

Índice de contenidos

BLOQUE I: MEMORIA	8
1. INTRODUCCIÓN	8
1.1 IDENTIFICACIÓN DEL PROYECTO	8
1.2 ESTRUCTURA DE LA DOCUMENTACIÓN	8
1.3 ORIGEN DEL PROYECTO.....	9
2. VISIÓN GENERAL DEL SISTEMA	10
2.1 DESCRIPCIÓN GENERAL DEL SISTEMA	10
2.2 DESCRIPCIÓN DE LAS DISTINTAS PARTES DEL SISTEMA.....	11
2.3 TECNOLOGÍAS EMPLEADAS.....	13
2.4 GENERACIÓN DE CÓDIGO	15
2.5 ARQUITECTURA DE SOFTWARE DIRIGIDA POR MODELOS (MDA)	18
2.6 XMI	20
2.6.1 MOF	22
2.6.2 Ventajas y desventajas del uso de XMI.....	23
2.7 FORMATO ZIP DE LA HERRAMIENTA.....	23
2.8 COMPARATIVA CON OTROS GENERADORES DE CÓDIGO	25
3. ARQUITECTURA DEL SISTEMA.....	26
3.1 ARQUITECTURA FÍSICA.....	26
3.2 ARQUITECTURA LÓGICA	27
4. ENTORNO DE IMPLANTACIÓN	28
4.1 SOFTWARE	28
4.2 HARDWARE.....	28
5. HERRAMIENTAS EMPLEADAS.....	28
6. METODOLOGÍA.....	29
7. PLANIFICACIÓN.....	30
7.1 PLANIFICACIÓN ESTIMADA.....	30
7.2 DURACIÓN REAL.....	31
7.3 CONCLUSIONES DE LA PLANIFICACIÓN.....	32
8. PRESUPUESTO	33
8.1 COSTE DE SOFTWARE	33
8.2 COSTE DE HARDWARE	33
8.3 COSTE DE RECURSOS HUMANOS	33
8.4 COSTE TOTAL.....	34
9. CONCLUSIONES.....	34
2.1 PROBLEMAS ENCONTRADOS	34
2.2 POSIBLES AMPLIACIONES	35
2.3 CONCLUSIONES.....	35
BLOQUE II: MANUAL TÉCNICO	37
1. ESPECIFICACIÓN DE REQUISITOS.....	37
1.1 OBJETIVOS PRINCIPALES.....	37
1.2 OBJETIVOS ADICIONALES.....	37
1.3 CARACTERÍSTICAS DE USUARIO	38
2. ANÁLISIS	39
2.1 DIAGRAMA DE CASOS DE USO	39

2.2 DESCRIPCIÓN DE LOS CASOS DE USO	42
2.2.1 <i>Caso de uso: Nuevo Proyecto.....</i>	42
2.2.2 <i>Caso de uso: Abrir Proyecto.....</i>	42
2.2.3 <i>Caso de uso: Importar Archivo XMI</i>	43
2.2.4 <i>Caso de uso: Generar Código Java.....</i>	44
2.2.5 <i>Caso de uso: Modificar Preferencias.....</i>	45
2.2.6 <i>Caso de uso: Seleccionar Clase Principal.....</i>	46
2.2.7 <i>Caso de uso: Guardar Proyecto.....</i>	46
2.2.8 <i>Caso de uso: Guardar Proyecto Como</i>	47
2.2.9 <i>Caso de uso: Refrescar contenidos</i>	48
2.2.10 <i>Caso de uso: Fijar Diagrama.....</i>	48
2.2.11 <i>Caso de uso: Ocultar Menú Principal.....</i>	49
2.2.12 <i>Caso de uso: Borrar todos los elementos.....</i>	49
2.2.13 <i>Caso de uso: Gestionar Clase</i>	50
2.2.14 <i>Caso de uso: Gestionar Atributos.....</i>	51
2.2.15 <i>Caso de uso: Gestionar Métodos</i>	53
2.2.16 <i>Caso de uso: Nuevo Archivo Java.....</i>	56
2.2.17 <i>Caso de uso: Abrir Archivo Java.....</i>	56
2.2.18 <i>Caso de uso: Borrar Archivo Java.....</i>	57
2.2.19 <i>Caso de uso: Guardar Todo.....</i>	57
2.2.20 <i>Caso de uso: Abrir contendor de archivos.....</i>	58
2.2.21 <i>Caso de uso: Abrir Archivo en pestaña</i>	58
2.2.22 <i>Caso de uso: Imprimir Archivo</i>	59
2.2.23 <i>Caso de uso: Guardar Pestaña.....</i>	59
2.2.24 <i>Caso de uso: Compilar Pestaña.....</i>	60
2.2.25 <i>Caso de uso: Cerrar Pestaña</i>	61
2.2.26 <i>Caso de uso: Copiar selección de código.....</i>	61
2.2.27 <i>Caso de uso: Pegar selección de código.....</i>	62
2.2.28 <i>Caso de uso: Cortar selección de código.....</i>	62
2.2.29 <i>Caso de uso: Cambiar a Vista Diseño.....</i>	63
2.2.30 <i>Caso de uso: Generar Documentación Javadoc.....</i>	63
2.2.31 <i>Caso de uso: Compilar Proyecto</i>	64
2.2.32 <i>Caso de uso: Ejecutar Proyecto.....</i>	64
3. DISEÑO	65
3.1 PATRÓN UTILIZADO.....	65
3.2 DIAGRAMAS DE SECUENCIA DE LA HERRAMIENTA	66
3.2.1 <i>Diagrama de secuencia: Nuevo Proyecto</i>	66
3.2.2 <i>Diagrama de secuencia: Abrir Proyecto</i>	67
3.2.3 <i>Diagrama de secuencia: Importar Archivo XMI</i>	67
3.2.4 <i>Diagrama de secuencia: Generar código Java</i>	68
3.2.5 <i>Diagrama de secuencia: Modificar Preferencias</i>	69
3.2.6 <i>Diagrama de secuencia: Seleccionar Clase Principal</i>	69
3.2.7 <i>Diagrama de secuencia: Guardar Proyecto</i>	70
3.2.8 <i>Diagrama de secuencia: Guardar Proyecto Como</i>	70
3.2.9 <i>Diagrama de secuencia: Refrescar contenidos</i>	71
3.2.10 <i>Diagrama de secuencia: Fijar Diagrama</i>	71
3.2.11 <i>Diagrama de secuencia: Ocultar Menú Principal.....</i>	71
3.2.12 <i>Diagrama de secuencia: Borrar todos los elementos</i>	72
3.2.13 <i>Diagrama de secuencia: Gestionar Clase</i>	72
3.2.14 <i>Diagrama de secuencia: Gestionar Atributos</i>	74
3.2.15 <i>Diagrama de secuencia: Gestionar Métodos.....</i>	75
3.2.16 <i>Diagrama de secuencia: Nuevo Archivo Java</i>	77
3.2.17 <i>Diagrama de secuencia: Abrir Archivo Java</i>	78
3.2.18 <i>Diagrama de secuencia: Borrar Archivo Java</i>	78
3.2.19 <i>Diagrama de secuencia: Guardar Todo</i>	79

3.2.20	<i>Diagrama de secuencia: Abrir contenedor de archivos</i>	79
3.2.21	<i>Diagrama de secuencia: Imprimir Archivo</i>	80
3.2.22	<i>Diagrama de secuencia: Abrir archivo en pestaña</i>	80
3.2.23	<i>Diagrama de secuencia: Guardar Pestaña</i>	80
3.2.24	<i>Diagrama de secuencia: Compilar Pestaña</i>	81
3.2.25	<i>Diagrama de secuencia: Cerrar Pestaña</i>	81
3.2.26	<i>Diagrama de secuencia: Copiar selección de código</i>	81
3.2.27	<i>Diagrama de secuencia: Cortar selección de código</i>	82
3.2.28	<i>Diagrama de secuencia: Pegar selección de código</i>	82
3.2.29	<i>Diagrama de secuencia: Cambiar a Vista Diseño</i>	82
3.2.30	<i>Diagrama de secuencia: Generar Documentación Javadoc</i>	83
3.2.31	<i>Diagrama de secuencia: Compilar Proyecto</i>	83
3.2.32	<i>Diagrama de secuencia: Ejecutar Proyecto</i>	83
3.3	DIAGRAMA DE CLASES	84
3.4	DESCRIPCIÓN DE LAS CLASES	84
3.4.1	<i>Clase UMLClass</i>	85
3.4.2	<i>Clase UMLAttribute</i>	85
3.4.3	<i>Clase UMLMethod</i>	85
3.4.4	<i>Clase UMLParameter</i>	86
3.4.5	<i>Clase UMLAssociation</i>	86
3.4.6	<i>Clase UMLGeneralization</i>	86
3.4.7	<i>Clase ClassContainer</i>	86
3.4.8	<i>Clase AssociationTable</i>	87
3.4.9	<i>Clase GeneralizationTable</i>	87
3.4.10	<i>Clase DataType</i>	87
3.4.11	<i>Clase DataTypeTable</i>	87
3.4.12	<i>Clase CodeGenerator</i>	87
3.4.13	<i>Clase ParseXMI</i>	88
3.4.14	<i>Clase Controller</i>	88
3.4.15	<i>Clase DocumentRender</i>	89
3.5	DIAGRAMAS DE CLASES PARCIALES	89
3.5.1	<i>Diagrama de clases parciales: Nuevo Proyecto</i>	89
3.5.2	<i>Diagrama de clases parciales: Abrir Proyecto</i>	89
3.5.3	<i>Diagrama de clases parciales: Importar archivo XMI</i>	90
3.5.4	<i>Diagrama de clases parciales: Generar código Java</i>	90
3.5.5	<i>Diagrama de clases parciales: Guardar Proyecto</i>	90
3.5.6	<i>Diagrama de clases parciales: Guardar Proyecto Como</i>	91
3.5.7	<i>Diagrama de clases parciales: Borrar todos los elementos</i>	91
3.5.8	<i>Diagrama de clases parciales: Gestionar Clase</i>	91
3.5.9	<i>Diagrama de clases parciales: Gestionar Atributos</i>	92
3.5.10	<i>Diagrama de clases parciales: Gestionar Métodos</i>	94
3.5.11	<i>Diagrama de clases parciales: Nuevo Archivo Java</i>	95
3.5.12	<i>Diagrama de clases parciales: Abrir Archivo Java</i>	96
3.5.13	<i>Diagrama de clases parciales: Borrar archivo Java</i>	96
3.5.14	<i>Diagrama de clases parciales: Guardar Todo</i>	96
3.5.15	<i>Diagrama de clases parciales: Imprimir Archivo</i>	97
3.5.16	<i>Diagrama de clases parciales: Guardar Pestaña</i>	97
4.	IMPLEMENTACIÓN.....	98
4.1	DIAGRAMA DE COMPONENTES	98
4.2	DIAGRAMAS DE DESPLIEGUE	99
5.	PRUEBAS.....	100
5.1	PRUEBAS DEL SISTEMA	100
5.2	PRUEBAS DE SOFTWARE.....	100
5.3	PRUEBAS DE UNIDAD.....	101

5.3.1	<i>Pruebas de caja blanca</i>	101
5.3.2	<i>Pruebas de caja negra</i>	101
5.3.2.1	Formulario Nuevo Proyecto	102
5.3.2.2	Formulario Abrir Proyecto.....	102
5.3.2.3	Formulario Importar Proyecto	103
5.3.2.4	Formulario de Preferencias.....	104
5.3.2.5	Formulario Guardar Como	105
5.3.2.6	Formulario Generar Código.....	106
5.3.2.7	Formulario Añadir/Modificar Clase	106
5.3.2.8	Formulario Añadir/Modificar Atributo	107
5.3.2.9	Formulario Añadir/Modificar Método	108
5.3.2.10	Formulario Añadir/Modificar Parámetro	110
5.3.2.11	Formulario Añadir Archivo java.....	111
5.3.2.12	Formulario Nuevo Archivo java.....	112
6.	ANÁLISIS Y EJEMPLOS DE LA APLICACIÓN GENERADA	113
6.1	DIAGRAMA DE CASOS DE USO DE LA APLICACIÓN GENERADA	113
6.2	DIAGRAMA DE CLASES DE LA APLICACIÓN GENERADA	113
6.3	EJEMPLOS.....	115
6.3.1	<i>Ejemplo Agenda de contactos</i>	115
6.3.2	<i>Ejemplo Videoclub</i>	122
6.3.3	<i>Ejemplo Colegio</i>	126
6.3.4	<i>Ejemplo Tienda online</i>	130
6.3.5	<i>Ejemplo Alquiler de Vehículos</i>	133
1.	BIBLIOGRAFÍA.....	138
2.	DIRECCIONES DE INTERNET.....	139

Índice de ilustraciones

Ilustración 1: Descripción del sistema	12
Ilustración 2: Funcionamiento de un generador de código activo	16
Ilustración 3: Funcionamiento de un generador de código pasivo	16
Ilustración 4: Generador de código - Partial class generator	17
Ilustración 5: Flujo de desarrollo convencional.....	17
Ilustración 6: Flujo de desarrollo de un generador de código.....	18
Ilustración 7: Transformación de modelos.....	19
Ilustración 8: Ejemplo estándares OMG.....	20
Ilustración 9: Escenario de intercambio XMI.....	21
Ilustración 10: Integración de estándares con XMI	21
Ilustración 11: Comunicación de seis aplicaciones de diferentes tecnologías	22
Ilustración 12: Principales elementos del modelo MOF.....	22
Ilustración 13: Arquitectura de sistema	26
Ilustración 14: Arquitectura de Java	27
Ilustración 15: Tabla de tareas y su estimación temporal – Planificación estimada	30
Ilustración 16: Diagrama de Gannt - Planificación estimada.....	31
Ilustración 17: Tabla de tareas y su estimación temporal – Planificación real.....	31
Ilustración 18: Diagrama de Gannt - Planificación real	32
Ilustración 19: Diagrama Casos de uso Diseño.....	40
Ilustración 20: Diagrama casos de uso IDE de Java	41
Ilustración 21: Patrón MVC	65
Ilustración 22: Diagrama de Secuencia - Nuevo proyecto.....	66
Ilustración 23: Diagrama de secuencia - Abrir proyecto	67
Ilustración 24: Diagrama de secuencia - Importar archivo XMI	67
Ilustración 25: Diagrama de secuencia - Generar código	68
Ilustración 26: Diagrama de secuencia - Modificar preferencias	69
Ilustración 27: Diagrama de secuencia - Seleccionar clase principal.....	69
Ilustración 28: Diagrama de secuencia - Guardar proyecto	70
Ilustración 29: Diagrama de secuencia - Guardar Proyecto como	70
Ilustración 30: Diagrama de secuencia - Refrescar contenidos.....	71
Ilustración 31: Diagrama de secuencia - Fijar diagrama	71
Ilustración 32: Diagrama de secuencia - Ocultar menú principal.....	71
Ilustración 33: Diagrama de secuencia - Borrar todos los elementos	72
Ilustración 34: Diagrama de secuencia - Añadir Clase.....	72
Ilustración 35: Diagrama de secuencia - Borrar Clase	73
Ilustración 36: Diagrama de secuencia - Modificar clase	73
Ilustración 37: Diagrama de secuencia - Añadir atributo	74
Ilustración 38: Diagrama de secuencia - Borrar atributo	74
Ilustración 39: Diagrama de secuencia - Modificar atributo	75
Ilustración 40: Diagrama de secuencia - Añadir método	75
Ilustración 41: Diagrama de secuencia - Borrar método	76
Ilustración 42: Diagrama de secuencia - Modificar método.....	76
Ilustración 43: Diagrama de secuencia - Nuevo archivo java	77
Ilustración 44: Diagrama de secuencia - Abrir archivo Java	78
Ilustración 45: Diagrama de secuencia - Borrar archivo java	78
Ilustración 46: Diagrama de secuencia - Guardar Todo	79
Ilustración 47: Diagrama de secuencia - Abrir contenedor de archivos.....	79
Ilustración 48: Diagrama de secuencia - Imprimir archivo	80
Ilustración 49: Diagrama de secuencia - Abrir archivo en pestaña	80
Ilustración 50: Diagrama de secuencia - Guardar Pestaña.....	80
Ilustración 51: Diagrama de secuencia - Compilar pestaña	81
Ilustración 52: Diagrama de secuencia - Cerrar pestaña	81

