
- Protected, cuando la variable deba ser accedida desde una clase determinada, y desde todas las clases derivadas a partir de ésta.
- Internal, en cuyo caso la variable podrá ser accedida po. el código que es parte del mismo componente (ensamblado), ya sea una aplicación o una biblioteca.
- Internal protected, que permite el acceso ya sea de forma internal o protected.

Ahora bien, la sintaxis para la declaración de una variable es la mostrada en la figura 2.3:

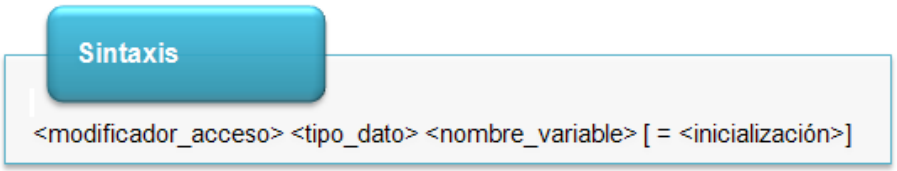


Figura 2.3: Sintaxis de la declaración de una variable

5.2.5 Programación Orientada a Objetos en C#

5.2.5.1 Ámbitos con nombre

Un espacio o ámbito con nombre (conocido como namespace) “es un ámbito delimitado explícitamente al que se ha asignado un identificador”[36]. Dentro de éstos, es posible incluir definiciones de tipos de datos como también nuevos ámbitos con nombre, dando paso a una jerarquía de namespaces.

[36] CHARTE OJEDA, Francisco. 2003. Programación C# .NET. España. ANAYA. 174 p.

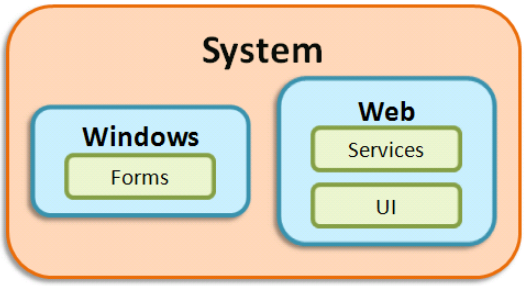


Figura 2.4: Anidamiento de diferentes Ámbitos con nombre[37]

Como se puede observar en la Figura 2.4, el ámbito SYSTEM contiene en su interior a dos namespaces: WINDOWS y WEB. Por su parte el ámbito WEB contiene a su vez, a los ámbitos SERVICES y UI. De ésta manera se puede comprobar es posible que se presente un anidamiento con todos los ámbitos que formen parte de un proyecto de software determinado.

La sintaxis de un namespace o ámbito con nombre es la mostrada en la Figura 2.5:

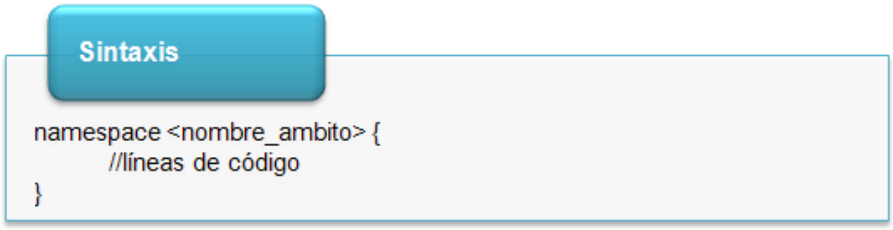


Figura 2.5: Sintaxis de declaración de un Ámbito con Nombre (Namespace)

5.2.5.2 Clases

Una clase es una plantilla en la que se describen las características y el comportamiento de todos los objetos que se creen a partir de ésta. Las clases son utilizadas para modelar las aplicaciones como si estuviesen conformadas por un conjunto de objetos.

En C# las clases representan un novedoso mecanismo para crear nuevos tipos de datos, a partir de las que se crearían los objetos para acceder a sus miembros. Lo que las diferencia de cualquier otro tipo de dato, es que las clases pueden derivarse unas de otras.

[37] CHARTE OJEDA, Francisco. 2003. Programación C# .NET. España. ANAYA. 174 p.

La palabra clave que las representa es **class**, misma que deberá ser precedida por un identificador; las instrucciones que la clase incluya serán delimitadas por llaves ({ }) de inicio y de cierre. Adicionalmente, se deberá tomar en consideración que cualquier clase deberá ser declarada dentro de un ámbito con nombre (namespace).

En lo referente a su visibilidad tenemos los siguientes modificadores:

- Public, si se desea que la clase sea visible en todos los ámbitos.
- Private, que es el ámbito más reducido de los existentes. En éste caso, la clase podrá ser utilizada únicamente en el interior del ámbito en donde haya sido declarada.
- Internal, en cuyo caso la clase será visible desde fuera de su ámbito inmediato, siempre y cuando pertenezca al mismo ensamblado.
- Abstract, en cuyo caso una clase no podrá ser instanciada, es decir, no se podrán crear objetos a partir de ella.
- Sealed, si se desea que la clase no pueda ser heredada.

Es importante indicar que las clases cuentan con constructores y destructores propios, los cuales son métodos específicos para la construcción y destrucción de los objetos instanciados a partir de una clase determinada.

Una misma clase puede contar con varios constructores siempre y cuando posean diferentes listas de parámetros, pero destructores, uno sólo. Finalmente en la Figura 2.6 se hace la descripción de la sintaxis utilizada para la declaración de una clase:

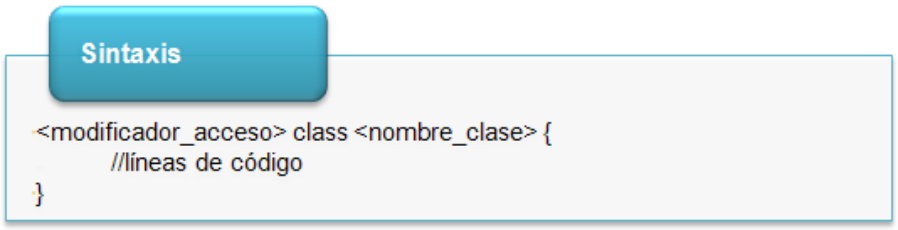


Figura 2.6: Sintaxis de declaración de una Clase

5.2.5.3 Miembros de una Clase

Los miembros que forman parte de una clase son los mostrados en la tabla 2.2:

Tabla 2.2: Miembros de una Clase

Miembro	Descripción
Variables	Identificador utilizado para almacenar datos generados durante la ejecución de un programa.
Métodos	Describen el comportamiento y la funcionalidad que va a tener una clase u objeto. Son parte del tipo (clase), pero no son parte del objeto.
Propiedades	Permiten personalizar a los objetos o bien, obtener información a partir de ellos.
Eventos	Son señales que permiten a los objetos interactuar con la aplicación en la que son utilizados. Una clase puede usar un evento para notificar a otra que algo ocurrió
Miembros compartidos (variables o métodos)	Están asociados con la clase, no con el objeto. Son compartidos por todos los objetos de la clase y se diferencian por contar con el modificador de visibilidad static. Son conocidos como miembros estáticos.
Miembros sobrecargados (métodos)	Permite que en una misma clase existan varios métodos con el mismo nombre, siempre y cuando la lista de parámetros varíe.
Miembros redefinidos	Facilitan una nueva implementación del miembro.

Independientemente del miembro de clase al que se desee hacer referencia, éstos pueden utilizar los modificadores de visibilidad private, protected, public, internal y protected internal (ver sección 2.4, Declaración de Variables).

5.2.5.4 Interfaz

Una interfaz es una clase en la que se enumeran los métodos que la componen, indicando tipos de retornos y listas de parámetros, y cuyos miembros no podrán ser implementados. Se la puede considerar como una especificación a la que podrán ajustarse las clases que así lo requieran.

La sintaxis utilizada es la que se muestra a continuación en la figura 2.7:

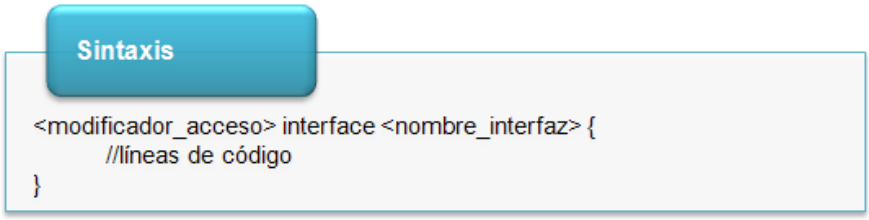


Figura 2.7: Sintaxis para la declaración de una interfaz

5.2.6 Colecciones de Datos

El lenguaje de Programación C# proporciona a los desarrolladores de software una gama variada de listas especializadas o colecciones de datos, las mismas serán utilizadas de acuerdo al tipo de información con el que se esté trabajando.

En la tabla 2.3 mostramos un listado de las Colecciones de Datos más importantes:

Tabla 2.3: Colecciones de Datos

Colección	Descripción
ArrayList	Almacena una lista de objetos.
SortedList	Almacena una lista de objetos, pero de forma ordenada.
Queue	Almacena una lista de objetos, pero con la restricción de que el primer objeto en entrar es el primero en salir (Tratamiento similar al de una cola).
Stack	Similar a la anterior, con la diferencia de que el último objeto en entrar será el primero en salir.
HashSet	Almacena una lista de objetos, con la restricción de que no se permiten objetos duplicados.
SortedList	Almacena una colección de pares de valores, una clave y un valor, de manera ordenada.
SortedDictionary	Almacena una colección de pares (clave, valor) que se ordenan en la clave.
ArraySegment	Delimita una sección de una matriz unidimensional.
LinkedList	Representa una lista doblemente vinculada.

5.2.6.1 Arreglos Unidimensionales

Como su nombre lo indica, se trata de arreglos de una sola dimensión, en donde el número de elementos que contendrá será dado al momento de su creación. Es por ello que pueden ser representadas gráficamente como una lista de valores.

Para declarar un arreglo se deberá especificar el ámbito, identificador, tipo, símbolo ([ ]) y nombre, tal y como se muestra en la figura 2.8:

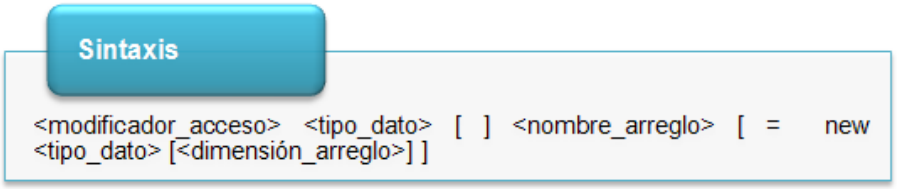


Figura 2.8: Sintaxis para la declaración de un Arreglo Unidimensional

5.2.6.2 Arreglos Multidimensionales

Son arreglos que tienen varias dimensiones (60 como máximo), cada una de las cuales será manejada con un rango diferente. Cada uno de éstos rangos podrá contar con tantos elementos como valores posibles hay en el tipo long. En la figura 2.9 se muestra la sintaxis utilizada para la declaración de un arreglo multidimensional:

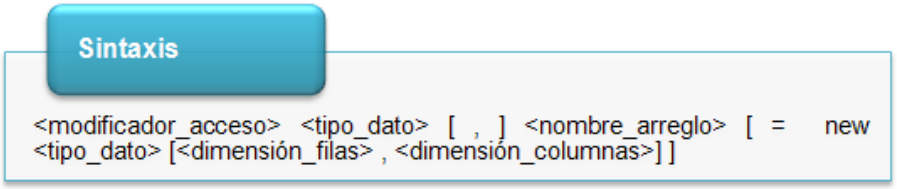


Tabla 2.9: Sintaxis para la declaración de un Arreglo Multidimensional

En éste caso se trata de un arreglo bidimensional, pero de acuerdo a las necesidades del programador, el número de dimensiones variará. Como se puede observar, el número total de elementos es 10 (5 elementos en la primera dimensión, por 2 elementos de la segunda dimensión).

5.2.6.3 Colecciones Genéricas

Las colecciones genéricas nos permiten definir el tipo de datos que se desea almacenar, cuando éste no sea un objeto.

En la tabla 2.4 se muestra un listado de las mismas:

Tabla 2.4: Colecciones Genéricas

Colección	Colección Genérica
ArrayList	List.(Representa una lista de objetos con establecimiento inflexible de tipos a la que se puede tener acceso a través de un índice. Proporciona métodos para buscar, ordenar y manipular listas)
SortedList	SortedList
Queue	Queue
Stack	Stack
HashSet	HashSet
SortedDictionary	Dictionary (Representa una colección de claves y valores)
ArraySegment	ArraySegment
LinkedList	LinkedList

CAPÍTULO III

Herramientas CASE

5.3.1 Definiciones

Las herramientas CASE (Ingeniería de Software Asistida por Computador, por sus siglas en inglés: Computer Aided Software Engineering), se definen como un conjunto de programas, métodos, utilidades y técnicas que apoyan a los analistas, ingenieros en sistemas y desarrolladores en general, durante todo el proceso de desarrollo de software.

Estas herramientas nos permiten automatizar dicho proceso, durante todos los pasos del ciclo de vida de software, ya sea completamente, o tan sólo en algunas de sus fases, con el único objetivo de aumentar la productividad y reducir el coste de desarrollo tanto en tiempo como en dinero.