Ilustración 53: Diagrama de secuencia - Copiar selección de código	81
Ilustración 54: Diagrama de secuencia - Cortar selección de código	82
Ilustración 55: Diagrama de secuencia - Pegar selección de código	82
Ilustración 56: Diagrama de secuencia - Cambiar a Vista Diseño.....	82
Ilustración 57: Diagrama de Secuencia - Generar documentación Javadoc.....	83
Ilustración 58: Diagrama de Secuencia - Compilar Proyecto.....	83
Ilustración 59: Diagrama de Secuencia - Ejecutar Proyecto	83
Ilustración 60: Diagrama de clases del sistema	84
Ilustración 61: Diagrama de clases parciales - Nuevo Proyecto	89
Ilustración 62: Diagrama de clases parciales - Abrir Proyecto.....	89
Ilustración 63: Diagrama de clases parciales - Importar archivo XMI.....	90
Ilustración 64: Diagrama de clases parciales - Generar código java.....	90
Ilustración 65: Diagrama de clases parciales - Guardar proyecto	90
Ilustración 66: Diagrama de clases parciales - Guardar proyecto como	91
Ilustración 67: Diagrama de clases parciales - Borrar todos los elementos	91
Ilustración 68: Diagrama de clases parciales - Añadir clase	91
Ilustración 69: Diagrama de clases parciales - Modificar Clase	92
Ilustración 70: Diagrama de clases parciales - Borrar clase.....	92
Ilustración 71: Diagrama de clases parciales - Añadir Atributo.....	92
Ilustración 72: Diagrama de clases parciales - Modificar Atributo	93
Ilustración 73: Diagrama de clases parciales - Borrar Atributo	93
Ilustración 74: Diagrama de clases parciales - Añadir método.....	94
Ilustración 75: Diagrama de clases parciales - Modificar método.....	94
Ilustración 76: Diagrama de clases parciales: Borrar método	95
Ilustración 77: Diagrama de clases parciales - Nuevo Archivo Java	95
Ilustración 78: Diagrama de clases parciales - Abrir Archivo Java	96
Ilustración 79: Diagrama de clases parciales - Borrar archivo Java	96
Ilustración 80: Diagrama de clases parciales - Guardar todo	96
Ilustración 81: Diagrama de clases parciales - Imprimir archivo	97
Ilustración 82: Diagrama de clases parcial - Guardar pestaña	97
Ilustración 83: Diagrama de componentes.....	98
Ilustración 84: Diagrama de despliegue	99
Ilustración 85: Diagrama de casos de uso de la aplicación generada	113
Ilustración 86: Diagrama de clases de la aplicación generada	114
Ilustración 87: Diagrama de clases Agenda - Modelado con ArgoUML.....	115
Ilustración 88: Archivo XMI ejemplo Agenda de contactos.....	116
Ilustración 89: Herramienta GiA Java - Ejemplo Agenda	117
Ilustración 90: Diagrama de clases Ejemplo Videoclub - Modelado con ArgoUML.....	122
Ilustración 91: Herramienta GiA Java - Ejemplo Videoclub	122
Ilustración 92: Diagrama de clase Ejemplo Colegio - Modelado con ArgoUML	126
Ilustración 93: Herramienta GiA Java - Ejemplo Agenda	126
Ilustración 94: Herramienta GiA Java - Ejemplo Tienda Online.....	131
Ilustración 95: Diagrama de clase Ejemplo Alquiler de vehículos - Modelado con ArgoUML.....	133

BLOQUE I: MEMORIA

1. INTRODUCCIÓN

1.1 Identificación del proyecto

Título: Generador integral de aplicaciones Java (GiA Java)

Código: ENI- 258

Autor: Digna Rodríguez Cudeiro

DNI: 44.486.540 - D

Fecha: Septiembre de 2009

Director: Dr. José Baltasar García Pérez-Schofield

Codirector: Dra. Lourdes Borrajo Diz

Área: Lenguajes y Sistemas Informáticos

Departamento de Informática

Universidad de Vigo

1.2 Estructura de la documentación

La documentación que se adjunta con este proyecto, consta de dos tomos divididos de la siguiente forma:

- **Manual Técnico :**

Documento que recoge toda la información sobre el proyecto desarrollado, objetivos, requerimientos hardware y software imprescindibles para poner en funcionamiento el sistema, así como las diversas pruebas realizadas durante y después de su desarrollo. Se observa una explicación detallada sobre las distintas fases de desarrollo, como son la fase de análisis, diseño e implementación, así como a las conclusiones a las que se han llegado una vez finalizado el desarrollo del sistema. Su finalidad es servir de guía para el mantenimiento y modificaciones en el sistema.

- **Manual Usuario :**

Es un documento dirigido al usuario final del sistema, pretende ser una guía explicativa para el manejo del mismo, dando a conocer cada una de las funcionalidades que el usuario puede desempeñar, de una manera detallada.

En este documento también se explican los detalles mínimos de la máquina en la que ese instalará el sistema y los pasos necesarios para realizar todas aquellas tareas de la puesta en

funcionamiento de la aplicación, como son la instalación y configuración del software utilizado y el acceso a la aplicación.

El CD-ROM adicional se encuentra dividido en los siguientes directorios:

Software: software necesario para la instalación, configuración y puesta en funcionamiento de la aplicación: JRE de java y ArgoUML.

Aplicación: Dentro de esta carpeta se encuentra otra denominada ‘GiaJava’ que almacena la herramienta de generación de código Java a partir de diagramas creados a partir de una herramienta de modelado UML que exporta los diagramas en formato XMI. También podemos encontrar dentro de esta una carpeta que contiene ejemplos de archivos XMI de entrada de la aplicación.

Documentación: Manual Técnico, Manual de Usuario y una carpeta con los ejemplos de código generado.

Código fuente: Carpeta que contiene todas las clases de la aplicación.

1.3 Origen del proyecto

Actualmente, debido a las exigencias del mercado y a la gran competitividad, las empresas necesitan desarrollar sistemas de información con calidad y dentro de unos plazos de tiempo cada vez menores, necesitan automatizar y acelerar procesos. En este escenario surge este proyecto, un generador integral de aplicaciones, que permitirá generar gran parte del código fuente de un sistema de información orientado a objetos a través de la transformación de modelos.

El principal aporte es agilizar el desarrollo de aplicaciones, proporcionando a los desarrolladores no tener que escribir código repetitivo al comienzo de la implementación del proyecto, y partir ya de una base consistente de código. El código fuente producido es más fácil de entender, de ser utilizado y por lo tanto de ser mantenido porque sigue unos patrones de codificación en relación a nombre de las clases, variables, métodos, alineamiento...

La generación de código crea una aproximación a la solución bastante buena debido a que intenta traducir lo que se definió en el análisis mediante diagramas UML, directamente al código, de todos modos el sistema contará también con la opción de añadir y modificar componentes para la regeneración de código posterior.

La utilidad del proyecto se basa principalmente en el aumento de productividad, puesto que partes de la aplicación se habrán creado sólo con el análisis.

También puede ayudarle a un usuario no cualificado a generar partes de una aplicación (incluso toda ella), puesto que él, describe su problema en un lenguaje no específico ni técnico más cercano al dominio del problema.

Por tanto el tiempo de desarrollo de la aplicación disminuye, así como la cualificación que se requiere por parte del creador.

Se pretende que la generación de código sea independiente de la herramienta de modelado, partiendo de un estándar de intercambio de datos como XMI.

2. VISIÓN GENERAL DEL SISTEMA

2.1 Descripción general del sistema

Este proyecto tiene por objetivo el desarrollo de un generador de código, con capacidad de producir código fuente en java a partir de un modelo orientado a objetos escrito en UML. Mediante una herramienta de diagramación UML, se creará un diagrama de clases que describa la estructura del sistema que se desea crear. Esta herramienta tiene que permitir la exportación del diagrama al formato de intercambio de metadatos XMI en su versión 1.2.

El sistema creado parte de este documento XMI o de otro modelo de datos que se detallará posteriormente, para realizar la generación de código base y de una interfaz gráfica de usuario para la aplicación.

No está orientado a todo tipo de problemas: es especialmente útil y sencillo de usar con problemas en los que todo está muy bien definido y el programador sería capaz de realizarlo "manualmente", sin embargo se trata de un trabajo tedioso y propenso a errores.

La solución al problema se genera de forma extensa y uniforme. Además para problemas de cierta entidad aporta un marco de trabajo útil para extender el código generado aplicando una metodología ya probada para problemas más pequeños.

Se ha denominado a la herramienta con el acrónimo GiA Java (Generador integral de aplicaciones Java) y será el nombre que se usará a posteriori en esta documentación.

2.2 Descripción de las distintas partes del sistema

La herramienta GiA Java consta de dos partes desde las que se realizan diferentes acciones tanto en el modelo como en el código, a continuación describiremos cada una de las partes.

1. Parte de Diseño:

Desde esta parte se realiza la generación de código a partir de una entrada de datos:

- Importando un archivo XMI versión 1.2. que ha sido creado previamente con una herramienta de diagramación como ArgouML.
- Creando un modelo de datos directamente en la aplicación. Este modelo de datos es un diagrama de clases.
- Abriendo un proyecto ya creado por la herramienta que se ha almacenado en un formato propio de la misma. Este formato consta de un conjunto de archivos que se explicarán más tarde comprimidos en un archivo ZIP.

Si se desea después de hacer la extracción de componentes del modelo se pueden realizar modificaciones en el mismo: Gestión de clases, atributos, métodos...

Y como último paso de esta parte la generación de código a partir de los componentes del modelo:

- Se generan las clases completas: constructores, métodos para acceder y modificar las propiedades de un objeto.
- Métodos para acceder a las relaciones entre objetos. Las relaciones que se han tenido en cuenta entre las diferentes clases del sistema son: relaciones de herencia (relaciones de generalización y especialización) y asociaciones (relaciones de agregación y composición).
- Se generan métodos en cada clase para guardar, buscar y recuperar los objetos en formato XML.
- Se genera la GUI de la aplicación, desde la que el usuario interactuará con la aplicación generada y podrá añadir, modificar o borrar datos de los objetos. Esta interfaz será la base de la aplicación software que deseemos implementar.

Los objetos se almacenarán en archivos externos y en formato XML. Cada clase tiene su correspondiente archivo XML, con todos los datos de los objetos de la clase.

2. Parte de IDE de Java

Es un entorno de desarrollo gráfico con capacidad de compilación y ejecución. Que consta de puntos de expansión para poder usar siempre métodos o archivos de código java externos si se desea.

El IDE provee de un marco de trabajo amigable similar a un bloc de notas y desde el se pueden editar los archivos de código fuente java generados, se pueden crear nuevos archivos y borrar los existentes.

Si el desarrollador no está contento con el resultado se puede retornar a la vista de Diseño y modificar el modelo de datos, para a posteriori volver a generar el código.

Desde el IDE se puede compilar, depurar y ejecutar el código generado y/o editado en la herramienta. Así mismo se generará la documentación javadoc de las clases creadas por el sistema.

A continuación se muestra un diagrama del funcionamiento del sistema:

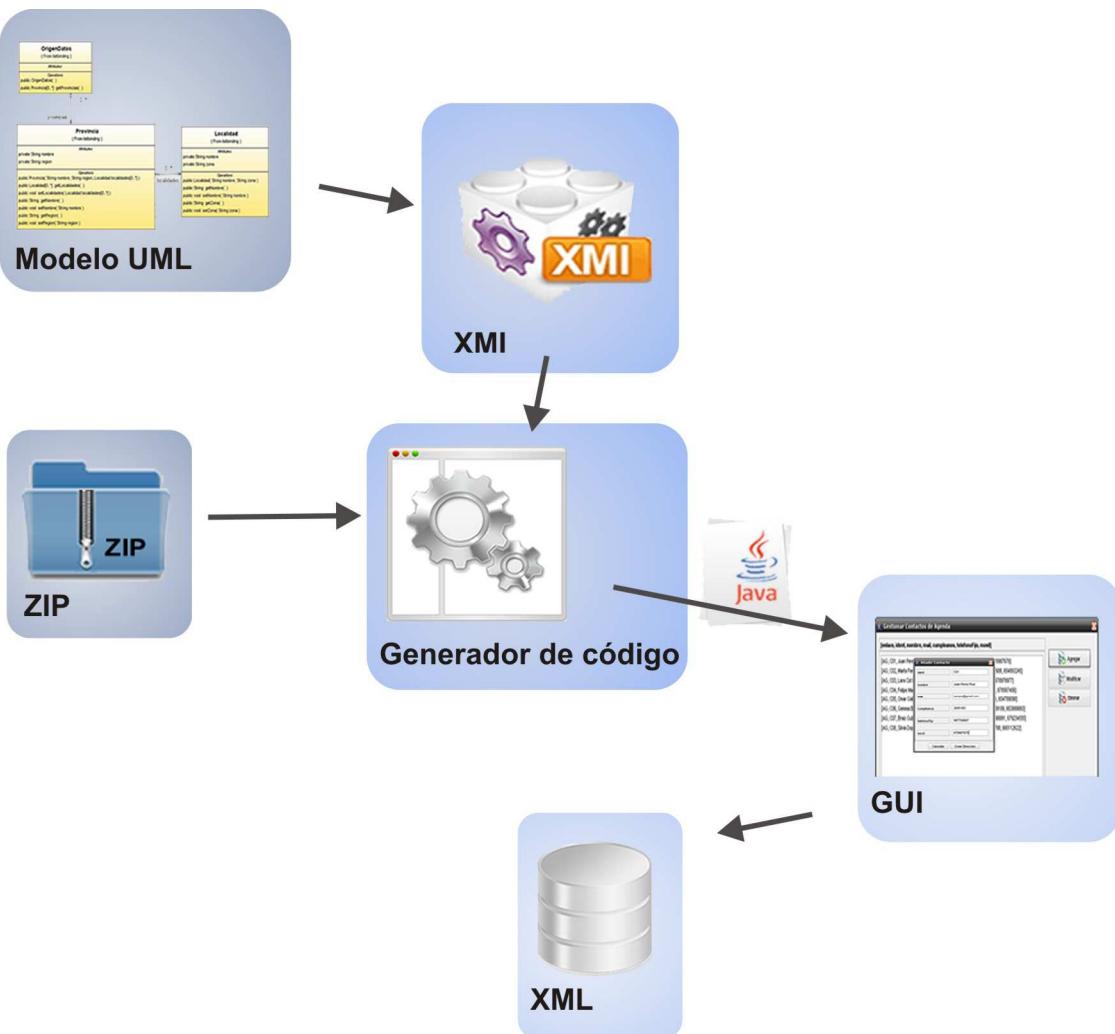
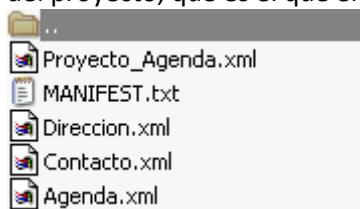


Ilustración 1: Descripción del sistema

En este esquema podemos comprobar cuál es el flujo de datos del sistema. Podemos partir del lenguaje UML nos da la capacidad para poder especificar, visualizar, construir y documentar un sistema software. A partir del diagrama de modelado UML se consigue un archivo XMI que es una de las posibles entradas de la herramienta.

Otra entrada del generador es un archivo ZIP. Los archivos ZIP, son archivos que ya se han creado en la herramienta con anterioridad. Comprimen una serie de archivos XML que representan a las clases del diagrama. Existe un archivo XML por cada clase del diagrama. El archivo ZIP también contiene un archivo MANIFEST.txt que contiene el nombre del proyecto. Es necesario porque dentro del ZIP también podemos encontrar un archivo XML con el nombre del proyecto, que es el que enlaza a todos los demás archivos XML de las clases.



El sistema extrae la información relevante de cualquiera de las posibles entradas al mismo almacenándola en una estructura interna del programa para la posterior generación de código base y de interfaz. Permitiendo este código almacenar los datos en una estructura XML.

2.3 Tecnologías empleadas

En el desarrollo del proyecto se han empleado las siguientes Tecnologías:



Java es un lenguaje de programación interpretado y orientado a objetos. Algunas de sus ventajas son: simplicidad, orientación a objetos, portabilidad, es dinámico, multihilo, robusto, distribuido...

El entorno de ejecución Java (JRE) se ha convertido en un componente habitual en los PC de usuario de los sistemas operativos más usados en el mundo. Además, muchas aplicaciones Java lo incluyen dentro del propio paquete de la aplicación de modo que se ejecuten en cualquier PC.

Además Java dispone de librerías adicionales (API's) que nos permiten que nuestros programas realicen diversas funciones, trabajen con determinados ficheros, etc. Las principales API's que usaremos son:

- Swing: es una biblioteca gráfica para Java. Incluye widgets para crear una interfaz gráfica de usuario, tales como cajas de texto, botones, desplegables y tablas...
- Jdom: Es una biblioteca de código fuente para manipulaciones de datos XML optimizados para Java. A pesar de su similitud con DOM del consorcio World Wide Web (W3C), es una alternativa como documento para modelado de objetos que no está incluido en DOM. Dispone de métodos para la validación de ficheros XML y manejo de sus elementos mediante el tipo de dato Element.
- Dom: Proporciona un conjunto estándar de objetos para representar documentos HTML y XML en forma de un árbol.
- Jgraph: Biblioteca Java de Fuente Abierta para la Visualización de Gráficos.

Estándares del OMG: UML y XMI



El Object Management Group u OMG (de sus siglas en inglés Grupo de Gestión de Objetos) es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI o CORBA. Se formó en 1989 con el propósito de crear una arquitectura estándar para objetos distribuidos en redes (componentes). Es una organización sin ánimo de lucro que promueve el uso de tecnología orientada a objetos mediante guías y especificaciones para las mismas. El grupo está formado por compañías y organizaciones de software como: Hewlett-Packard (HP), IBM, Sun Microsystems o Apple Computer.

El compromiso asumido por el OMG busca el desarrollo de especificaciones para la industria del software que sean técnicamente 'excelentes', comercialmente viables e independientes del vendedor.

El OMG define *object management* como el desarrollo software que modela el mundo real mediante su representación como objetos. Estos objetos no son más que la encapsulación de atributos, relaciones y métodos de componentes software identificables.

Existen dos especificaciones del OMG utilizadas en el proyecto:

- UML: Lenguaje unificado de modelado



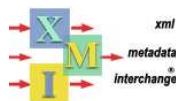
UML es un sistema notacional destinado al modelado de sistemas que utiliza un conjunto convenciones ampliamente aceptadas para representar los conceptos relacionados al análisis y diseño orientado a objetos (OOAD).

Se ha elegido porque es un lenguaje gráfico que cubre todas las fases del desarrollo y permite visualizar, especificar, construir y documentar las partes de un sistema software. Tiene metodologías y diagramas para reflejar el comportamiento de funciones, la estructura de las clases escritas en un lenguaje de programación orientado a objetos concreto, esquemas para reflejar la estructura de la base de datos y de componentes software reutilizables.