5.3.2 Objetivos

- Optimizar la productividad, efectividad y eficiencia en el desarrollo de software.
- Reducir el coste asociado con el desarrollo de software, a través de la disminución en la inversión de recursos, principalmente tiempo y dinero.
- Mejorar la planificación de recursos que se deberán invertir en un proyecto.
- Automatizar el proceso de desarrollo y de mantenimiento del software.
- Ayudar en la reutilización, portabilidad y documentación de proyecto de software.
- Gestión global en todas las fases de desarrollo de software con una misma herramienta.

5.3.3 Componentes

5.3.3.1 Repositorio (Directorio)

Almacena los elementos definidos o creados por la herramienta CASE, y su gestión se realiza gracias a un Sistema de Gestión de Base de Datos (SGBD) o a un sistema de gestión de ficheros.

5.3.3.2 Meta Modelo

Marco para la definición de técnicas y metodologías que serán soportadas por la herramienta.

5.3.3.3 Carga o Descarga de Datos

Permiten cargar el repertorio de la herramienta con los datos que provienen desde otros sistemas, o bien generar información que podrá ser utilizada en ellos.

5.3.3.4 Comprobación de Errores

Es una de las capacidades más importantes de las herramientas CASE, puesto que analiza la exactitud, integridad y consistencia de los esquemas generados por dicha herramienta. Es así que se incrementa tanto la productividad como la calidad del producto.

Existen cinco tipos básicos de comprobaciones que una herramienta CASE debe ejercer sobre los diferentes diagramas:

- Comprobación de sintaxis y tipo
- Comprobación de integridad y consistencia
- Comprobación de descomposición funcional
- Comprobación cruzada de consistencia a través de todos los niveles y vistas
- Comprobación de rastreo de requerimientos

5.3.3.5 Módulo de Diagramación y Modelación

Está constituido por editores de texto y herramientas de diseño gráfico que facilitan el trabajo realizado por los analistas o desarrolladores en general. Permite a los usuarios elaborar los diferentes diagramas para el modelado de sistemas con la aplicación de un modelo determinado, ya sea de manera automática o semiautomática.

Gracias a éste, cuando se realiza cualquier cambio en el sistema, se modifican los diagramas, mas no las aplicaciones: el código se genera nuevamente, y la documentación e información estará siempre actualizada. Más allá de la automatización del proceso de creación de diagramas debemos tomar en consideración tres aspectos fundamentales:

- Capturando el significado de un objeto dibujado en la pantalla del ordenador, podemos comprobar su corrección y complejidad.

- Almacenando los objetos de un diagrama determinado, podemos compartir objetos entre diferentes diagramas.

- Recogiendo el significado de una eficaz corrección de errores realizada de manera automática sobre los diagramas dibujados, podemos generar código automáticamente.

5.3.3.6 Generador de Código

Es el encargado de transformar el diseño realizado, en código fuente de la aplicación que se va a desarrollar.

5.3.3.7 Generador de Documentación

Se alimenta del repositorio de la herramienta para transcribir las especificaciones contenidas en él.

5.3.4 Clasificación

No existe una clasificación estándar de herramientas CASE, por lo que varios autores utilizan múltiples criterios para hacerlo. De acuerdo a su funcionalidad las herramientas pueden ser clasificadas tal como se muestra en la Tabla 3.1:

Tabla 3.1: Clasificación de las Herramientas CASE según su funcionalidad

Nombre	Descripción
Herramientas integradas I – CASE	Integrated CASE o CASE integrado, abarcan todo el ciclo de vida del desarrollo de software. También son conocidas como CASE workbench.
Herramientas de alto nivel U – CASE	Upper CASE o CASE superior, son herramientas orientadas a automatizar y dar soporte a las actividades que se desarrollan durante el análisis y el diseño.
Herramientas de bajo nivel L – CASE	Lower CASE o CASE inferior, son herramientas que están orientadas a automatizar las fases de construcción e implantación.

5.3.5 Herramientas CASE para el diseño de la arquitectura de un producto software

Según el tipo de licencia que posean, estas herramientas pueden dividirse en dos grandes grupos: por una parte las privativas o de uso comercial, y por otra parte, las no privativas o de uso libre. A continuación presentamos una descripción de carácter general:

Tabla 3.2: Herramientas CASE para el diseño de la arquitectura de un producto software

Uso Libre	
Logotipo	**Descripción**
BOUML	**BOUML.-** Permite definir y generar código en C++, Java, IDL y PHP y es capaz de generar documentación en varios formatos, como por ejemplo HTML, XMI, etc. Su principal ventaja, es que a pesar de estar trabajando con varios diagramas a la vez, consume muy poca memoria. Además es multiplataforma, puesto que es compatible con los Sistemas Operativos Windows / Unix / Linux / Solaris / MacOS X.

	**StarUML.-** Permite modelar diagramas UML y traducir ese modelo a código fuente como C++, Java y C# por ejemplo. Se puede comenzar a dibujar los gráficos manualmente o seleccionar las plantillas que contiene el archivo de instalación para modificarlas. Puede ser utilizado en los Sistemas Operativos Windows 98 SE/ Me / 2000 / NT / XP.
	**Umbrello UML Modeller.-** Herramienta para el desarrollo de diagramas UML, que también permite la generación de código fuente para los lenguajes ActionScript, Ada, C++, Corba IDL, Java, JavaScript, PHP, Perl, Python, SQL y XMLSchema. Su licencia es GPL (no privativa), y funciona sobre los Sistemas Operativos Windows / Linux / BSD / UNIX.
	**Dia.-** Es una aplicación de propósito general, puesto que además de permitir el diseño de diagramas UML, Dia permite el diseño de circuitos electrónicos a nivel de puertas y dispositivos, diseño de redes de computadores, etc. Funciona en sistemas operativos Windows 98 / NT / 2000 / ME / XP / Linux.
	**ArgoUML.-** Herramienta para el modelado de software que puede ser utilizado sobre cualquier plataforma o sistema operativo. ArgoUML ofrece a los usuarios la posibilidad de generar código automáticamente, en base a los diagramas realizados previamente, en los lenguajes de programación Java, C++, C# y PHP. Sin embargo, carece de soporte completo para el desarrollo de ciertos tipos de diagramas (Secuencia y Colaboración). Al igual que la anterior, ArgoUML funciona en sistemas operativos Windows 98 / NT / 2000 / ME / XP / Linux.
	**EclipseUML.-** Es una herramienta multiplataforma para el modelado visual de objetos orientada al lenguaje Java que se integra directamente en el entorno de desarrollo Eclipse. Genera código Java a partir de los diagramas UML. Permite realizar no sólo los diagramas UML de una aplicación, sino que también permite modelar y gestionar bases de datos visualmente. La versión 2.0 soporta el reciente UML 2.0 así como Eclipse 3.0.1.
	**UMLet.-** Herramienta Open Source en Java que permite dibujar de forma sencilla y rápida los diferentes diagramas UML (clases, paquetes, estados, casos de uso, actividad, secuencia y colaboración). Puede ser ejecutado desde un archivo jar, o como

	un pluggin de Eclipse.
	**NetBeans.-** Éste IDE dispone de un pluggin para el modelado de sistemas mediante diagramas de clases, actividades, colaboración, componentes, despliegue, paquetes, secuencia, estado y casos de uso. A partir de dichos diagramas genera automáticamente el código fuente en el lenguaje de programación Java. Funciona en sistemas operativos Windows / Linux.
ORACLE JDEVELOPER	**JDeveloper.-** Ésta IDE incluye herramientas de modelado UML, herramientas para la creación y orquestación de servicios web y para la gestión de flujos de negocio. Además incluye herramientas destinadas al desarrollo visual de interfaces de usuario. A pesar de que en sus inicios era de carácter comercial, actualmente se distribuye de manera gratuita.
Uso Comercial	
Logotipo	**Descripción**
TogetherSoft	**Together.-** Esta totalmente adaptada a Java y su característica más importante es la generación de código sincronizada con los diagramas. Soporta los lenguajes Java, C++ e IDL y sus próximas versiones incluirán Visual Basic y ASP.
Rational Rose 98 The world's leading visual modeling tool.	**Rational.-** Soporta la generación de código para Ada, Ansi C++, C++, Java, Visual Basic y CORBA. Puede trabajar en diseños de bases de datos con capacidad de soportar la integración de los datos y los requerimientos de aplicación a través de diseños lógicos y físicos.
ENTERPRISE ARCHITECT	**Enterprise Architect.-** Esta herramienta proporciona el entorno adecuado para el desarrollo de diagramas UML para plataformas Windows. Permite la generación de código a los lenguajes ActionScript 2.0, Java, C#, C++, VB.NET, Delphi, Visual Basic, Python y PHP. Funciona sobre los Sistemas Operativos Win98 / WinME/ Windows2000 / WinXP / Windows2003.

6. Metodología

Es preciso indicar que para el desarrollo de la presente investigación haremos uso de las diferentes técnicas, métodos, herramientas y procedimientos que la Investigación Científica y el Desarrollo de Software ponen a nuestra disposición, con la finalidad de aplicar, sistematizar y descubrir nuevos conocimientos que nos conduzcan a dar una alternativa de solución a la problemática planteada.

Por su parte, la Matriz de Consistencia General fue elaborada con la finalidad de contrastar la consistencia entre la problemática, tema, problema, objetivo y la hipótesis general de investigación.

6.1 Matriz de Consistencia General

Problema General de Investigación.- Falta de una Herramienta para el análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp.			
Tema	**Objeto de Investigación**	**Objetivo General**	**Hipótesis**
Desarrollo de una Herramienta de análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp (ECMA – 334)	▪ Herramientas para el análisis y diseño de sistemas ▪ Lenguaje Unificado de Modelado UML 2.0 ▪ Especificación del Lenguaje de Programación C Sharp	Desarrollar una herramienta de análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp (ECMA – 334)	El desarrollo de una herramienta de análisis y diseño de sistemas que permita la documentación de software basada en UML 2.0 y la generación de código fuente bajo la Especificación del Lenguaje de Programación C Sharp (ECMA – 334), permitirá que los desarrolladores realicen un esquema adecuado de la arquitectura de su proyecto, minimizando la aparición de riesgos asociados al mal diseño.

6.2 Materiales, Métodos y Técnicas de Trabajo

6.2.1 Materiales

- Fuentes bibliográficas e internet para recopilar información acerca del Marco Teórico de la investigación, así como también para desarrollar el producto software planteado.
- Hardware y software que apoyarán a la elaboración tanto de la documentación, como de la aplicación a desarrollar:
  - Un ordenador Intel Pentium 4, CPU de 2.00GHz, 512 de RAM.
  - Un ordenador Intel Pentium 4, CPU de 3.00GHz, 512 de RAM.
  - Una impresora Canon iP1000.
  - Sistema Operativo: Windows XP Professional Versión 2002 compilación 6200, Ubuntu Utimate Edition 8.1.
  - Herramienta para la Planificación: Microsoft Office Project 2003.
  - Herramienta para la Documentación: OpenOffice 3.0.
  - Lenguaje de Programación: Java, C#.
  - Herramienta para codificación: NetBeans 6.7.1.
  - Herramienta para diseño de arquitectura: Enterprise Architect 7.1, NetBeans 6.7.1.
  - Herramienta para diseño gráfico: Gimp.

6.2.2 Métodos

6.2.2.1 Metodología para el Desarrollo de la Investigación

Según Ander Egg, la **Investigación Científica** es un procedimiento reflexivo, sistemático, controlado y crítico, que permite descubrir nuevos hechos o datos, relaciones o leyes, en cualquier campo del conocimiento humano. Ésta se apoya en el **Método Científico** para plantear los problemas científicos que hayan sido detectados en la sociedad y para poner a prueba las hipótesis científicas que hayan sido planteadas en base a esos problemas.

Para el desarrollo de nuestro proyecto investigativo hemos creído conveniente hacer uso fundamentalmente de los Métodos Inductivo y Deductivo, ya que por una parte el **Método Inductivo** (de lo particular a lo general), nos permitirá analizar casos

particulares que se presentan en cuanto al uso de Herramientas CASE para la generación de código a partir del diseño de la arquitectura del software, a partir de los cuales se extraerán conclusiones de carácter general para el planteamiento de las hipótesis. Asimismo, en base a ésta generalización, se podrán determinar las necesidades que deberán ser cubiertas por la herramienta a desarrollar.

De su lado, el **Método Deductivo** (de lo general a lo particular) servirá para que en base a conceptos o principios generales, se extraiga la causa del problema planteado y se determinen las consecuencias producidas. De ésta manera se podrán definir los pasos lógicos a seguir con la finalidad de proporcionar a la sociedad una alternativa de solución al problema planteado.

6.2.2.2 Metodología para el Desarrollo de la Herramienta

Para el diseño y desarrollo de la herramienta para la carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja, que permita la generación de código fuente para el Lenguaje de Programación C# a partir del diseño de diagramas estructurales (clases, paquetes, componentes y despliegue) y de comportamiento (casos de uso, robustez y secuencia) basados en UML 2.0, hemos creído conveniente emplear la **Metodología Ágil para el Desarrollo de Software RUP** (Rational Unified Process – Proceso Unificado de Modelado), debido a la simplicidad de sus reglas y prácticas, a su orientación a equipos de desarrollo de tamaño pequeño, a su flexibilidad ante los cambios, a su ideología de colaboración y a que define un ciclo de vida iterativo que prioriza el uso de lenguajes de modelado, casos de uso y centrado en la arquitectura.