Además de en las distintas fases del desarrollo de la aplicación se ha hecho uso de este lenguaje a la hora de crear un diagrama de clases en una aplicación externa como ArgoUML que será una posible entrada al sistema.

El ciclo de vida que se ha seguido es iterativo e incremental, ya que no se tenían muy claros los conceptos al principio, en parte debido a la ausencia de un cliente, y algunos requisitos fueron cambiando. Además el ciclo de vida seguido está dirigido por casos de uso, ya que a partir de estos se guía todo el proceso de desarrollo y centrado en la arquitectura o estructura de las partes que formarán el sistema.

- XMI: XML Metadata Interchange



XMI o XML de Intercambio de Metadatos es una especificación para el Intercambio de Diagramas. Este estándar especifica como serializar modelos utilizando XML, y permite que distintas herramientas de modelado puedan intercambiar modelos entre sí.

La especificación para el intercambio de diagramas fue escrita para proveer una manera de compartir modelos UML entre diferentes herramientas de modelado. En versiones anteriores de UML se utilizaba un Schema XML para capturar los elementos utilizados en el diagrama; pero este Schema usaba ninguna forma para guardar como se debía de graficar el modelo.

Para solucionar este problema la nueva Especificación para el Intercambio de Diagramas fue desarrollada mediante un nuevo Schema XML que permite construir una representación SVG (Scalable Vector Graphics). Típicamente esta especificación es solamente utilizada por quienes desarrollan herramientas de modelado UML.

El soporte de estos dos estándares es de vital importancia para el desarrollador de generadores de código, ya que le abren un abanico de posibilidades al permitirle utilizar cualquier herramienta de modelado UML que soporte XMI para crear los modelos que serán el punto de entrada de su generador de código. También es posible usar herramientas para UML 1.4 y XMI 1.2 y otras variantes, gracias a la existencia de un conjunto de archivos XSLT que permiten su transformación a UML 2.0 sobre XMI 2.1.



XML: Extensible Markup Language

Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML (lenguaje de marcado generalizado) y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. En el proyecto se usa para almacenar los objetos de las clases generadas en el sistema.



HTML: Hypertext Markup Language

Lenguaje que se basa en el uso de etiquetas que establecen una estructura al documento, es muy sencillo y permite describir hipertexto, es decir, texto presentado de forma estructurada y agradable, con enlaces que conducen a otros documentos o fuentes de información relacionadas, y con inserciones multimedia (gráficos, sonido...) La descripción se basa en especificar en el texto la estructura lógica del contenido (títulos, párrafos de texto normal, enumeraciones, definiciones, citas, etc) así como los diferentes efectos que se quieren dar (especificar los lugares del documento donde se debe poner cursiva, negrita, o un gráfico determinado) y dejar que luego la presentación final de dicho hipertexto se realice por un programa especializado.

En el proyecto la ayuda se ha creado en formato HTML. Este lenguaje también se usa en la generación documental que automáticamente realiza Javadoc.

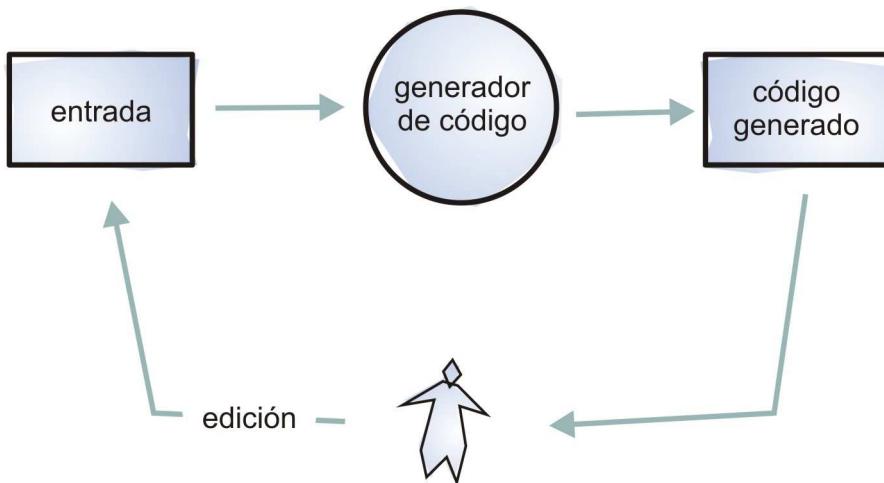
2.4 Generación de código

La generación de código es el proceso de crear un software que tiene la capacidad de escribir otros software. De esta forma un generador de código recibe unos parámetros de entrada, realiza un proceso sobre los mismos para producir código fuente como salida.

Los generadores de código pueden clasificarse según el tipo de generación de código utilizado o según el tipo de archivo de entrada y salida.

Según el tipo de generación se pueden dividir en activos o pasivos.

Los generadores activos pueden ser ejecutados varias veces sobre el mismo código de acuerdo con alteraciones realizadas en el input (archivo de entrada) el código generado nunca es modificado por el programador. Un generador de código activo mantiene el código durante la vida del proyecto a través de diferentes ciclos.

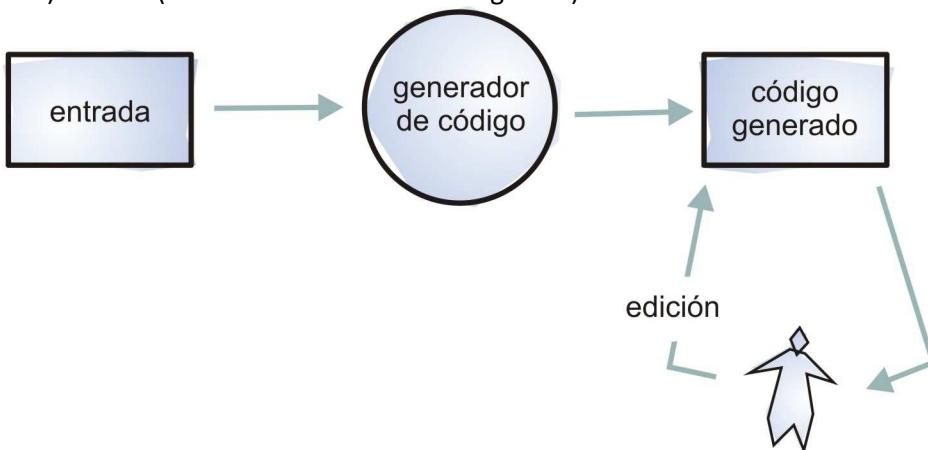
**Ilustración 2: Funcionamiento de un generador de código activo**

Los generadores pasivos permiten que el código sea generado una única vez y todo el mantenimiento necesario será realizado por parte de los desarrolladores.

Los generadores de código pasivos producen una aceleración inicial en la productividad del proyecto, pero en caso de que los requisitos se modifiquen se debe de modificar el input del generador de código y todas las modificaciones realizadas en la base del código inicial se perderán.

Muchos de los generadores de código pasivos son soluciones creadas por los propios desarrolladores para resolver tareas simples y repetitivas.

Un generador de código pasivo es muy útil en la producción de comentarios y anotaciones en el código. Este tipo de generador de código generalmente se difunde como wizard (software de asistente) en IDEs (entornos de desarrollo integrados).

**Ilustración 3: Funcionamiento de un generador de código pasivo**

GIA Java es un generador pasivo para producir código base (código de clase, getters y setters de los atributos, métodos de gestión de archivos XML, bloques de comentarios que generan documentación javadoc...) que a posteriori de la generación serán modificados por el programador.

Otra clasificación de los generadores de código se realiza en base a los archivos de entrada y salida. Existen generadores como Code Munger, Inline Code Expander o Mixed Code cuya entrada son archivos de código fuente que se transforman y producen un archivo fuente de salida. Existe otro tipo denominado Partial Class Generator que es el tipo de nuestro

generador GIA Java. Un generador de este tipo produce un conjunto básico de clases para implementar actividades simples.

Y en caso de que sea necesario se debe de crear extensiones de las clases para implementar nuevas funcionalidades. Este comportamiento se ilustra en la figura siguiente.

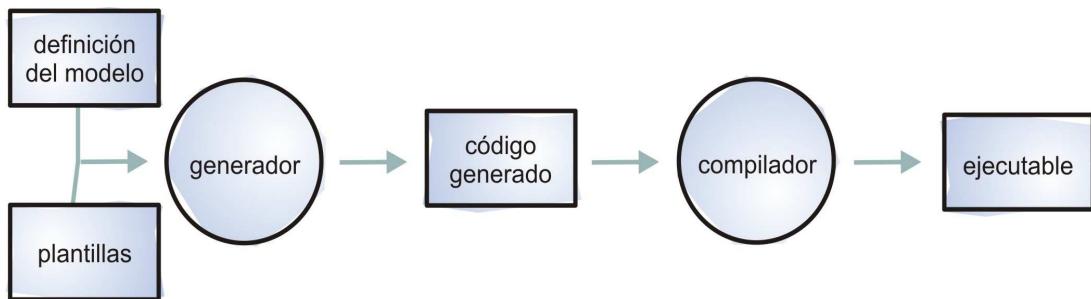


Ilustración 4: Generador de código - Partial class generator

El generador de código parte de un modelo de datos en el que está definido el sistema y utiliza para la generación de código templates. Los templates son documentos de texto plano, donde se escribe código y pueden existir algunos marcadores especiales. Estos marcadores serán reemplazados por valores durante la ejecución de un programa. No hay un formato predefinido para los marcadores, cada programador puede escoger el esquema que más le guste o que más le convenga.

Un generador de código de este tipo controla toda la estructura de acceso a los datos y la calidad del código está directamente asociada a los templates utilizados para la generación de código. Las modificaciones que se realicen en los templates serán contempladas en todo el código generado.

Flujo de desarrollo:

La utilización de un generador de código provoca una alteración en el flujo de desarrollo de un sistema. En el flujo convencional, representado en la ilustración 5 los desarrolladores escriben el código, realizan la compilación y a continuación realizan las pruebas . En caso de que sea necesario el código será editado reiniciando el flujo.

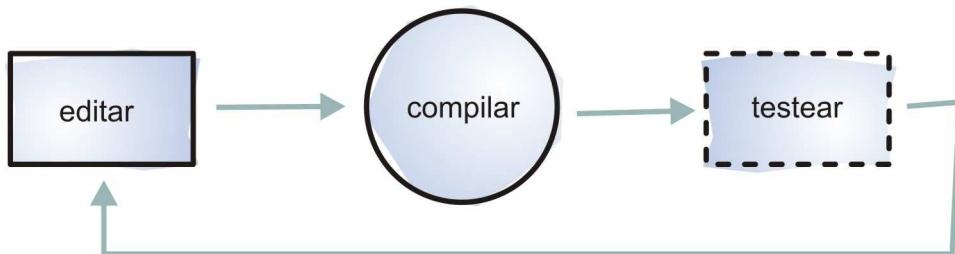


Ilustración 5: Flujo de desarrollo convencional

La utilización de un generador de código añade dos etapas, la creación de los templates y la generación del código fuente. La figura 6 muestra el desarrollo con un generador de código.

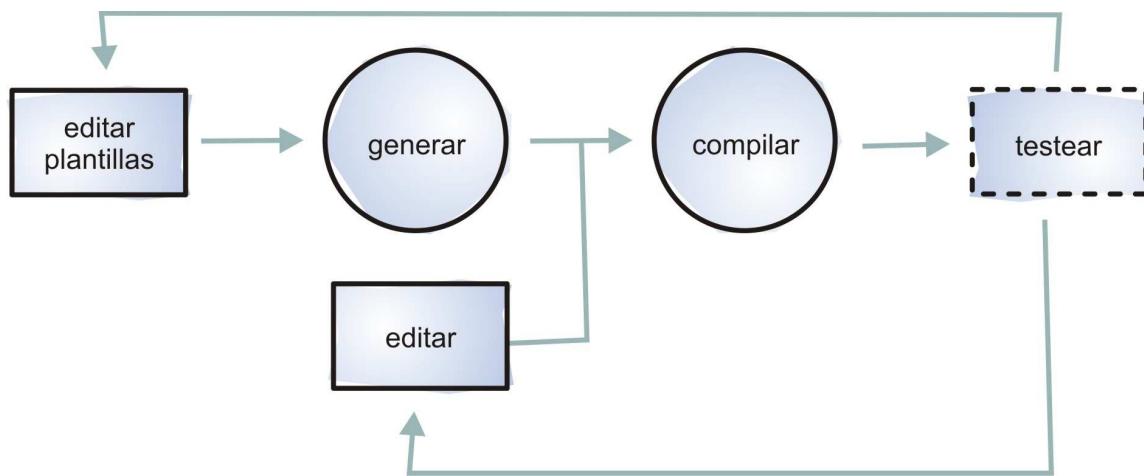


Ilustración 6: Flujo de desarrollo de un generador de código

2.5 Arquitectura de software dirigida por modelos (MDA)

MDA son las siglas de Model-Driven Architecture o arquitectura dirigida por modelos. Es un acercamiento al diseño de software, proporciona un conjunto de guías para estructurar especificaciones expresadas como modelos. Pero su característica más importante es que permite realizar diseños que sean independientes de la plataforma donde vayan a ser implementados e incluso del lenguaje de programación, para más tarde, generar automáticamente el código (o parte) del sistema descrito en el modelo.

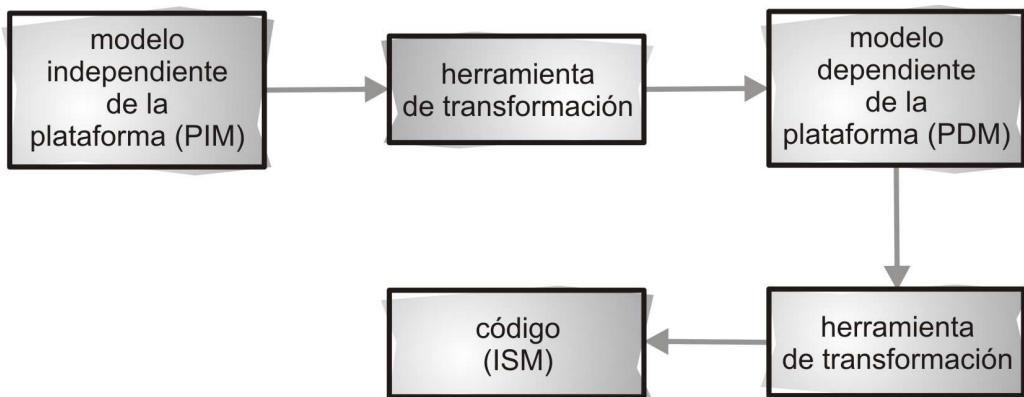
Un modelo es la descripción o especificación con un determinado propósito de un sistema o de sus partes. Un modelo generalmente está representado como una combinación de gráficos y texto. El texto puede estar escrito en algún lenguaje de modelado o mismo en lenguaje natural.

La arquitectura MDA fue propuesta y está respaldada por Object Management Group (OMG) y está relacionada con múltiples normas: UML, MOF, XML e XMI, Enterprise Distributed Object Computing, Software Process Engineering Metamodel, Common Warehouse Metamodel y muchas más.

Ofrece muchas ventajas como poder generar el framework y parte del código automáticamente a partir del modelo para diferentes plataformas y lenguajes.

Los modelos que se usen de entrada para el MDA pueden ser perfectamente diagramas de clases, de componentes , de despliegue ,etc. Es decir diagramas que están recogidos en la documentación y que fueron realizados a partir de la especificación de los requisitos.

MDA surge como una de las estrategias más prometedoras para el diseño y desarrollo de aplicaciones, debido a que unifica y simplifica las fases de modelado, diseño, implementación e integración de aplicaciones; convirtiendo el desarrollo de software en un proceso de transformación de modelos, lo que hace que el software se adapte con mayor facilidad a los cambios de tecnología.

**Ilustración 7: Transformación de modelos**

La transformación de modelos es el proceso de convertir un modelo en otro modelo del mismo sistema.

Usando la metodología MDA, la funcionalidad del sistema será definida en primer lugar como un modelo independiente de la plataforma (Platform-Independent Model o PIM) a través de un lenguaje específico para el dominio del que se trate. Dado un modelo de definición de la plataforma (Platform Definition Model o PDM) correspondiente a CORBA, .NET, Web, etc., el modelo PIM puede traducirse entonces a uno o más modelos específicos de la plataforma (Platform-specific models o PSMs, también llamados Platform-dependent models o PDMs) para la implementación correspondiente, usando diferentes lenguajes específicos del dominio, o lenguajes de propósito general como Java, C#, Python, etc. También denominados ISM o (Implementation specific model) La traducción entre el PIM y los PSMs se realizan normalmente utilizando herramientas automatizadas, como herramientas de transformación de modelos .

La arquitectura MDA proporciona mejoras en la productividad, portabilidad, interoperabilidad y el uso y mantenimiento que se hace de la documentación y del código.

OMG provee un framework conceptual y estándares para expresar modelos, relaciones y transformaciones modelo-a-modelo compuesto por: UML, MOF,XMI...

A continuación mostramos un ejemplo en el que están relacionados estos estándares siguiendo la arquitectura dirigida por modelos.