El proceso de ciclo de vida de RUP se divide en cuatro fases bien definidas, que a su vez son subdivididas en iteraciones, cada una de las cuales produce una pieza de software demostrable:

- **Incepción**

  Dentro de ésta fase se establecerán claramente cuáles serán los requerimientos de la aplicación, su alcance, sus condiciones de límite y los criterios de aceptabilidad. De igual manera se identificarán los casos de uso que orientarán la funcionalidad de la herramienta, para en base a ello, proceder al diseño de las arquitecturas preliminares, a la estimación del cronograma de trabajo y del presupuesto de todo el proyecto.

- **Elaboración**

  Se analizará el dominio de la problemática planteada para definir el plan concreto del proyecto, es decir, para especificar en detalle las actividades a desarrollar por cada uno de los responsables. Posteriormente se realizará la identificación de los casos de uso y la descripción de la arquitectura de la herramienta, con la finalidad de elaborar un prototipo base a partir de ésta descripción y de realizar un análisis que permita determinar los riesgos presentados en el software, en la planificación y en el presupuesto.

- **Construcción**

  Se procederá a desarrollar, integrar y verificar los diferentes módulos que formarán parte de la herramienta progresivamente, es decir, la herramienta será desarrollada en diferentes versiones preliminares, de tal manera que en cada una de ellas se realice una depuración de los errores encontrados, y poco a poco se vayan incorporando cada una de las prestaciones establecidas inicialmente.

- **Transición**

  Finalmente, dentro de ésta etapa se realizará una depuración completa del producto final, es decir, se harán las últimas correcciones y se agregarán los detalles finales a la aplicación. De igual manera se producirá la documentación necesaria tanto para desarrolladores como para los usuarios, de tal manera que se pueda explotar a la herramienta de la manera más óptima.

Es importante destacar que a través de las fases antes mencionadas, se desarrollan en paralelo siete actividades: Modelado de Negocios, Requerimientos, Análisis, Diseño, Codificación, Prueba y Gestión de Configuración del software.

6.2.3 Técnicas de Trabajo

La realización del presente proyecto presupone la práctica de ciertas técnicas que contribuyan al proceso investigativo en la recolección de datos y que sirvan como instrumentos de medición y de prueba:

6.2.3.1 Observación Directa

Ésta técnica nos permitió conocer en parte la realidad de la Carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja, para determinar los problemas existentes en la misma, y seleccionar el que a nuestro criterio, será objeto de nuestro proyecto de investigación. Así mismo, nos permitió vincularnos con el problema (objeto de nuestra investigación) y con sus fuentes de información teóricas y empíricas.

6.2.3.2 Encuestas

Ésta técnica será dirigida a los desarrolladores de software de la Carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja que hacen uso de C#, con la finalidad de obtener información relacionada con el problema planteado. En base a ésta información, se realizará la recolección de material bibliográfico, la organización del material recogido, el procesamiento de los datos, su análisis y documentación.

7. Cronograma

8. Presupuesto y Financiamiento

8.1 Recursos y Costos

8.1.1 Recursos Humanos

Autores:

- Danny Emanuel Muñoz Flores
- Alexandra Elizabeth Maurad Córdova

Asesores:

- Dra. Nora Tene
- Ing. Patricio Analuisa
- Ing. Milton Labanda

8.1.2 Recursos Materiales

Recursos Materiales			
Material	**Cantidad**	**Unidad**	**Valor Total**
Anillado	10	$ 2,00	$ 20,00
Carpetas de perfil	3	$ 0,60	$ 1,80
Empastado	3	$ 10,00	$ 30,00
Material de oficina	-	$ 50,00	$ 50,00
Resma de papel Inen A4	10	$ 4,00	$ 40,00
Recarga de tinta a color para impresora (genérica)	4	$ 12,50	$ 50,00
Recarga de tinta negra para impresora (genérica)	8	$ 6,00	$ 48,00
Infocus	2	$ 15,00	$ 30,00
SUBTOTAL			$ 269,80

8.1.3 Servicios Básicos

Servicios Básicos		
Servicio	**Valor por mes**	**Valor Total**
Luz	$ 20,00	$ 260,00
Teléfono	$ 10,00	$ 130,00
Transporte	$ 25,00	$ 325,00
SUBTOTAL		$ 715,00

8.1.4 Recursos Técnicos y Tecnológicos

Hardware					
Equipo	**Costo**	**Costo Residual**	**Vida Útil (años)**	**Uso (días)**	**Valor**
Computadora de escritorio	$ 1000,00	$ 100,00	10 años	297	$ 74,23
Computadora de escritorio	$ 1000,00	$ 100,00	10 años	297	$ 74,23
Flash Memory	$ 18,00	$ 5,00	2 años	150	$ 2,72
Impresora	$ 60,00	$ 20,00	3 años	30	$ 1,11
SUBTOTAL					$ 152,29

Software	
Aplicación	**Valor Total**
SO: Windows XP Professional Versión 2002 compilación 6200	$ 143,77
SO: Ubuntu Ultimate Edition 8.1	$ 000,00
Herramienta para la Planificación: Microsoft Office Project 2003	$ 120,00
Herramienta para la Documentación: OpenOffice 3.0	$ 000,00
Lenguaje de Programación: Java	$ 000,00
Lenguaje de Programación: C#	$ 000,00
Herramienta para codificación: NetBeans 6.7.1	$ 000,00
Herramienta para diseño de arquitectura: Enterprise Architect 7.1	$ 135,00
Herramienta para diseño gráfico: Gimp	$ 000,00
SUBTOTAL	$ 398,77

Comunicaciones			
Medio	**Cantidad**	**Valor Unitario**	**Valor Total**
Internet	60	$ 0,80	$ 48,00
SUBTOTAL			$ 48,00

Recursos Técnicos y Tecnológicos	
Recurso	**Valor Total**
Hardware	$ 152,29
Software	$ 398,77
Comunicaciones	$ 48,00
SUBTOTAL	$ 599,06

8.1.5 Resumen de Costos

Resumen de Costos	
Recurso	**Valor Total**
Recursos Humanos	$ 000,00
Recursos Materiales	$ 269,80
Servicios Básicos	$ 715,00
Recursos Técnicos y Tecnológicos	$ 599,06
SUBTOTAL	$ 1.583,86
IMPREVISTOS (5 %)	$ 79,19
TOTAL	$ 1.663,05

8.2 Financiamiento

El total de los gastos que se deberán realizar para el desarrollo del presente trabajo investigativo, monto que asciende a MIL SEISCIENTOS SESENTA Y TRES DÓLARES AMERICANOS CON CINCO CENTAVOS ($ 1.663,05), serán financiados en su totalidad por los tesistas, sin riesgo de requerir algún tipo de crédito para el efecto.

9. Bibliografía

9.1 Libros

- CHARTE OJEDA, Francisco. 2003. Programación C#.NET. España. Anaya.

9.2 Sitios web

- SOTO MADRIGAL, Alejandro. 2007. Colecciones de Datos. http://alejandrocr.blogspot.com/2007/06/colecciones-de-datos.html, 03 de febrero del 2009.
- BARROS JUSTO, José Luis. 2006. Herramientas CASE. http://creaweb.ei.uvigo.es/creaweb/Asignaturas/DAS/apuntes/Teoria%20semana%209.pdf, 27 de noviembre del 2008.
- Herramientas CASE. 2008. Wikipedia. http://es.wikipedia.org/wiki/CASE, 27 de noviembre del 2008.
- Lenguaje Unificado de Modelado. 2008. Wikipedia. http://es.wikipedia.org/wiki/Lenguaje_Unificado_de_Modelado, 28 de noviembre del 2008.
- NetBeans UML. 2007. NetBeans org. http://plugins.netbeans.org/PluginPortal/faces/PluginDetailPage.jsp?pluginid=1801, 07 de febrero de 2009.
- Oracle JDeveloper. 2008. Proiektutaldea-ren Weblog. http://proiektutaldea.wordpress.com/2008/04/16/oracle-jdeveloper/, 07 de febrero de 2009.
- HERNÁNDEZ ORALLO, Enrique. 2002. El Lenguaje Unificado de Modelado (UML). http://www.disca.upv.es/enheror/pdf/ActaUML.PDF, 28 de noviembre del 2008.
- Empresa Arquitecto para UML 2.1 (Enterprise Architect for UML 2.1) v6.5. 2006. Free Download Manager. http://www.freedownloadmanager.org/es/downloads/Arquitecto_de_Empresa_para_UML_2.1_3280_p, 04 de febrero de 2009.
- ALARCÓN, José. 2004. Disponible Eclipse UML 2.0. PC World Digital. http://www.idg.es/pcworld/Disponible-Eclipse-UML-2.0/art163162.htm, 04 de febrero de 2009.
- REYNOSO, Carlos. 2006. Métodos Heterodoxos en Desarrollo de Software. Universidad de Buenos Aires, Argentina, MSDN Library.

http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/heterodox.mspx, 30 de octubre del 2008.

- BARRIENTOS ENRÍQUEZ, Aleida Mirian. 2003. Bolivia. El desarrollo de sistemas de información empleando el lenguaje de modelado unificado UML. Monografías.com. http://www.monografias.com/trabajos16/lenguaje-modelado-unificado/lenguaje-modelado-unificado.shtml, 28 de noviembre del 2008.
- Una lista de herramientas CASE. 2006. Navegapolis.net, anual 1 (1). http://www.navegapolis.net/content/view/406/, 27 de noviembre del 2008.
- ArgoUML 0.24. 2008. Osalt.com. http://www.osalt.com/es/argouml, 04 de febrero de 2009.
- StarUML 5.0. 2008. Taringa. http://www.taringa.net/posts/downloads/1677819/StarUML-5_0-Modelador-Diagramas-UML-Excelente!.html, 04 de febrero de 2009.
- NAVARRO, Juan. 2005. UMLet 1.5. http://www.versioncero.com/noticia/19/umlet-15, 07 de febrero de 2009.

10.Anexos

10.1 Matriz de Consistencia Específica

10.1.1 Objetivo Específico 1

Problema Específico.- La falta de conocimiento por parte de los desarrolladores de software, acerca de las herramientas para análisis y diseño de sistemas que soportan C Sharp y UML 2.0.			
Objetivo Específico	**Hipótesis Específica**	**Unidad de Observación**	**Sistema Categorial**
Analizar las herramientas de análisis y diseño de sistemas basadas en la especificación UML 2.0 más utilizadas por los desarrolladores de software	Los desarrolladores de software no tienen un completo conocimiento acerca de las herramientas de análisis y diseño de sistemas que implementen las especificaciones dadas por UML 2.0, incrementando el riesgo de pérdida de tiempo a la hora de seleccionar la más adecuada.	▪ Herramientas para el análisis y diseño de sistemas basadas en UML 2.0	▪ Herramientas CASE ▪ Objetivos ▪ Componentes ▪ Clasificación ▪ Herramientas CASE para el análisis y diseño de la arquitectura de un producto software. ▪ Comparativa de las ventajas y desventajas de las herramientas CASE para el análisis y diseño de la arquitectura de un producto software.

10.1.2 Objetivo Específico 2

Problema Específico.- La falta de una herramienta de análisis y diseño de sistemas que soporte el lenguaje de programación C Sharp y que permita diagramar la arquitectura de un producto software mediante la especificación UML 2.0 para la documentación de diagramas de clases, paquetes, componentes, despliegue, casos de uso y secuencia, causa que los desarrolladores adquieran malos hábitos en el proceso de construcción de software.			
Objetivo Específico	**Hipótesis Específica**	**Unidad de Observación**	**Sistema Categorial**
Desarrollar un módulo de diagramación basado en el Lenguaje de Programación C Sharp, y en la especificación UML 2.0 para modelar diagramas de clases, paquetes, componentes, despliegue, casos de uso y secuencia	El desarrollo de un módulo de diagramación basado en UML 2.0 y en la Especificación del Lenguaje C Sharp (ECMA – 334), permitirá documentar correctamente la arquitectura de un sistema, mediante diagramas de clases, paquetes, componentes, despliegue, casos de uso y secuencia.	▪ Lenguaje Unificado de Modelado UML 2.0	▪ Lenguaje Unificado de Modelado UML. ▪ Historia ▪ Funciones de UML ▪ Estructura de un modelo ▪ Diagramas ✍ Diagramas de Estructura ✍ Diagramas de Comportamiento ✍ Diagramas de Interacción

10.1.3 Objetivo Específico 3

Problema Específico.- La falta de una herramienta que permita la generación de código fuente para el lenguaje de programación C Sharp a partir del diseño de diagramas de clases y secuencia, basados en UML 2.0.			
Objetivo Específico	**Hipótesis Específica**	**Unidad de Observación**	**Sistema Categorial**