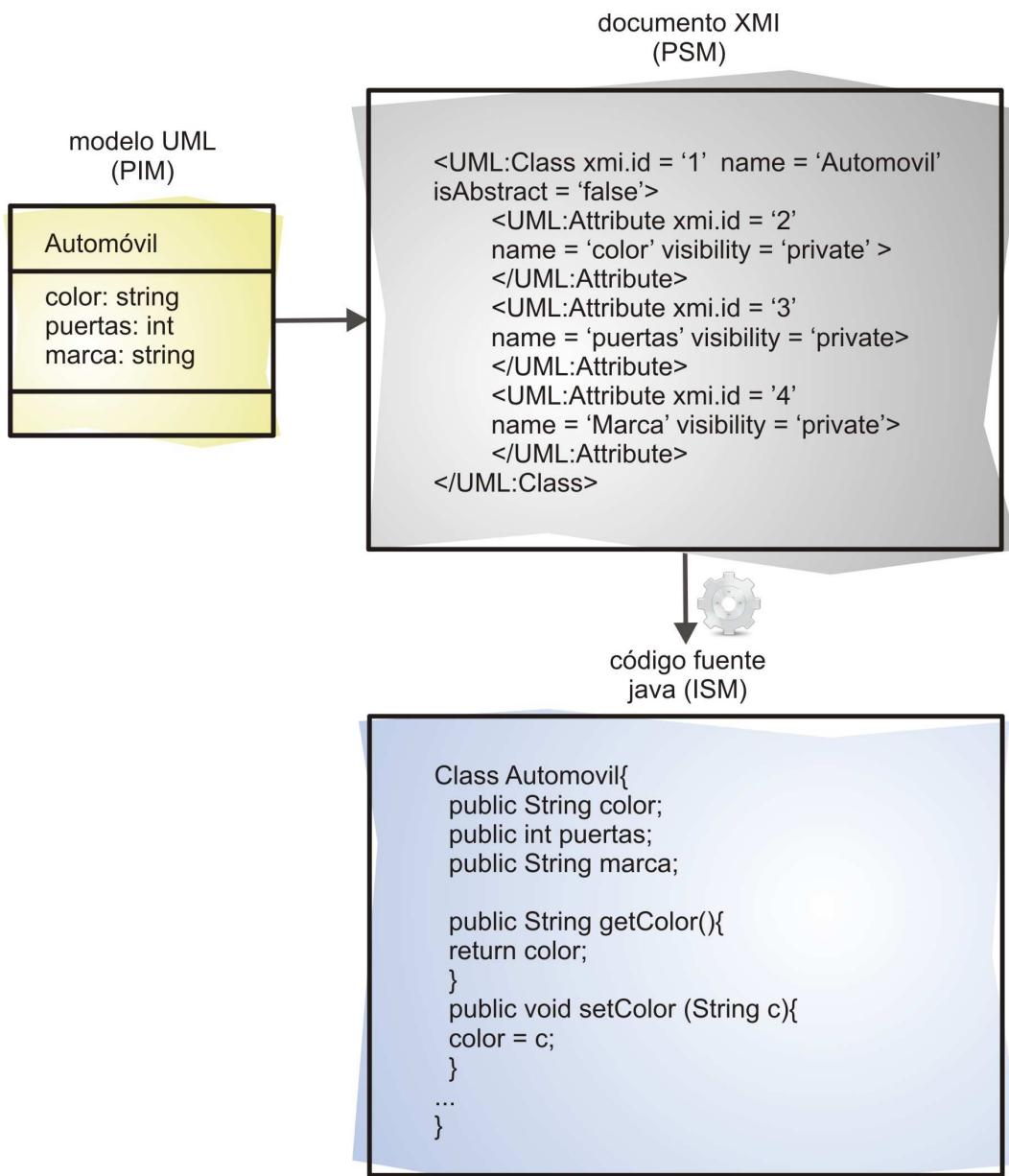


Ilustración 8: Ejemplo estándares OMG

2.6 XMI

XMI es el nombre que recibe el estándar para el intercambio de metamodelos usando XML. Su principal objetivo es permitir un intercambio de metainformación entre herramientas de modelado basadas en UML y repositorios de metainformación basados en MOF en heterogéneos entornos distribuidos. Realizando este intercambio mediante streams o ficheros con formato estándar basado en XML.

La arquitectura de XMI permite la simplificación de la comunicación entre aplicaciones de diferentes tecnologías ahorrando mucho trabajo y tiempo, además potencia la reutilización de objetos y componentes.

La siguiente figura muestra un posible escenario de intercambio usando XMI.

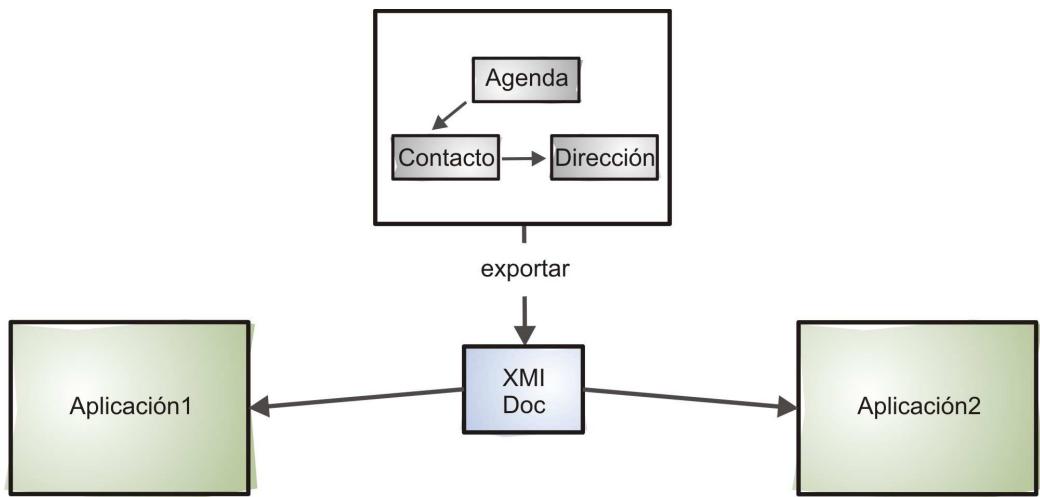


Ilustración 9: Escenario de intercambio XMI

Como ya hemos dicho XMI integra tres estándares: UML, XML y MOF.

UML es un estándar que define un lenguaje de modelado orientado a objetos que es soportado por una gama de herramientas de diseño gráfico y MOF es un estándar que define un marco de trabajo para definir modelos de metainformación y proporciona herramientas con interfaces programadas para almacenar y acceder a metainformación en un repositorio. XML va a permitir la comunicación entre las máquinas o herramientas.

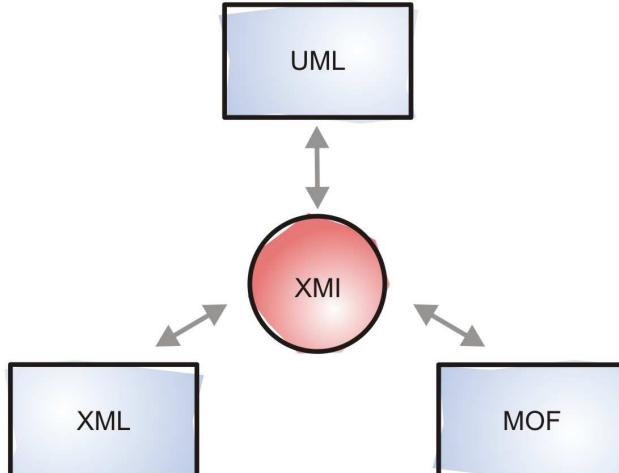


Ilustración 10: Integración de estándares con XMI

Existen diversidad de herramientas CASE, pongamos por ejemplo en la ilustración 11 tenemos 6 herramientas de diferentes tecnologías usando XMI (izquierda) y sin utilizarlo (derecha), en el primer caso sólo son necesarios 6 puentes de conexión y en el segundo se necesitan 30 puentes. Se puede observar como el sistema que usa XMI, para implementar las mismas conexiones presenta un número menor de de enlaces que el sistema que no lo usa. Lo que conlleva que la complejidad del intercambio de información sea menor.

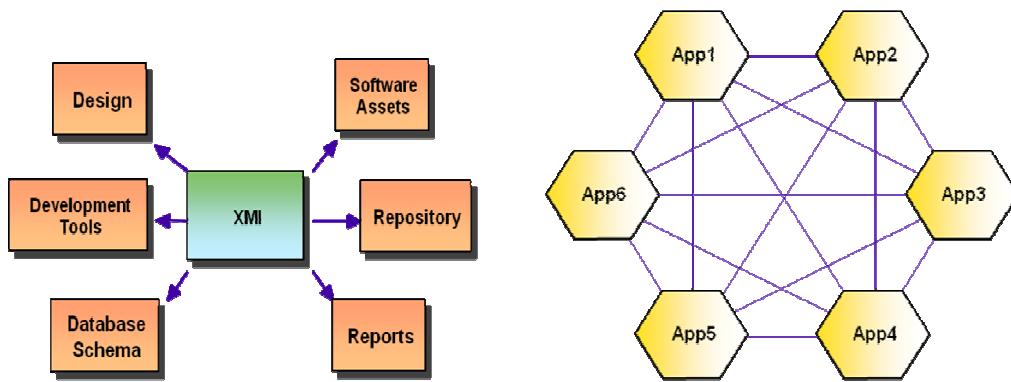


Ilustración 11: Comunicación de seis aplicaciones de diferentes tecnologías

2.6.1 MOF

Se trata de un estándar que define un conjunto de constructores que pueden ser usados para definir lenguajes de modelado.

La semántica de MOF define generalmente servicios para el repositorio de metadatos, para así permitir la construcción, la localización, la actualización, etc..

Es importante saber que la especificación MOF no define ninguna representación textual o gráfica del lenguaje. La siguiente figura muestra los elementos principales del patrón MOF y como están relacionados entre sí. Cada uno de los elementos representa un elemento del diagrama de clases UML.

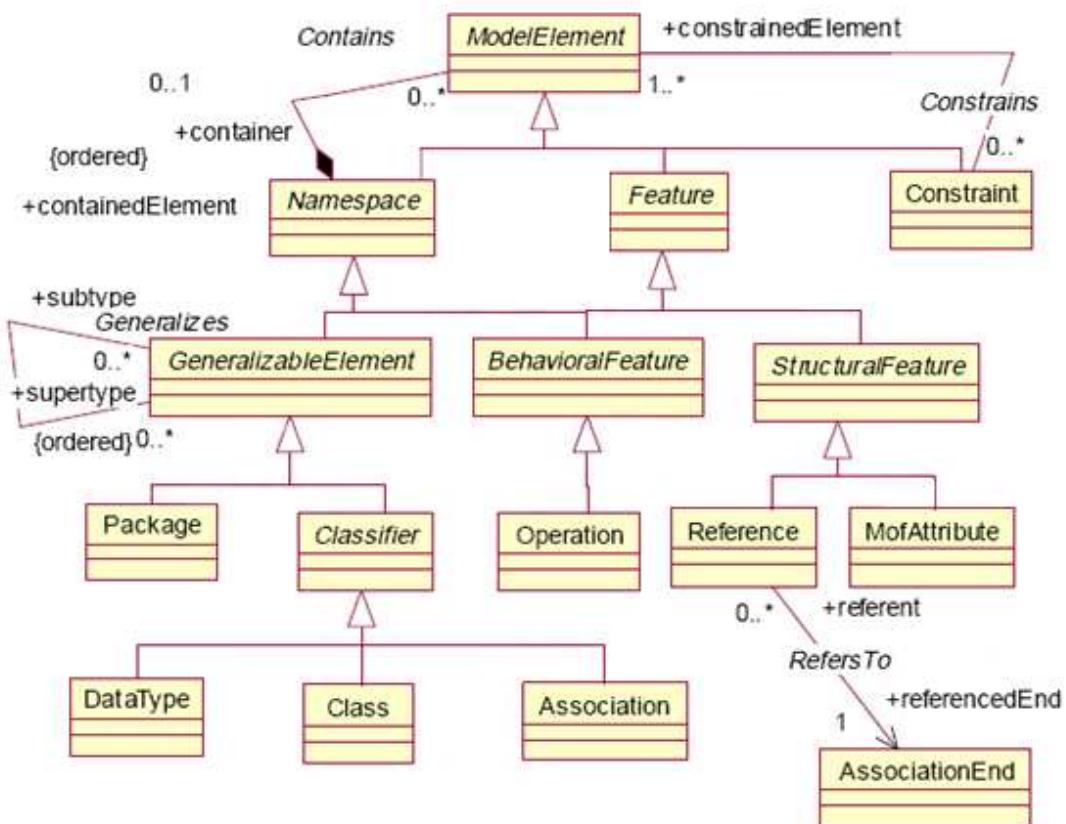


Ilustración 12: Principales elementos del modelo MOF

Es interesante explicar la organización de los elementos. Todas las clases son subclases del elemento del modelo ModelElement. Classifier es una interfaz que define clases (Class), tipos de datos (DataType) y asociaciones (Associations). Un Namespace se puede considerar como una agrupación de elementos. Los elementos que están contenidos en un namespace reciben el nombre de feature y pueden ser comportamientos (BehavioralFeature) o estructuras (StructuralFeature). Las operaciones o métodos (Operation) y las excepciones (Exception) son features de comportamiento. Los atributos (MofAttribute) y referencias (Reference) son features estructurales. Tanto Package como Classifier son subclases de GeneralizableElement y por tanto pueden ser generalizadas.

2.6.2 Ventajas y desventajas del uso de XMI

Ventajas:

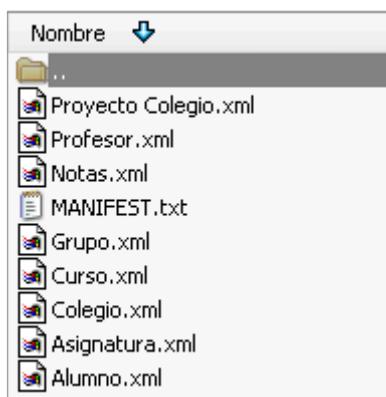
- Trabaja con Internet y está construido en base a estándares industriales como: XML, HTML, UML, MOF, etc.
- Un único formato de archivo para todas las herramientas CASE.
- Permite el intercambio de objetos entre aplicaciones.
- Reutilización de objetos y componentes.
- Método sencillo de empaquetar información y metainformación
- Existen gran cantidad de herramientas que usen XMI: Visual Paradigm for uml, Poseidon, NetBeans, altova, Microsoft Visio, Rational Rose, Eclipse UML...

Desventajas

- No permite recoger datos gráficos.
- Incompatibilidad entre diferentes versiones.
- Algunas herramientas únicamente importan o exportan.

2.7 Formato ZIP de la herramienta

A continuación explicaremos el formato de entrada ZIP de la herramienta. Cuando en la herramienta decidimos guardar el proyecto, el sistema crea una serie de archivos XML que comprime en un archivo ZIP. El contenido de este archivo ZIP lo ilustraremos un ejemplo:



- Se crea un archivo XML por cada clase del modelo, que guarda las características propias de una clase UML. En este caso se crea un archivo XML para las clases: Profesor, Notas, Grupo, Curso, Colegio, Asignatura y Alumno.

Si abrimos por ejemplo el archivo Profesor.xml nos encontramos con la siguiente estructura:

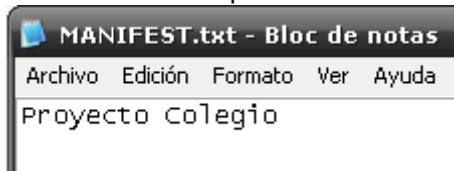
```

<?xml version="1.0" encoding="UTF-8" ?>
<uml:class
  xmlns:uml="http://www.template.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation= "http://www.template.com  schemaUML.xsd" >
<nameclass>Profesor</nameclass>
<visibility>1</visibility>
<abstract>no</abstract>
<attribute>
  <name_atb>nomProf</name_atb>
  <visib_atb>1</visib_atb>
  <type_atb>int</type_atb>
  <type_vector>null</type_vector>
  <static_atb>no</static_atb>
</attribute>
<attribute>
  <name_atb>cursos</name_atb>
  <visib_atb>1</visib_atb>
  <type_atb>Vector</type_atb>
  <type_vector>null</type_vector>
  <static_atb>no</static_atb>
</attribute>
</uml:class>

```

Como se puede comprobar para una clase se almacenan datos como nombre, visibilidad, abstracción, atributos y métodos(en este caso no tiene).Y para cada atributo o método sus características.

- En el archivo ZIP también se encuentra un archivo MANIFEST.txt cuya utilidad es reflejar cual es el nombre del proyecto y por tanto el archivo XML principal del mismo que también se encontrará en el archivo ZIP. En este caso dentro del archivo MANIFEST.txt nos encontramos escrito en su primera línea:



Lo que implica que el archivo 'Proyecto Colegio.xml' es el archivo principal del sistema.

- Por último queda explicar el contenido de este archivo principal del ZIP ('Proyecto Colegio.xml'). Es un XML raíz que enlaza a todos los archivos XML que se encuentran dentro del ZIP. Es decir es un vínculo a todas las clases del sistema. Su estructura es la siguiente:

```

<?xml version="1.0" encoding="UTF-8" ?>
<project>
  <file>Colegio.xml</file>
  <file>Alumno.xml</file>
  <file>Profesor.xml</file>
  <file>Asignatura.xml</file>
  <file>Notas.xml</file>
  <file>Curso.xml</file>
  <file>Grupo.xml</file>
</project>

```

2.8 Comparativa con otros generadores de código

En la actualidad existen varias aplicaciones, que nos ayudan a que el trabajo repetitivo se lleve a automatizar. Algunas son herramientas de UML que generan un código básico a partir de los diagramas, como pueden ser Visual Paradigm for UML o Enterprise Architect, que permiten dibujar todos los tipos de diagramas, realizar código inverso, generar código desde diagramas y generar documentación, pero tiene el inconveniente de ser aplicaciones comerciales, y en relación con nuestro proyecto no permiten generar código para el tratamiento de datos en formato XML, ni generar una interfaz para las aplicaciones.

AndroMDA es una herramienta para la generación extensible de código, que se adhiere al paradigma de la arquitectura dirigida por modelos. Es open source utiliza UML 1.4 y XMI 1.2, no posee herramienta de modelado, utiliza cartuchos como hibernate, web services , struts o EJB para transformar el modelo a código, Se utiliza como plugin que se puede utilizar con eclipse y genera código java.

OpenMDX es una plataforma MDA de código abierto basada en estándares del OMG. Genera J2SE,J2EE, CORBA y .NET. El proceso de desarrollo se centra en el modelado con diagramas MOF.

Otra herramienta conocida es Data layer Generator está diseñado para ahorrarle tiempo al desarrollar sus aplicaciones .NET, y para reducir el número de errores de código. Este programa es un generador de código C# y es freeware. En concordancia la aplicación desarrollada GIA Java también permite realizar funciones básicas de borrado, inserción y actualización de datos, pero directamente en su base de datos en SQL Server 2005/2008 en vez de utilizar el estándar XML.

BOUML es una herramienta CASE gratuita que permite trabajar con UML2 , y puede generar código java C++, e IDL , también es capaz de generar documentación en formato HTML .

Ninguna de estas herramientas generan código de lectura, escritura y búsqueda en archivos XML, tampoco permiten generar una interfaz para la aplicación.

Directamente relacionado con el proyecto están todos aquellas herramientas CASE que permiten exportar al formato estándar XMI 1.2, como pueden ser como software libre ArgoUML o Umbrello y como software comercial Rational Rose y Visual Paradigm. GiA Java permite importar el formato XMI 1.2.

3. ARQUITECTURA DEL SISTEMA.

3.1 Arquitectura física

La arquitectura de la herramienta se muestra a continuación. Se trata de una herramienta Java por lo que podrá ser ejecutado en cualquier plataforma o máquina que tenga implementado un intérprete Java.

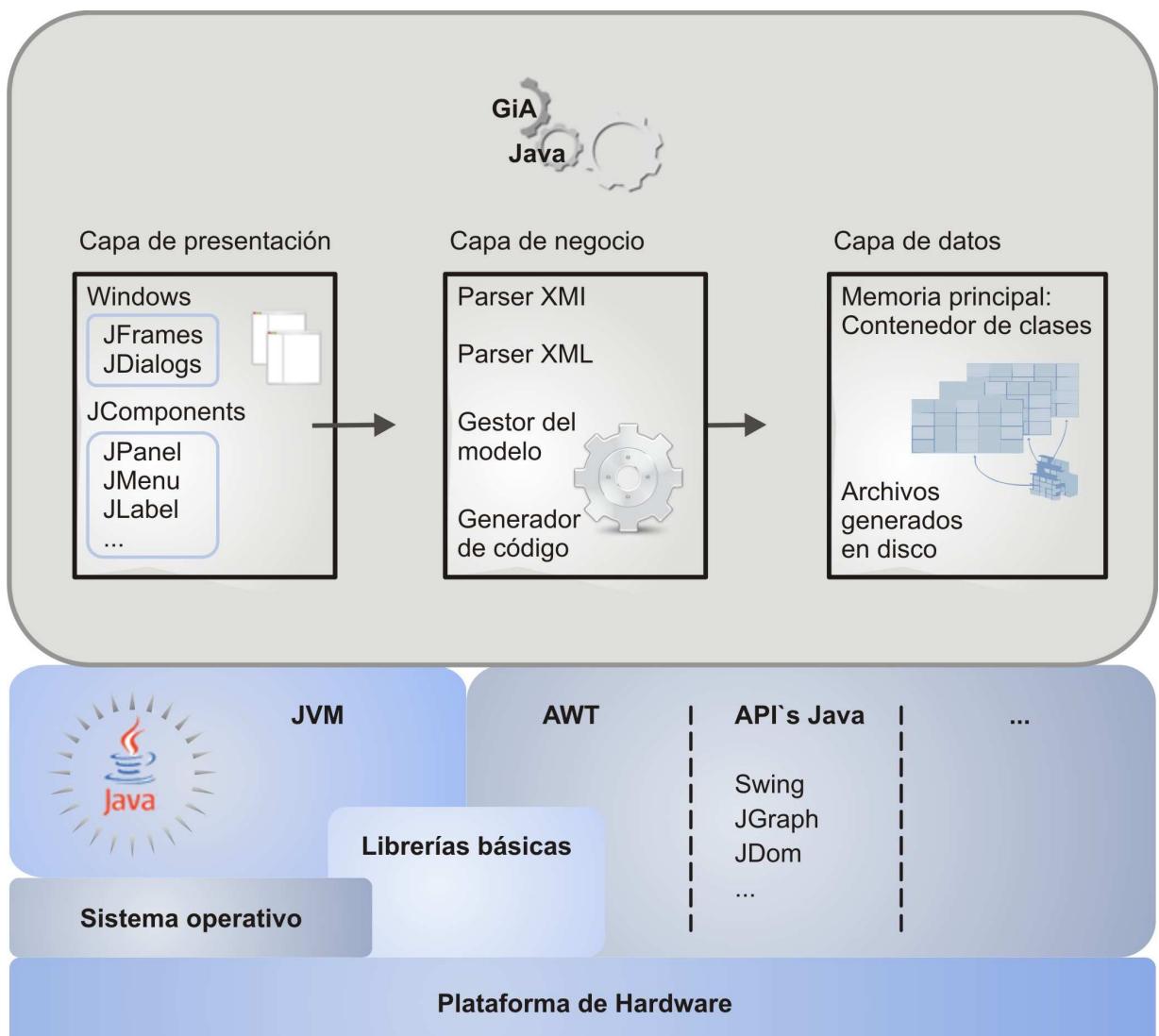


Ilustración 13: Arquitectura de sistema

Los diferentes componentes de la aplicación se detallan a continuación:

- Sistema Operativo : Cualquier S.O. que tenga un entorno de ejecución java.
- Java Virtual Machine : La máquina virtual de Java, es necesaria para poder ejecutar la aplicación.
- Librerías : Corresponde a todas las librerías con las que se comunica la aplicación para su correcto funcionamiento.

- Aplicación GiA Java: Las clases por las que se compone la aplicación, serán las encargadas de realizar las funcionalidades del sistema.

La aplicación mantiene una estructura en capas:

Capa de Presentación: La capa de presentación contiene los componentes necesarios para interactuar con el usuario de la aplicación. Permite exploración gráfica de las clases y acciones para manipular el modelo de forma directa de una forma sencilla. Esta capa procesa solicitudes para su envío a la lógica de negocio.

Capa de Negocio: consiste en la lógica que realiza las funciones principales de la aplicación: procesamiento de datos, parseamiento xmi, parseamiento del xml, administración del modelo y generación de código.

Capa de datos. La capa de datos está formada por los datos almacenados en las estructuras de datos internas del sistema, estos datos no son datos persistentes.

También se engloban en esta capa los archivos de código generados y almacenados en disco.

La herramienta que es ejecutada por un usuario en su equipo. La aplicación se carga en memoria al ejecutarse, permitiendo al usuario ejecutar las funciones implementadas en esta. Al cerrar la herramienta esta es desalojada de memoria y los recursos que estaba utilizando son liberados.

3.2 Arquitectura lógica

Se ha utilizado la plataforma de desarrollo de Java para la realización de la aplicación. La siguiente imagen muestra la arquitectura de Java.

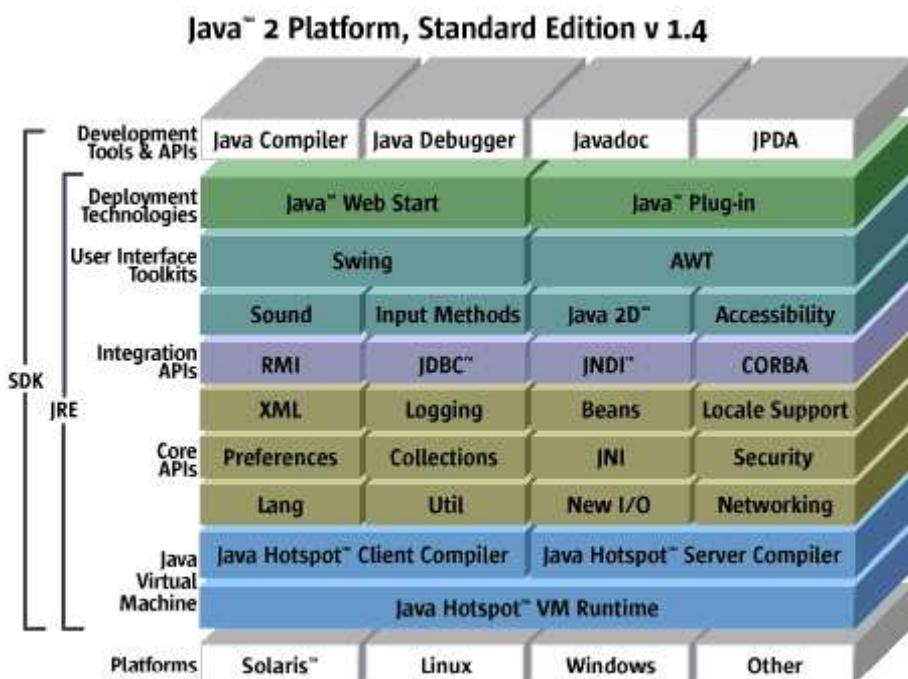


Ilustración 14: Arquitectura de Java

Reúne un conjunto de características, como independencia de plataformas, soporte para base de datos, soporte para XML, servicios Web, recursos de interfaz gráfica, entre otras.

4. ENTORNO DE IMPLANTACIÓN

4.1 Software

A continuación se muestra el software mínimo necesario para ejecutar el servidor.

Sistema Operativo Windows XP

JDK 1.5 o posterior

4.2 Hardware

En este apartado se muestra el hardware mínimo necesario para el equipo del servidor.

Intel Pentium 1.73 GHz.

1 Gbyte de memoria RAM.

5. HERRAMIENTAS EMPLEADAS

Durante el desarrollo de la aplicación, han sido utilizadas las siguientes herramientas:

Microsoft Project Professional



Herramienta gráfica de planificación. Durante la realización de este proyecto se ha empleado esta herramienta para crear tanto la planificación estimada como la real, y generar sus correspondientes diagramas de Gantt.

Visual Paradigm



Es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis, diseño orientado a objetos, construcción, pruebas y despliegue.

Jbuilder X Enterprise



Es un IDE (entorno de desarrollo integrado) para Java desarrollado Borland. Permite a los desarrolladores construir, probar, optimizar y mantener grandes aplicaciones. Este entorno incluye diseñadores swing con los que es fácil crear interfaces de usuario.

Adobe Fireworks CS4



Fireworks es una aplicación versátil para crear, editar y optimizar gráficos Web. Permite crear y editar imágenes de mapa de bits y vectoriales, diseñar efectos Web, como rollovers y menús emergentes, recortar y optimizar elementos gráficos para reducir su tamaño de archivo y automatizar tareas repetitivas para ahorrar tiempo.

Microsoft Word 2007



Editor de texto que se ha utilizado para la creación de la documentación del sistema.

ArgoUML



Herramienta de diagramación UML usada para crear modelos y exportarlos en formato XMI 1.2. que será una posible entrada de la herramienta GiJava.

6. Metodología

El desarrollo del sistema se ha llevado a cabo siguiendo una metodología orientada a objetos que permite modelar las distintas fases de su desarrollo. La metodología seleccionada es RUP, Rational Unified Process, porque utiliza UML como lenguaje de modelado, está guiado por casos de uso, se centra en la arquitectura y es iterativo e incremental, de modo que el resultado de cada iteración es un sistema que puede ser probado, integrado y ejecutado.

Siguiendo la metodología RUP, el desarrollo del sistema se ha realizado en varias iteraciones. En cada una de ellas se ha centrado la atención en una fase específica de desarrollo del sistema: análisis, diseño, implementación y pruebas; obteniendo en cada iteración un producto ejecutable, que ha sido modificado en iteraciones posteriores.

La [fase de análisis](#) ha sido la primera toma de contacto con el desarrollo del sistema, por lo que en ella se ha realizado un estudio global del mismo y sus funcionalidades.

En esta fase se han llevado a cabo los siguientes pasos:

- Análisis de los requisitos del sistema que identifiquen las funcionalidades.
- Diagrama de casos de uso a partir de la información obtenida en el punto anterior.
- Breve descripción textual de cada uno de los casos de uso y los correspondientes diagramas de secuencia conceptuales.

En la [fase de diseño](#) se han detallado las funcionalidades del sistema, representadas por los casos de uso. Los pasos realizados para la fase de diseño son:

- Se han estudiado con más detalle los casos de uso del análisis, refinando los diagramas de secuencia.
- Se han detallado las clases necesarias para el sistema y se realizan los diagramas de clases parciales.

Una vez terminado el diseño y tomando como base el resultado del estudio de las fases anteriores comienza la [fase de implementación](#).

Para esta fase se realizó lo siguiente:

- Se han diseñado la arquitectura del sistema. Por lo tanto, se han realizado el diagrama de componentes y el diagrama de despliegue.
- Se llevó a cabo la implementación del sistema.

La última fase del desarrollo del sistema es la [fase de pruebas](#). En ella se comprueba que se cumplen criterios de corrección y calidad, es fundamental para garantizar el buen funcionamiento y la calidad del sistema que está siendo desarrollado. Se han realizado pruebas del sistema y pruebas de caja negra de todos los formularios.
lenguaje

7. PLANIFICACIÓN

En este apartado se muestra la planificación temporal realizada al inicio del proyecto así como la duración real que ha llevado realizarlo, dividiendo el proyecto en distintas actividades e indicando para cada una de ellas las duraciones estimada y real. También se da una explicación de las causas que provocaron el desfase entre ambas.

La documentación se ha realizado de modo secuencial, dividiéndola en dos partes: análisis y diseño, realizando cada una de ellas al acabar las diferentes etapas del análisis y diseño respectivamente.

7.1 Planificación estimada

La dedicación estimada es de 6 horas al día.

	Nombre de tarea	Duración	Comienzo	Fin	Pred
1	<input type="checkbox"/> Proyecto	106 días	lun 06/10/08	lun 02/03/09	
2	<input type="checkbox"/> Análisis y Diseño	33 días	lun 06/10/08	mié 19/11/08	
3	Estudio de las tecnologías y herramientas	5 días	lun 06/10/08	vie 10/10/08	
4	Estudio del problema	4 días	lun 13/10/08	jue 16/10/08	3
5	Análisis de requerimientos	10 días	vie 17/10/08	jue 30/10/08	4
6	Diseño del sistema	7 días	vie 31/10/08	lun 10/11/08	5
7	Diseño de la interfaz del sistema	7 días	mar 11/11/08	mié 19/11/08	6
8	<input type="checkbox"/> Implementación	60 días	jue 20/11/08	mié 11/02/09	
9	Codificación del sistema	40 días	jue 20/11/08	mié 14/01/09	7
10	Implementación de la interfaz	20 días	jue 15/01/09	mié 11/02/09	9
11	<input type="checkbox"/> Pruebas	13 días	jue 12/02/09	lun 02/03/09	
12	Pruebas de codificación	5 días	jue 12/02/09	mié 18/02/09	10
13	Pruebas globales	8 días	jue 19/02/09	lun 02/03/09	12
14	<input type="checkbox"/> Documentación	94 días	vie 17/10/08	mié 25/02/09	
15	Documentación de análisis	10 días	vie 17/10/08	jue 30/10/08	4
16	Documentación de diseño	8 días	vie 31/10/08	mar 11/11/08	5
17	Documentación de la implementación	5 días	jue 15/01/09	mié 21/01/09	9
18	Documentación de las pruebas	5 días	jue 19/02/09	mié 25/02/09	12
19	Manual de usuario	4 días	jue 19/02/09	mar 24/02/09	12

Ilustración 15: Tabla de tareas y su estimación temporal – Planificación estimada

A partir de la tabla anterior, se realiza el siguiente Diagrama de Gantt que ayuda a entender e interpretar la secuencia seguida en la realización de las tareas.

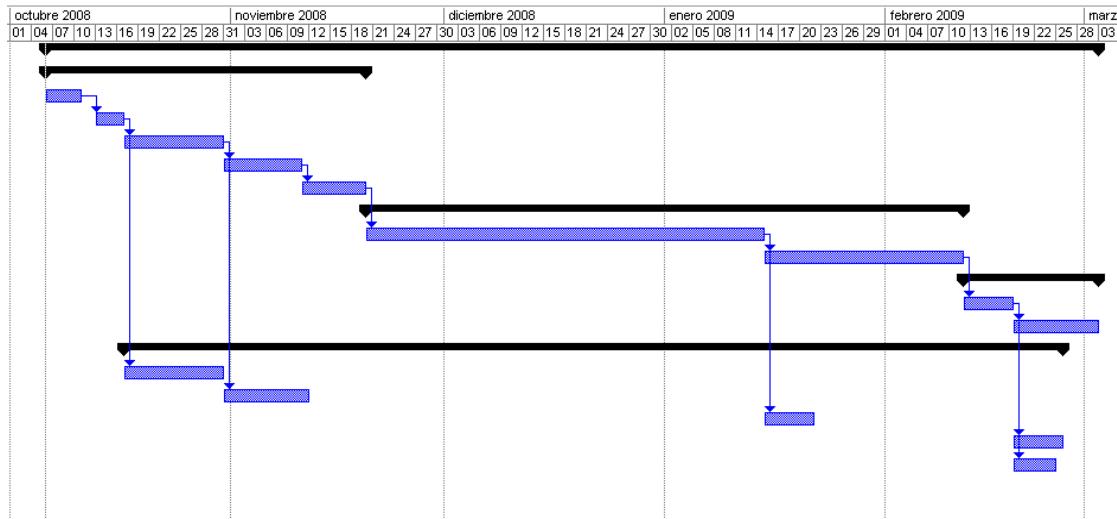


Ilustración 16: Diagrama de Gannt - Planificación estimada

7.2 Duración real

La planificación real para este proyecto, ha sufrido cambios con respecto a la estimación inicial realizada para el mismo. Estos cambios son fundamentalmente en el tiempo de realización de las distintas tareas.