Desarrollar un módulo de generación de código fuente para el Lenguaje de Programación C Sharp, a partir de los diagramas de clases y secuencia	El desarrollo de un módulo de generación de código para el Lenguaje de Programación C Sharp a partir de sus diagramas UML 2.0, evitará que los desarrolladores de software deban codificar manualmente la arquitectura de un producto software o bien aplicar malos hábitos como la ingeniería inversa o, codificar para luego corregir.	▪ Especificación del Lenguaje de Programación C Sharp (ECMA – 334)	▪ Lenguaje de Programación C Sharp
			▪ Características
			▪ Tipos de Datos
			  ⬎ Tipos de Datos nativos en C Sharp
			▪ Declaración de variables
			▪ Programación Orientada a Objetos en C Sharp
			  ⬎ Ámbitos con nombre
			  ⬎ Clases
			  ⬎ Miembros de una clase
			  ⬎ Interfaces
			▪ Colecciones de datos
			  ⬎ Arreglos Unidimensionales
			  ⬎ Arreglos Multidimensionales
			  ⬎ Colecciones Genéricas

10.2 Matriz de Operatividad de Objetivos Específicos

10.2.1 Objetivo Específico 1

Objetivo Específico.- Analizar las herramientas de análisis y diseño de sistemas basadas en la especificación UML 2.0 más utilizadas por los desarrolladores de software.						
Actividad o Tarea	**Metodología**	**Fecha**		**Responsables**	**Presu-puesto**	**Resultados Esperados**
		Inicio	**Final**			
Investigar las herramientas de análisis y diseño de sistemas basadas en UML 2.0	▪ Consultas en Internet	27/11/2008	09/02/2009	▪ Alexandra Maurad ▪ Danny Muñoz	$ 40,00	Documentación de las herramientas basadas en UML 2.0
Determinar las herramientas UML más utilizadas por los desarrolladores de software	▪ Consultas en Internet ▪ Encuestas dirigidas a desarrolladores de software ▪ Análisis de la información recolectada	29/04/2009	07/05/2009	▪ Alexandra Maurad ▪ Danny Muñoz	$ 50,00	Listado de las herramientas basadas en UML 2.0 más utilizadas por los desarrolladores de software
Descargar de internet las herramientas más utilizadas por los desarrolladores de software	▪ Internet	08/05/2009	20/05/2009	▪ Danny Muñoz	$ 25,00	Instaladores de las herramientas identificadas
Instalar en el ordenador las	▪ Método Científico	08/05/2009	20/05/2009	▪ Alexandra	$ 12,50	Herramientas instaladas

herramientas descargadas				Maurad		
Realizar pruebas por cada herramienta	▪ Método Deductivo	08/05/2009	25/05/2009	▪ Alexandra Maurad ▪ Danny Muñoz	$ 7,50	Listado de las prestaciones de cada una de las herramientas instaladas
Identificar las fortalezas y debilidades de cada herramienta	▪ Método Inductivo ▪ Método Deductivo	08/05/2009	25/05/2009	▪ Alexandra Maurad ▪ Danny Muñoz	$ 5,00	Documentación de fortalezas y debilidades de cada herramienta
Realizar una comparativa entre las herramientas analizadas	▪ Método Deductivo	26/05/2009	29/05/2009	▪ Alexandra Maurad ▪ Danny Muñoz	$ 5,00	Comparativa de las ventajas y desventajas de las herramientas más utilizadas por los desarrolladores de software

10.2.2 Objetivo Específico 2

Objetivo Específico.- Desarrollar un módulo de diagramación basado en el Lenguaje de Programación C Sharp, y en la especificación UML 2.0 para modelar diagramas de clases, paquetes, componentes, despliegue, casos de uso y secuencia.						
Actividad o Tarea	**Metodología**	**Fecha**		**Responsables**	**Presu-puesto**	**Resultados Esperados**
		Inicio	**Final**			
Incepción (análisis)						
Determinación y filtrado de requerimientos	▪ Observación directa ▪ Método Deductivo ▪ RUP	01/06/2009	03/06/2009	▪ Alexandra Maurad	$ 7,50	Listado de Requerimientos
Identificación y verificación de casos de uso	▪ RUP	04/06/2009	05/06/2009	▪ Alexandra Maurad	$ 5,00	Tabla de Casos de uso validados
Análisis del dominio	▪ RUP	08/06/2009	11/06/2009	▪ Alexandra Maurad	$ 10,00	Documentación del Dominio
Prototipo preliminar de pantallas	▪ RUP	12/06/2009	18/06/2009	▪ Alexandra Maurad	$ 12,00	Prototipo de Pantallas
Descripción de los casos de uso	▪ RUP	19/06/2009	30/06/2009	▪ Alexandra Maurad	$ 15,00	Tabla de descripción de Casos de Uso
Elaboración (diseño)						
Refinamiento del Prototipo de pantallas	▪ RUP	01/07/2009	06/07/2009	▪ Alexandra Maurad	$ 10,00	Prototipo de Pantallas
Modelado: casos de uso	▪ RUP	01/07/2009	01/07/2009	▪ Alexandra Maurad	$ 2,50	Diagrama de Casos de

						Uso
Modelado: paquetes	▪ RUP	02/07/2009	02/07/2009	▪ Alexandra Maurad	$ 2,50	Diagrama de Paquetes
Modelado: componentes	▪ RUP	03/07/2009	03/07/2009	▪ Alexandra Maurad	$ 2,50	Diagrama de Componentes
Modelado: despliegue	▪ RUP	06/07/2009	06/07/2009	▪ Alexandra Maurad	$ 2,50	Diagrama de Despliegue
Modelado: clases final	▪ RUP	07/07/2009	15/07/2009	▪ Alexandra Maurad	$ 17,50	Diagramas de Clases
Modelado: robustez	▪ RUP	16/07/2009	29/07/2009	▪ Alexandra Maurad	$ 25,00	Diagramas de Robustez
Modelado: secuencia	▪ RUP	21/07/2009	03/08/2009	▪ Alexandra Maurad	$ 25,00	Diagramas de Secuencia
Transición (pruebas, integración)						
Verificación, Pruebas y corrección de los modelos	▪ RUP	01/07/2009	10/08/2009	▪ Alexandra Maurad	$ 10,00	Diagramas verificados y corregidos
Construcción (implementación)						
Codificación de la interfaz de usuario	▪ RUP	11/08/2009	27/08/2009	▪ Danny Muñoz	$ 30,00	Interfaz de Usuario
Codificación Submódulo: administración de proyectos	▪ RUP	28/08/2009	03/09/2009	▪ Danny Muñoz	$ 12,00	Submódulo: administración de proyectos
Codificación Submódulo: diagramación de casos de uso	▪ RUP	04/09/2009	10/09/2009	▪ Danny Muñoz	$ 12,00	Submódulo: diagramación de Casos de Uso
Codificación Submódulo: diagramación de clases	▪ RUP	11/09/2009	17/09/2009	▪ Danny Muñoz	$ 12,00	Submódulo: diagramación de Clases