A continuación se muestra una tabla con las distintas tareas, y el tiempo de realización de cada una de ellas.

	Nombre de tarea	Duración	Comienzo	Fin	Predec
1	■ Proyecto	216 días	mié 12/11/08	mié 09/09/09	
2	■ Análisis y Diseño	54 días	mié 12/11/08	lun 26/01/09	
3	Estudio de las tecnologías y herramientas	11 días	mié 12/11/08	mié 26/11/08	
4	Estudio del problema	10 días	jue 27/11/08	mié 10/12/08	3
5	Análisis de requerimientos	10 días	jue 11/12/08	mié 24/12/08	4
6	Diseño del sistema	12 días	jue 25/12/08	vie 09/01/09	5
7	Diseño de la interfaz del sistema	11 días	lun 12/01/09	lun 26/01/09	6
8	■ Implementación	134 días	mar 27/01/09	vie 31/07/09	
9	Codificación del sistema	107 días	mar 27/01/09	mié 24/06/09	7
10	Implementación de la interfaz	27 días	jue 25/06/09	vie 31/07/09	9
11	■ Pruebas	21 días	lun 03/08/09	lun 31/08/09	
12	Pruebas de codificación	9 días	lun 03/08/09	jue 13/08/09	10
13	Pruebas globales	12 días	vie 14/08/09	lun 31/08/09	12
14	■ Documentación	195 días	jue 11/12/08	mié 09/09/09	
15	Documentación de análisis	12 días	jue 11/12/08	vie 26/12/08	4
16	Documentación de diseño	12 días	jue 25/12/08	vie 09/01/09	5
17	Documentación de la implementación	9 días	jue 25/06/09	mar 07/07/09	9
18	Documentación de las pruebas	7 días	mar 01/09/09	mié 09/09/09	12;13
19	Manual de usuario	6 días	mar 01/09/09	mar 08/09/09	12;13

Ilustración 17: Tabla de tareas y su estimación temporal – Planificación real

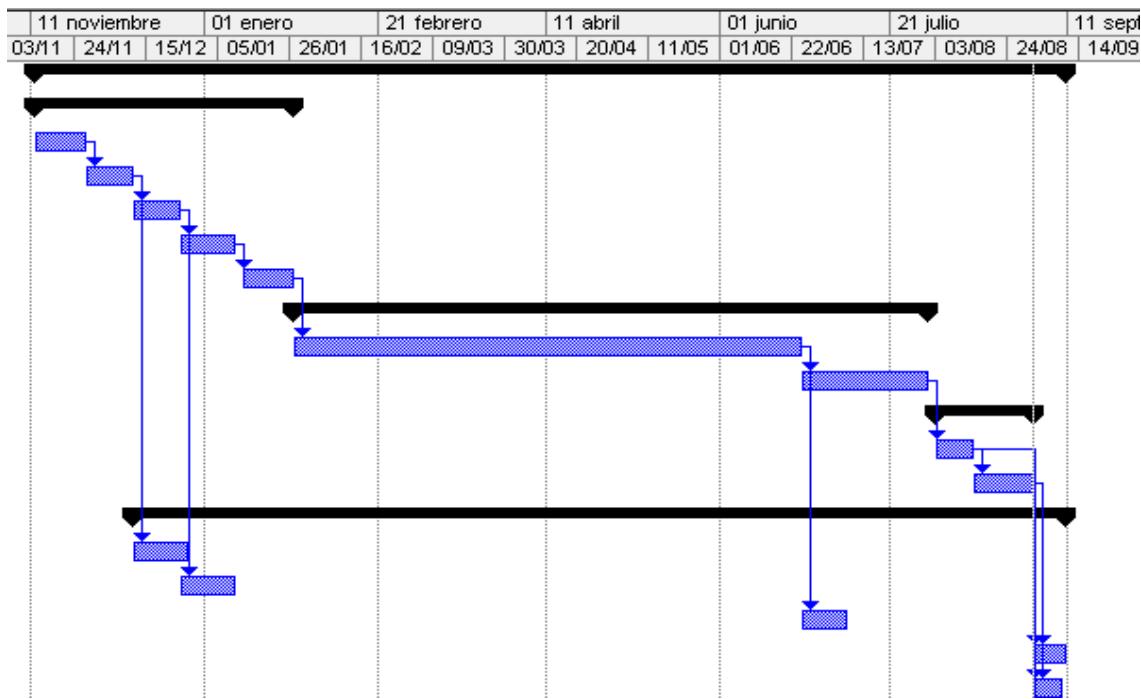


Ilustración 18: Diagrama de Gannt - Planificación real

7.3 Conclusiones de la planificación

Durante todo el desarrollo han surgido diversas causas por las cuales todas las etapas se han alargado más de lo previsto, además de una planificación no muy ajustada al caso real. A la vista, tanto de la estimación temporal como de la estimación real, resulta evidente que el proyecto no se realizó consecuentemente con la estimación del tiempo que se había planificado. Los motivos se detallan a continuación:

- Debido a actividades de formación complementaria entre los meses de Enero de 2009 y Junio de 2009, ha limitado la cantidad de tiempo dedicado al proyecto en un 50%. Por lo que si en la planificación estimada se consideró que se trabajarían 6 horas al día, en la real fueron 3 horas al día.
- Estudio de las tecnologías utilizadas durante el proceso de implementación (XML, XMI, HTML y las librerías Java usadas para su manejo como son JDOM, JGRAPH, DOM o las usadas para el desarrollo de interfaces en Java como es SWING). El desconocimiento previo de las mismas ha supuesto una incorrecta valoración de la complejidad del proyecto a desarrollar.
- La estimación temporal realizada en el anteproyecto había sido realizada sin conocer en profundidad detalles de bajo nivel.
- Otra causa que ha ocasionado la demora han sido los cambios que se han producido en la interfaz de la herramienta Java, dividiendo la herramienta en dos bloques (Parte de Diseño y parte del IDE de Java).

- Las técnicas de verificación y prueba. El hallazgo de errores de implementación ha prolongado temporalmente el desarrollo del proyecto

8. PRESUPUESTO

En este apartado se indicará el presupuesto final del proyecto. Indicando los costes del mismo debidos a recursos humanos, así como los derivados de los dispositivos físicos y programas utilizados durante el desarrollo del mismo.

8.1 Coste de software

A continuación se muestra el coste del hardware y el software utilizados en el desarrollo del sistema. Pero sólo un porcentaje de estos costes serán incluidos en el presupuesto final. Esto es debido a que los recursos físicos no van incluidos en el sistema, y por lo tanto pueden ser amortizados en varios años.

Se ha realizado una amortización a 4 años del material hardware y software.

Software de desarrollo	Coste (€)
Microsoft Windows XP Professional	462.69
Borland JBuilder X Enterprise Edition	1499.00
Adobe Fireworks CS4	208.00
Visual Paradigm Profesional Edition	838.56
Microsoft Project 2007	275.96
Microsoft Office 2007	582.06
Coste Total Software	3866,27
Coste Total Software amortizado	571,99

8.2 Coste de hardware

Hardware	Coste (€)
PC, Intel Pentium, 1Gb RAM	900€
Impresora Epson Scan	70 €
Coste Total Hardware	970 €
Coste Total Hardware amortizado	143,50 €

8.3 Coste de recursos humanos

El coste de los Recursos Humanos depende de la labor desempeñada, por ello, en la siguiente tabla se muestran las horas necesarias para cada tarea, así como el coste de cada una. En la última columna se muestra el coste total de cada tarea, y el coste total de los recursos humanos.

El tiempo estimado de dedicación a cada una de las tareas aparece detallado en el diagrama de Gantt.

Recursos Humanos	Horas	Coste/hora	Coste Total (€)
Investigación	63	18 €	1134 €
Análisis	30	22 €	660 €
Diseño	69	22 €	1518 €
Implementación	402	18 €	7236 €
Pruebas	73	18 €	1314 €
Documentación	138	18 €	2484 €
Coste Total Recursos Humanos			14346 €

8.4 Coste total

Recurso	Coste Total (€)
Hardware	143,50 €
Software	571,99 €
Recursos Humanos	14346 €
Coste Total del Sistema	15061,49 €

9. CONCLUSIONES

2.1 Problemas encontrados

A continuación se enumeran algunas de las dificultades que han ido surgiendo durante la realización del proyecto:

- Como principal dificultad destacar el aprendizaje de las tecnologías usadas (XMI, XML, HTML y las librerías Java) para sacar el mayor rendimiento de las mismas así como la realización de pruebas y la complicada localización de los errores.
- La flexibilidad que proporciona el lenguaje de modelado UML para el análisis y diseño. Al no tratarse de una metodología, dota al desarrollador de una gran libertad que ha originado dudas al analizar y diseñar la aplicación.
- El diseño de una interfaz sencilla, intuitiva y manejable, para que el usuario final se sienta cómodo usando la aplicación.
- La poca familiaridad con el tema de la generación directa de código. Y la escasez de documentación al respecto.

2.2 Posibles ampliaciones

Este sistema ha sido realizado con unos objetivos que fueron expuestos anteriormente, y aunque éstos fueron cumplidos, existen otros que cabe plantearse una vez realizado el sistema.

Por ello se expone a continuación, una especie de plan de ampliaciones en el que se muestran las más interesantes:

- La aplicación únicamente genera código fuente en Java. Una posible ampliación es generar código para otros lenguajes o traducir código fuente de un lenguaje a otro.
- La aplicación es restrictiva en cuanto a la versión de XMI que se utiliza. El formato XMI cambia considerablemente de la versión 1.2 y 1.3 que usan MOF (Meta-Object Facility) 1.4 a la versión 2.1 que usa MOF 2.0. La herramienta solo lee xmi versión 1.2 que es la versión actual en la que se exportan archivos de ArgoUML. Una ampliación sería adaptar la aplicación para que pueda usar archivos exportados de otras herramientas de diagramación UML que soportan versiones superiores de XMI como Enterprise architect o visual paradigm que además de soportar xmi versión 1.1 y 1.2 exportan xmi 2.1.
- Se puede mejorar la aplicación para que reconozca paquetes que puedan existir en el diagrama de clases inicial.
- Adaptar la aplicación para que pueda ser un plugin de ArgoUML.
- Dar la posibilidad al usuario de escoger un conjunto de iconos para personalizar más el diseño de la interfaz de la aplicación generada.

2.3 Conclusiones

La realización de este proyecto ha sido un aprendizaje continuo de conocimientos sobre Java, XML, XMI, HTML y las APIs de Java utilizadas JDOM, JGRAPH, DOM o las usadas para el desarrollo de interfaces en Java como SWING.

Muchas de estas tecnologías resultaban desconocidas totalmente o conocidas sólo en un aspecto muy general, por lo que el tiempo empleado para profundizar en su aprendizaje fue muy notable. Además, los problemas no sólo surgieron al principio del planteamiento del proyecto, sino que continuaron durante su desarrollo. Debido a la inexperiencia en proyectos de esta envergadura no se ha tenido en cuenta suficientemente el tiempo que se debía invertir en este aprendizaje. Es por ello que la planificación estimada inicial difiere en gran medida del tiempo final empleado para la realización de este proyecto.

Con relación a la arquitectura de modelado de datos se ha sacado como conclusión que el uso de MDD es una opción muy prometedora para el desarrollo de software porque se centra en el modelo aunque no permiten generar el 100% código automáticamente y las transformaciones entre diferentes modelos son difíciles de definir.

Se ha podido comprobar que es de gran importancia la realización de un buen análisis y diseño a la hora de realizar un proyecto, ya que esto facilita en gran medida el trabajo. Y anterior a estos el análisis de los requerimientos, sobre todo los exigidos por el cliente ya que me encontraba ante un mundo totalmente desconocido para mí y era esencial saber cuáles eran los objetivos que se debían cumplir para que el proyecto no distara mucho de la idea inicial.

Durante la implementación se han separado en diferentes paquetes la parte del diseño gráfico de la interfaz de la aplicación y la lógica de la aplicación, permitiendo que si se desea en un futuro se pueda cambiar completamente la interfaz, sin necesidad de cambiar excesivamente la lógica del sistema. Destacar que el entorno de desarrollo Borland Builder X ha sido de gran ayuda en el diseño de interfaces ya que incluye un entorno de herramientas de drag-and-drop desde la sección de componentes de la barra de menú, lo que permite construir las interfaces con mayor rapidez. También son interesantes las ayudas que ofrece Jbuilder en cuanto a navegabilidad en el código y al buscar nombres de clases o métodos o al encontrar errores en tiempo real.

La fase de pruebas, puso de manifiesto la importancia de las mismas para el desarrollo de cualquier sistema, debido a la gran cantidad de opciones que se encuentran en el sistema y la cantidad de datos a validar en los distintos formularios. Otra conclusión importante extraída de la realización de las pruebas es que un buen diseño las mismas facilita su elaboración y eficiencia.

En relación al uso de XMI cabe destacar que es una forma sencilla de empaquetar información y que existe una gran cantidad de herramientas que usan XMI como eclipse UML, altova, poseidon, netbeans,visual paradigm,rational rose... así un único formato de archivo se usa para muchas herramientas case permitiendo el intercambio de objetos entre ellas y también la reutilización de los mismos.

La principal desventaja del uso de XMI es la incompatibilidad entre diferentes versiones. Dependiendo de las herramientas que intercambien el formato XMI debemos de tener en cuenta las versiones. Por ejemplo ArgoUML no puede leer archivos XMI que contengan modelos UML 1.5 o UML 2.0, solo lee archivos UML 1.4(XMI 1.2) y UML 1.3(XMI 1.0).

BLOQUE II: MANUAL TÉCNICO

1. ESPECIFICACIÓN DE REQUISITOS

En este apartado se explicarán de forma general las necesidades del sistema a desarrollar, obteniendo las funciones que realizan los usuarios en el sistema, así como las funciones adicionales del sistema. El resultado de esta fase puede tomarse como base para negociar las condiciones de realización del proyecto con el cliente, debiendo ser toda la documentación obtenida en esta fase sencilla y comprensible por cualquier persona.

1.1 Objetivos principales

GIA Java es un generador de código, con capacidad de producir código fuente en java a partir de un modelo orientado a objetos escrito en UML. El sistema se divide en dos partes la parte de Diseño y la parte de IDE de java por lo que también dividiremos los objetivos en dos partes:

Parte de Diseño:

- Creación de un modelo de datos sobre el que trabajar.
- Importación de archivos XMI creados con ArgoUML.
- Edición de modelos UML (Gestión de clases, atributos, métodos...)
- Generación de código (Generación de clases completas, con métodos get/set de los atributos y métodos de administración XML)
- Generación de una interfaz de usuario.
- Sistema de guardado automático de los ficheros generados en el espacio de trabajo seleccionado.

Parte de IDE de Java:

- Edición de los archivos de código fuente java generados, se pueden crear nuevos archivos y borrar los existentes.
- El entorno de desarrollo permitirá compilar, depurar y ejecutar el código generado y/o editado.
- Desde el IDE de Java se podrá generar automáticamente una documentación Javadoc de todas las clases, métodos y atributos generados.

1.2 Objetivos adicionales

- Facilidad de uso: Se pretende que la aplicación sea fácil de usar. Todas las opciones son accesibles desde el menú principal dividido en secciones. Las opciones que más se usan aparecen en una barra de botones con iconos simples, tanto en la parte de Diseño como en la parte de IDE de Java

El sistema tiene una sección de ayuda en la que se les intenta resolver a los usuarios las dudas que le surjan al realizar cualquier función en el sistema.

- Entorno: Uno de los objetivos principales del sistema es disponer de un entorno sencillo, amigable, intuitivo y rápido para proporcionar a los usuarios comodidad y navegabilidad. Tanto la interfaz del sistema como la interfaz generada para las aplicaciones Java intentan cumplir estos objetivos.
- Fácil comprensión del código generado: El código generado de las clases es código perfectamente comprensible por un programador, existen métodos de lectura, búsqueda y escritura en XML que pueden resultar un poco más complejos para un usuario que no haya trabajado antes con XML, JDOM y Java, por lo que se aconseja al usuario la lectura previa de algún manual en el que se traten estos temas. En la sección de bibliografía en el apartado Java podemos ver documentación al respecto.

1.3 Características de usuario

La aplicación va dirigida a desarrolladores de software que necesitan crear sistemas de información en plazos cortos y que ayudados por GIA Java aceleran el proceso de implementación generando gran parte del código fuente de un sistema orientado a objetos.

2. ANÁLISIS

Una vez identificadas las funcionalidades básicas del sistema en la fase de análisis de requisitos se procede a analizarlas de forma más concisa en la fase de análisis con el fin de obtener una arquitectura del sistema capaz de resolver el sistema en condiciones ideales. Para ello se usan las herramientas que ofrece UML para esta fase del desarrollo y que están muy relacionadas con la identificación de casos de uso, acciones que representan requisitos funcionales del sistema y que se pueden descomponer en una secuencia de interacciones entre un usuario y el sistema.

2.1 Diagrama de casos de uso

Un caso de uso es un escenario que describe como el software va a ser utilizado en una determinada situación. Una vez recopilados los requisitos, ha sido necesario crear un conjunto de escenarios que identificaran una línea de utilización para el sistema construido. Los casos de uso forman parte del análisis y nos han ayudado a representar qué hace el sistema desde el punto de vista del usuario, sirven para reflejar su comportamiento, aspectos dinámicos, funcionalidades, los actores y límites.

Es decir, describen un uso del sistema y cómo este interactúa con el usuario.

A continuación se muestran dos diagramas de casos de uso, uno para la parte de Diseño y otro para la parte de IDE de Java.

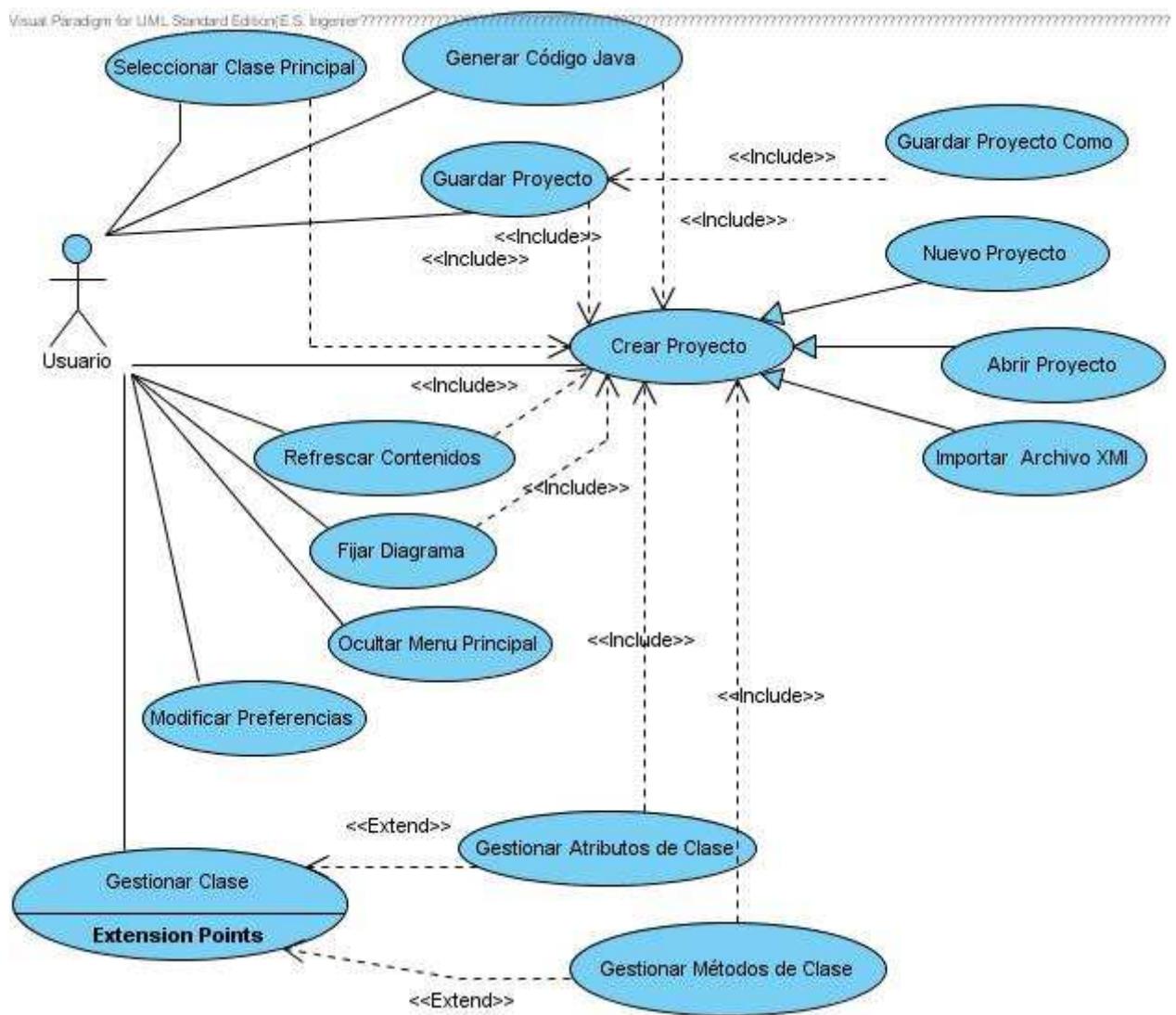


Ilustración 19: Diagrama Casos de uso Diseño

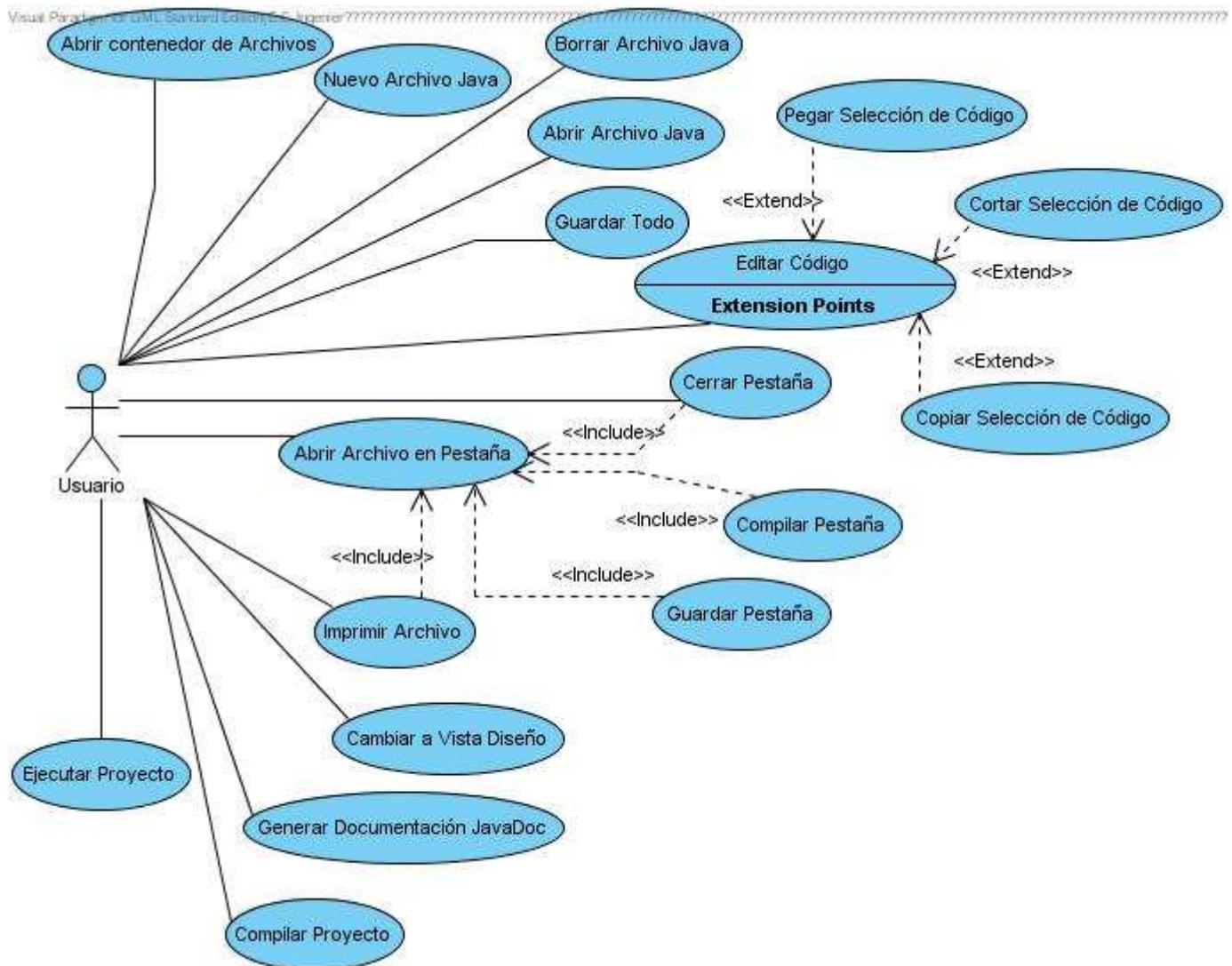


Ilustración 20: Diagrama casos de uso IDE de Java

Como se puede ver en los diagramas sólo existe un usuario que es el encargado de realizar todas las funciones del sistema. Todas las tareas que realiza el usuario en la parte de IDE de Java se realizan habiendo generado el código previamente en la parte de Diseño.

2.2 Descripción de los casos de uso

PARTE DE DISEÑO DE CLASES

2.2.1 Caso de uso: Nuevo Proyecto

Nombre	Nuevo Proyecto
Objetivo	Crear un nuevo proyecto en blanco sobre el que trabajar.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema.
Pos condiciones	Existirá un nuevo proyecto sobre el que trabajar, en el que se podrá crear un nuevo diagrama de clases, para el problema a resolver.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario accede al menú Archivo - Nuevo Proyecto. 2. El sistema muestra un formulario para introducir el nombre que se le quiere dar al proyecto, que será el nombre que va a tomar el archivo Zip en el que se almacenarán los ficheros XML, relativos a las clases que se añadan al diagrama de clases del sistema. En el formulario también se introduce la ruta en la que se almacenarán el archivo ZIP a crear. 3. El usuario da un nombre al proyecto e introduce manualmente o selecciona la ruta donde se almacenarán los ficheros. 4. El sistema valida que se ha completado el formulario y si la ruta no existe la crea. 5. El sistema inicializa las estructuras internas en la que se almacenarán el modelo: Donde se almacenan las clases (ClassContainer), las asociaciones entre las clases en la (AssociationTable), y las relaciones de herencia en la (GeneralizationTable) para poder utilizarlos.
Flujo de eventos excepcional	4a. Si no se han llenado todos los campos el sistema muestra un mensaje de error y se vuelve al paso 2.
Frecuencia de uso	Muy Alta

2.2.2 Caso de uso: Abrir Proyecto.

Nombre	Abrir Proyecto
Objetivo	Abrir un nuevo proyecto sobre el que trabajar, que ya ha sido creado por nuestra herramienta con anterioridad. Se parte de un conjunto de archivos XML comprimidos en un archivo Zip que tienen el formato que nuestra herramienta entiende.
Actores	Usuario

Precondiciones	Que la herramienta esté ejecutándose en el sistema Que exista un proyecto Zip que ya ha sido creado por la herramienta, o que tenga el mismo formato.
Pos condiciones	Los ficheros estarán disponibles para que la herramienta realice su tratamiento.
Flujo de eventos normal	<p>6. El usuario accede al menú Archivo - Abrir Proyecto.</p> <p>7. El sistema muestra un formulario para introducir la ruta del archivo ZIP a cargar, permitiendo acceder al archivo además mediante un menú de exploración de carpetas, filtrado por la extensión *.Zip.</p> <p>8. El usuario introduce manualmente o selecciona la ruta del fichero.</p> <p>9. El sistema valida las rutas.</p> <p>10. El sistema comprueba que los ficheros estén bien formados y valida los elementos contenidos en el fichero.</p> <p>11. El sistema almacena los datos en las estructuras internas de la herramienta: las clases en el ClassContainer, las asociaciones entre las clases en la AssociationTable y las relaciones de herencia en la GeneralizationTable para poder utilizarlos a posteriori.</p>
Flujo de eventos excepcional	<p>4a. Si la ruta introducida no es válida o que el formato de los ficheros no es correcto. El sistema mostrará un mensaje de error y solicitará que se introduzca de nuevo.</p> <p>5a. Si los ficheros no están bien formados o incluyen elementos que no se ajustan al formato interno de la herramienta el sistema muestra un mensaje de error.</p>
Frecuencia de uso	Muy Alta

2.2.3 Caso de uso: Importar Archivo XMI

Nombre	Importar Archivo XMI
Objetivo	Crear un nuevo proyecto sobre el que trabajar y que el archivo XMI de partida generado por una herramienta de diagramación UML, esté disponible para la herramienta.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema Que exista un archivo XMI que cargar.
Pos condiciones	Los ficheros estarán disponibles para que la herramienta realice su tratamiento.

Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario accede al menú de Archivo – Importar Archivo XMI. 2. El sistema muestra un formulario para introducir la ruta del archivo XMI a cargar, permitiendo acceder al archivo además mediante un menú de exploración de carpetas, filtrado por la extensión *.xmi. 3. El usuario introduce manualmente o selecciona la ruta del fichero. 4. El sistema valida la ruta 5. El sistema comprueba que el fichero esté bien formado y valida los elementos contenidos en el fichero. 6. El sistema almacena los datos en la estructura que contiene las clases ClassContainer, las asociaciones entre las clases en la AssociationTable y las relaciones de herencia en la GeneralizationTable. 7. El sistema muestra los datos del fichero XMI en el escritorio de trabajo y los datos de las clases son representados mediante un árbol.
Flujo de eventos excepcional	<ol style="list-style-type: none"> 4a. Si la ruta introducida no es válida o que el formato de los ficheros no es el correcto. El sistema mostrará un mensaje de error y solicitará que se introduzcan de nuevo. 5a. Si el fichero XMI no está bien formado o incluye elementos que no se ajustan al formato interno de la herramienta, el sistema muestra un mensaje de error.

Frecuencia de uso Muy Alta

2.2.4 Caso de uso: Generar Código Java

Nombre	Generar Código Java
Objetivo	Disponer de un directorio de trabajo que sirva de almacén de los archivos que genere la herramienta. Generar los archivos *.java y permitir que estos estén disponibles para la herramienta en la parte IDE de Java.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado o abierto un proyecto. Que se haya seleccionado la clase principal del sistema.
Pos condiciones	Se generan los archivos *.java de cada clase, la interfaz de usuario y las clases para gestionar los datos de las clases.

Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario accede al menú Herramientas - Generar código. 2. El sistema comprueba que existe un proyecto cargado en la herramienta sobre el cual trabajar. 3. El sistema comprueba que se ha marcado la clase principal del sistema. 4. El sistema pregunta al usuario si se desean guardar los cambios antes de generar el código. 5. El usuario decide guardar el proyecto y se ejecuta el caso de uso guardar o guardar como. 6. El sistema muestra un formulario para introducir una ruta que servirá como directorio de trabajo y en donde se almacenarán los archivos generados. 7. El usuario introduce manualmente o selecciona la ruta del fichero mediante un menú de exploración de carpetas. 8. El sistema comprueba que se ha introducido una ruta y que es válida. 9. El sistema genera los archivos de código y los guarda en el espacio de trabajo. 10. El sistema actualiza el árbol que contiene los archivos generados por la aplicación.
Flujo de eventos excepcional	<ol style="list-style-type: none"> 2a. Si no se ha cargado ningún proyecto el sistema muestra un mensaje de error. 3a. Si no se ha marcado la clase principal del sistema se muestra un mensaje de error y se abre la ventana de selección de clase principal. 5a. Si el usuario decide no guardar los cambios se pasa directamente al paso 6. 8a. Si la ruta introducida no existe el sistema crea los directorios necesarios.
Frecuencia de uso	Muy Alta

2.2.5 Caso de uso: Modificar Preferencias

Nombre	Modificar Preferencias
Objetivo	Cambiar los directorios por defecto que existen en la aplicación. Se puede cambiar el directorio en el que se almacenan los archivos java y el directorio en el que se almacena el proyecto al ser guardado.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema
Pos condiciones	Cuando se pulse el botón guardar o cuando se pulse el botón generar código, tanto los archivos como el código será guardados o generados en los directorios por defecto en rellenados en Modificar Preferencias.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario accede al menú de Herramientas – Preferencias. 2. El sistema muestra una ventana que contiene un formulario para introducir las rutas. 3. El usuario introduce manualmente o selecciona las rutas del fichero. 4. El sistema almacena los datos de las rutas en memoria.

Frecuencia de uso Normal.

2.2.6 Caso de uso: Seleccionar Clase Principal

Nombre	Seleccionar Clase Principal
Objetivo	Marcar una clase como principal del proyecto, y sirve como origen para generar la interfaz gráfica.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado, abierto o importado un proyecto.
Pos condiciones	Se ha marcado la clase principal
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario accede al menú Herramientas - Seleccionar Clase Principal. 2. El sistema comprueba que existe un proyecto cargado en la herramienta sobre el cual trabajar. 3. El sistema muestra una ventana desde la que se permite hacer una selección de entre todas las clases de la aplicación. 4. El usuario selecciona una clase como principal. 5. El sistema almacena la clase en memoria. 6. El sistema muestra un mensaje informativo.
Flujo de eventos excepcional	2a. Si no se ha cargado ningún proyecto el sistema muestra un mensaje de error.
Frecuencia de uso	Muy Alta

2.2.7 Caso de uso: Guardar Proyecto

Nombre	Guardar Proyecto
Objetivo	Salvar los cambios realizados en el proyecto para tener la última versión del proyecto creado.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado, abierto o importado un proyecto. Que se haya guardado como previamente el proyecto en disco.
Pos condiciones	Se han guardado los cambios del proyecto en formato propio de la herramienta. Se ha generado un archivo JPG con el diagrama de clases.