Codificación Submódulo: diagramación de paquetes	▪ RUP	18/09/2009	24/09/2009	▪ Danny Muñoz	$ 12,00	Submódulo: diagramación de Paquetes
Codificación Submódulo: diagramación de componentes	▪ RUP	25/09/2009	01/10/2009	▪ Danny Muñoz	$ 12,00	Submódulo: diagramación de Componentes
Codificación Submódulo: diagramación de despliegue	▪ RUP	02/10/2009	08/10/2009	▪ Danny Muñoz	$ 12,00	Submódulo: diagramación de Despliegue
Codificación Submódulo: diagramación de secuencia	▪ RUP	09/10/2009	20/10/2009	▪ Danny Muñoz	$ 20,00	Submódulo: diagramación de Secuencia
Codificación Submódulo: exportación a formato imagen	▪ RUP	21/10/2009	22/10/2009	▪ Danny Muñoz	$ 5,00	Submódulo: exportación a formato imagen
Codificación de submódulo: impresión	▪ RUP	23/10/2009	26/10/2009	▪ Danny Muñoz	$ 5,00	Submódulo: impresión
Transición (pruebas, integración)						
Verificación, pruebas y corrección al código fuente del módulo de diagramación	▪ RUP	11/08/2009	02/11/2009	▪ Alexandra Maurad ▪ Danny Muñoz	$ 12,00	Módulo de Diagramación

10.2.3 Objetivo Específico 3

Objetivo Específico.- Desarrollar un módulo de generación de código fuente para el Lenguaje de Programación C Sharp, a partir de los diagramas de clases y secuencia.						
Actividad o Tarea	**Metodología**	**Fecha**		**Responsables**	**Presu-puesto**	**Resultados Esperados**
		Inicio	**Final**			
Incepción (análisis)						
Determinación y filtrado de requerimientos	▪ Observación directa ▪ Método Deductivo ▪ RUP	03/11/2009	03/11/2009	▪ Alexandra Maurad	$ 5,00	Listado de Requerimientos
Identificación y verificación de casos de uso	▪ RUP	04/11/2009	04/11/2009	▪ Alexandra Maurad	$ 5,00	Casos de Uso validados
Prototipado de pantallas	▪ RUP	05/11/2009	05/11/2009	▪ Alexandra Maurad	$ 5,00	Prototipo de Pantallas
Descripción de los casos de uso	▪ RUP	06/11/2009	09/11/2009	▪ Alexandra Maurad	$ 5,00	Tabla de descripción de Casos de Uso
Elaboración (diseño)						
Modelado del módulo de generación de código fuente	▪ RUP	10/11/2009	13/11/2009	▪ Alexandra Maurad	$ 10,50	Diagramas del Módulo de Generación de código fuente
Transición (pruebas, integración)						
Verificación, pruebas y	▪ RUP	16/11/2009	16/11/2009	▪ Alexandra	$ 2,50	Diagramas verificados y

corrección del modelado				Maurad ▪ Danny Muñoz		corregidos
Construcción (implementación)						
Codificación del módulo de generación de código fuente	▪ RUP	17/11/2009	25/11/2009	▪ Danny Muñoz	$ 20,50	Módulo de Generación de Código Fuente
Transición (pruebas, integración)						
Verificación y optimización del código fuente	▪ RUP	26/11/2009	27/11/2009	▪ Alexandra Maurad ▪ Danny Muñoz	$ 7,50	Código fuente optimizado
Pruebas y corrección al módulo de generación de código fuente	▪ RUP	30/11/2009	01/12/2009	▪ Alexandra Maurad ▪ Danny Muñoz	$ 5,00	Módulo de Generación de Código Fuente depurado

10.3 Matriz de Control de Resultados

No.	Resultados	Fecha	Firma Director de Tesis
Analizar las herramientas UML 2.0 más utilizadas por desarrolladores de SW			
1	Documentación de las herramientas basadas en UML 2.0	09/02/2009	
2	Listado de las herramientas basadas en UML 2.0 más utilizadas por los desarrolladores de software	07/05/2009	
3	Instaladores de las herramientas identificadas	20/05/2009	
4	Herramientas instaladas	20/05/2009	
5	Listado de las prestaciones de cada una de las herramientas instaladas	25/05/2009	
6	Documentación de fortalezas y debilidades de cada herramienta	25/05/2009	
7	Comparativa de las ventajas y desventajas de las herramientas más utilizadas por los desarrolladores de software	29/05/2009	
Módulo de Diagramación			
8	Listado de Requerimientos	03/06/2009	
9	Tabla de Casos de uso validados	05/06/2009	
10	Documentación del Dominio	11/06/2009	
11	Prototipo de Pantallas	18/06/2009	
12	Tabla de descripción de Casos de Uso	30/06/2009	
13	Prototipo de Pantallas	06/07/2009	
14	Diagrama de Casos de Uso	01/07/2009	
15	Diagrama de Paquetes	02/07/2009	
16	Diagrama de Componentes	03/07/2009	
17	Diagrama de Despliegue	06/07/2009	
18	Diagramas de Clases	15/07/2009	
19	Diagramas de Robustez	29/07/2009	
20	Diagramas de Secuencia	03/08/2009	

21	Diagramas verificados y corregidos	10/08/2009	
22	Interfaz de Usuario	27/08/2009	
23	Submódulo: Administración de Proyectos	03/09/2009	
24	Submódulo: diagramación de Casos de Uso	10/09/2009	
25	Submódulo: diagramación de Clases	17/09/2009	
26	Submódulo: diagramación de Paquetes	24/09/2009	
27	Submódulo: diagramación de Componentes	01/10/2009	
28	Submódulo: diagramación de Despliegue	08/10/2009	
29	Submódulo: diagramación de Secuencia	20/10/2009	
30	Submódulo: exportación a formato imagen	22/10/2009	
31	Submódulo: impresión	26/10/2009	
32	Módulo de Diagramación	02/11/2009	
Módulo de Generación de Código Fuente			
33	Listado de Requerimientos	03/11/2009	
34	Casos de Uso validados	04/11/2009	
35	Prototipo de Pantallas	05/11/2009	
36	Tabla de descripción de Casos de Uso	09/11/2009	
37	Diagramas del Módulo de Generación de código fuente	13/11/2009	
38	Diagramas verificados y corregidos	16/11/2009	
39	Módulo de Generación de Código Fuente	25/11/2009	
40	Código fuente optimizado	27/11/2009	
41	Módulo de Generación de Código Fuente depurado	01/12/2009	