Flujo de eventos normal	1. El usuario accede al menú Archivo - Guardar. 2. El sistema comprueba que existe un proyecto cargado en la herramienta sobre el cual trabajar. 3. El sistema almacena el proyecto en el directorio en el que se ha almacenado previamente mediante la opción guardar como.
Flujo de eventos excepcional	2a. Si no se ha cargado ningún proyecto el sistema muestra un mensaje de error.
Frecuencia de uso	Muy Alta

2.2.8 Caso de uso: Guardar Proyecto Como

Nombre	Guardar Proyecto Como
Objetivo	Tener una copia del proyecto creado o modificado en disco.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado, abierto o importado un proyecto.
Pos condiciones	Se ha almacenado en formato propio de la herramienta un proyecto en disco. Se ha generado un archivo JPG con el diagrama de clases.
Flujo de eventos normal	1. El usuario accede al menú Archivo – Guardar Como... 2. El sistema comprueba que existe un proyecto cargado en la herramienta sobre el cual trabajar. 3. El sistema muestra una ventana que contiene un formulario en el que aparecen dos campos uno para introducir el nombre que se le quiere dar al proyecto y otro para introducir la ruta en la que se quiere almacenar. Por defecto este campo aparece relleno con la ruta que se haya introducido desde la ventana de preferencias. 4. El usuario completa los datos del formulario. 5. El sistema comprueba que se le ha dado un nombre al proyecto y que es un nombre válido. 6. El sistema si la ruta introducida no existe crea los directorios que sean necesarios para poder almacenar los archivos en la ruta.
Flujo de eventos excepcional	2a. Si no se ha cargado ningún proyecto el sistema muestra un mensaje de error. 5a. Si no se ha introducido ningún nombre para el proyecto el sistema muestra un mensaje de error y se vuelve al paso 3. 5b. Si el nombre del proyecto contiene caracteres inválidos se muestra un mensaje de error y se vuelve al paso 3.
Frecuencia de uso	Muy Alta

2.2.9 Caso de uso: Refrescar contenidos

Nombre	Refrescar Contenidos.
Objetivo	Conseguir que el árbol lateral izquierdo que contiene todos los datos relativos a las clases del proyecto a crear esté actualizado. Y refrescar el contenido del panel del diagrama de clases.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema Que se haya creado, abierto o importado un proyecto en la herramienta.
Pos condiciones	Los datos relativos a las clases del sistema están actualizados. El diagrama de clases se vuelve a pintar.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario accede al menú de Herramientas - Refrescar contenidos. 2. El sistema comprueba que existe un proyecto cargado en la herramienta sobre el cual trabajar. 3. El sistema volverá a pintar el árbol de contenidos y también el diagrama de clases.
Flujo de eventos excepcional	2a. Si no se ha cargado ningún proyecto no hay datos que refrescar, por lo que el sistema mostrará un mensaje de error.
Frecuencia de uso	Normal

2.2.10 Caso de uso: Fijar Diagrama

Nombre	Fijar Diagrama.
Objetivo	Mantener el diagrama de clases no editable por el usuario. El sistema si lo podrá editar al añadir, eliminar o modificar cualquier elemento del mismo. Cuando el usuario pulse el botón Refrescar contenidos o Herramientas - Diagrama editable el diagrama podrá volver a editarse.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado, abierto o importado un proyecto en la herramienta.
Pos condiciones	El usuario no podrá mover y colocar los elementos del diagrama.

Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario accede al menú de Edición - Fijar diagrama. 2. El sistema comprueba que existe un proyecto cargado en la herramienta sobre el cual trabajar. 3. El sistema cambia a no editable el diagrama y cambia la opción de menú Herramientas – Fijar diagrama por Herramientas – Diagrama Editable, para que el usuario decida cuando quiere volver a modificar la posición de los elementos del diagrama.
Flujo de eventos excepcional	<p>2a. Si no se ha cargado ningún proyecto no hay datos que refrescar, por lo que el sistema mostrará un mensaje de error.</p>
Frecuencia de uso	Normal

2.2.11 Caso de uso: Ocultar Menú Principal

Nombre	Ocultar Menú Principal
Objetivo	Hacer la zona de trabajo más amplia, capturando el espacio que ocupa el menú principal.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema.
Pos condiciones	El menú principal está oculto.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario accede al menú de Herramientas - Ocultar Menú Principal. 2. El sistema oculta el menú.
Frecuencia de uso	Poca

2.2.12 Caso de uso: Borrar todos los elementos

Nombre	Borrar todos los elementos
Objetivo	Disponer de un proyecto en blanco sobre el que trabajar.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado, importado o abierto un proyecto en la herramienta.
Pos condiciones	El árbol no tiene ningún elemento y el diagrama de clases está vacío.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario hace clic en borrar todos los elementos. 2. El sistema comprueba que se ha cargado un proyecto. 3. El sistema borra todos los elementos del modelo de datos.
Flujo de eventos excepcional	<p>2a. Si no se ha cargado ningún proyecto sobre el que trabajar el sistema mostrará un error y se vuelve al paso 1.</p>

Frecuencia de uso Poca

2.2.13 Caso de uso: Gestionar Clase

Escenario: Añadir Clase

Nombre	Añadir Clase.
Objetivo	Agregar al modelo de datos de la herramienta una nueva clase.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado, abierto o importado un proyecto en la herramienta.
Pos condiciones	El usuario podrá editar los datos (atributos y métodos) de una nueva clase.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario selecciona el elemento raíz del árbol de clases denominado “Contenedor de clases” 2. El usuario pulsa el botón “Añadir elemento al árbol”. 3. El sistema comprueba que existe un proyecto cargado en la herramienta sobre el cual trabajar. 4. El sistema abre un nuevo formulario para introducir los datos de la nueva clase a añadir. 5. El usuario introduce los datos. 6. El sistema comprueba que se han introducido los datos correctamente. 7. El sistema almacena la nueva clase en el contenedor de clases y refresca el contenido del diagrama de clases añadiéndole la nueva clase.
Flujo de eventos excepcional	<ol style="list-style-type: none"> 3a. Si no se ha cargado ningún proyecto sobre el que trabajar el sistema mostrará un error. 6a. Si no se rellenan todos los datos del formulario el sistema mostrará un error.

Frecuencia de uso Alta

Escenario: Borrar Clase

Nombre	Borrar Clase.
Objetivo	Eliminar del modelo de datos de la herramienta una clase.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado, abierto o importado un proyecto en la herramienta.
Pos condiciones	Se ha eliminado la clase del sistema.

Flujo de eventos normal	1. El usuario selecciona una clase del árbol de clases cuyo elemento raíz es el denominado “Contenedor de clases” 2. El usuario pulsa el botón “Borrar del árbol el elemento seleccionado”. 3. El sistema abre una nueva ventana en la que pide al usuario confirmación para borrar la clase y todos sus elementos. 4. El usuario acepta el mensaje de confirmación. 5. El sistema borra la clase del contenedor de clases y refresca el contenido del diagrama de clases.
Flujo de eventos excepcional	4a. Si el usuario pulsa el botón cancelar el caso de uso finaliza.
Frecuencia de uso	Alta

Escenario: Modificar Clase

Nombre	Modificar Clase.
Objetivo	Modificar alguna de las características propias de una clase que se verán reflejadas a la hora de generar el código.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado, abierto o importado un proyecto en la herramienta.
Pos condiciones	
Flujo de eventos normal	1. El usuario selecciona una clase del árbol de clases. 2. El usuario pulsa el botón “Modificar elemento” 3. El sistema muestra un formulario desde el que se pueden editar todos los datos de una clase. 4. El usuario modifica los datos de las clases que considere necesarios. 5. El sistema comprueba que se han rellenado todos los elementos de la clase. 6. El sistema modifica la clase dentro del contenedor de clases.
Flujo de eventos excepcional	5a. Si el usuario no ha rellenado todos los datos el sistema muestra un mensaje de error y se vuelve al paso 3.
Frecuencia de uso	Alta

2.2.14 Caso de uso: Gestionar Atributos**Escenario: Añadir Atributo**

Nombre	Añadir Atributo.
Objetivo	Agregar al modelo de datos de la herramienta un nuevo atributo de una clase.
Actores	Usuario

Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado, abierto o importado un proyecto en la herramienta.
Pos condiciones	El usuario podrá editar los datos de un nuevo atributo.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario selecciona una clase del árbol de clases cuyo elemento raíz es el denominado “Contenedor de clases” 2. El usuario pulsa el botón “Añadir elemento al árbol”. 3. El sistema abre una nueva ventana en la que el usuario tiene que escoger que tipo de elemento va a añadir a la clase. 4. El usuario marca Atributo y pulsa el botón aceptar. 5. El sistema muestra un formulario con los datos a añadir relativos a un nuevo atributo de una clase. 6. El usuario rellena todos los datos del formulario. 7. El sistema comprueba que se han introducido los datos correctamente. 8. El sistema almacena el nuevo atributo en la clase seleccionada del contenedor de clases del sistema.
Flujo de eventos excepcional	<ol style="list-style-type: none"> 4a. Si el usuario pulsa el botón cancelar el caso de uso finaliza. 6a. Si no se rellenan todos los datos del formulario el sistema mostrará un error y se vuelve al paso 5.
Frecuencia de uso	Alta

Escenario: Borrar Atributo

Nombre	Borrar Atributo.
Objetivo	Eliminar del modelo de datos de la herramienta un atributo de una clase.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado, abierto o importado un proyecto en la herramienta.
Pos condiciones	
Flujo de eventos normal	<ol style="list-style-type: none"> 7. El usuario selecciona un atributo del árbol de clases cuyo elemento raíz es el denominado “Contenedor de clases” 8. El usuario pulsa el botón “Borrar del árbol el elemento seleccionado”. 9. El sistema abre una nueva ventana en la que pide al usuario confirmación para borrar el atributo. 10. El usuario acepta el mensaje de confirmación. 11. El sistema borra el atributo de la clase del contenedor de clases.
Flujo de eventos excepcional	<ol style="list-style-type: none"> 4a. Si el usuario pulsa el botón cancelar el caso de uso finaliza.
Frecuencia de uso	Alta

Escenario: Modificar Atributo

Nombre	Modificar Atributo.
Objetivo	Modificar alguna de las características propias de un atributo de una clase que se verán reflejadas a la hora de generar el código.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado, abierto o importado un proyecto en la herramienta.
Pos condiciones	
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario selecciona un atributo de una clase del árbol de clases. 2. El usuario pulsa el botón “Modificar elemento” 3. El sistema muestra un formulario desde el que se pueden editar todos los datos de un atributo. 4. El usuario modifica los datos del atributo que considere necesarios. 5. El sistema comprueba que se han rellenado todos los elementos del atributo. 6. El sistema modifica el atributo de la clase dentro del contenedor de clases.
Flujo de eventos excepcional	5a. Si el usuario no ha rellenado todos los datos el sistema muestra un mensaje de error y se vuelve al paso 3...
Frecuencia de uso	Alta

2.2.15 Caso de uso: Gestionar Métodos**Escenario: Añadir Método**

Nombre	Añadir Método.
Objetivo	Agregar al modelo de datos de la herramienta un nuevo método de una clase.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado, abierto o importado un proyecto en la herramienta.
Pos condiciones	El usuario podrá editar los datos de un nuevo método.

Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario selecciona una clase del árbol de clases cuyo elemento raíz es el denominado “Contenedor de clases” 2. El usuario pulsa el botón “Añadir elemento al árbol”. 3. El sistema abre una nueva ventana en la que el usuario tiene que escoger que tipo de elemento va a añadir a la clase. 4. El usuario marca Método y pulsa el botón aceptar. 5. El sistema muestra un formulario con los datos a añadir relativos a un nuevo método de una clase. 6. El usuario rellena todos los datos del formulario. 7. El sistema comprueba que se han introducido los datos correctamente. 8. El sistema almacena el nuevo método en la clase seleccionada del contenedor de clases del sistema.
Flujo de eventos excepcional	<ol style="list-style-type: none"> 4a. Si el usuario pulsa el botón cancelar el caso de uso finaliza. 6a. Si no se rellenan todos los datos del formulario el sistema mostrará un error y se vuelve al paso 5.
Frecuencia de uso	Alta

Escenario: Borrar Método

Nombre	Borrar Método.
Objetivo	Eliminar del modelo de datos de la herramienta un método de una clase con todos sus atributos y métodos.
Actores	Usuario
Precondiciones	<p>Que la herramienta esté ejecutándose en el sistema.</p> <p>Que se haya creado, abierto o importado un proyecto en la herramienta.</p>
Pos condiciones	Se ha eliminado un método de una clase.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario selecciona un método del árbol de clases cuyo elemento raíz es el denominado “Contenedor de clases” 2. El usuario pulsa el botón “Borrar del árbol el elemento seleccionado”. 3. El sistema abre una nueva ventana en la que pide al usuario confirmación para borrar el método. 4. El usuario acepta el mensaje de confirmación. 5. El sistema borra el método de la clase del contenedor de clases y refresca el contenido del diagrama de clases.
Flujo de eventos excepcional	<ol style="list-style-type: none"> 4a. Si el usuario pulsa el botón cancelar el caso de uso finaliza.
Frecuencia de uso	Alta

Escenario: Modificar Método

Nombre	Modificar Método.
Objetivo	Modificar alguna de las características propias de un método de una clase que se verán reflejadas a la hora de generar el código.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado, abierto o importado un proyecto en la herramienta.
Pos condiciones	
Flujo de eventos normal	<ol style="list-style-type: none">1. El usuario selecciona un método de una clase del árbol de clases.2. El usuario pulsa el botón “Modificar elemento”3. El sistema muestra un formulario desde el que se pueden editar todos los datos de un método.4. El usuario modifica los datos del atributo que considere necesarios.5. El sistema comprueba que se han rellenado todos los elementos del método.6. El sistema modifica el método de la clase dentro del contenedor de clases.
Flujo de eventos excepcional	5a. Si el usuario no ha rellenado todos los datos el sistema muestra un mensaje de error y se vuelve al paso 3.
Frecuencia de uso	Alta

PARTE DE IDE DE JAVA

2.2.16 Caso de uso: Nuevo Archivo Java

Nombre	Nuevo Archivo Java
Objetivo	Crear un nuevo archivo Java en blanco sobre el que trabajar.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que exista un proyecto creado, abierto o importado. Que se haya generado el código.
Pos condiciones	Se ha creado un nuevo archivo en disco sobre el que escribir código.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario accede al menú Archivo - Nuevo Archivo Java. 2. El sistema muestra un formulario para introducir el nombre que se le quiere dar al archivo a crear y también se selecciona una carpeta para colocar debajo el documento creado. 3. El usuario da un nombre al archivo y selecciona la carpeta. 4. El sistema valida que se ha completado el formulario. 5. El sistema crea un nuevo archivo en disco, añade un acceso directo al árbol de archivos java y abre el archivo en blanco en el panel de código.
Flujo de eventos excepcional	4a. Si no se ha introducido un nombre para el archivo a crear el sistema muestra un mensaje de error y se vuelve al paso 3.
Frecuencia de uso	Alta

2.2.17 Caso de uso: Abrir Archivo Java

Nombre	Abrir Archivo Java
Objetivo	Añade un nuevo archivo Java ya existente en disco al proyecto.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que exista un proyecto creado, abierto o importado. Que se haya generado el código.
Pos condiciones	Se ha añadido un nuevo archivo de código java al proyecto.

Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario accede al menú Archivo - Abrir Archivo Java. 2. El sistema muestra un formulario para introducir la ruta en la que se encuentra el archivo a añadir al proyecto, esta ruta puede añadirse manualmente o a través de un botón que abre una ventana en la que se filtran los datos por la extensión *.java. En el formulario también se encuentra una selección para escoger donde quiere que se ponga el acceso directo del archivo añadido. 3. El usuario rellena la ruta y selecciona la carpeta. 4. El sistema valida que se ha completado el formulario. 5. El sistema copia el archivo en la carpeta de los archivos generados java, y añade el acceso directo al archivo al árbol de enlaces a ficheros java.
Flujo de eventos excepcional	<p>4a. Si la ruta es incorrecta y el sistema no encuentra el archivo se muestra un mensaje de error y se vuelve al paso 3.</p>
Frecuencia de uso	Alta

2.2.18 Caso de uso: Borrar Archivo Java

Nombre	Borrar Archivo Java
Objetivo	Borra de disco un archivo java existente en el proyecto.
Actores	Usuario
Precondiciones	<p>Que la herramienta esté ejecutándose en el sistema.</p> <p>Que exista un proyecto creado, abierto o importado.</p> <p>Que se haya generado el código.</p>
Pos condiciones	Se ha borrado un archivo de código java o una imagen del proyecto.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario accede al menú Herramientas – Borrar Archivo Java 2. Si se ha seleccionado un nodo del árbol contenedor de los enlaces a ficheros java el sistema muestra un mensaje de confirmación. 3. El usuario acepta el mensaje de confirmación. 4. El sistema borra el archivo de disco.
Flujo de eventos excepcional	<p>2a. Si no se ha seleccionado ningún archivo del árbol el sistema muestra un mensaje de error.</p>
Frecuencia de uso	Normal

2.2.19 Caso de uso: Guardar Todo

Nombre	Guardar Todo
Objetivo	Guarda todos los archivos java del proyecto que están abiertos en pestañas.
Actores	Usuario

Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que exista un proyecto creado, abierto o importado. Que se haya generado el código.
Pos condiciones	Se ha guardado una última versión de todos los archivos abiertos en las pestañas.
Flujo de eventos normal	1. El usuario accede al menú Herramientas – Guardar Todo 2. Si existen archivos abiertos en pestañas el sistema los guarda.
Frecuencia de uso	Normal

2.2.20 Caso de uso: Abrir contendor de archivos

Nombre	Abrir contendor de archivos
Objetivo	Localizar en disco de una manera rápida los archivos generados.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado o abierto un proyecto. Que se haya generado el código del proyecto abierto.
Pos condiciones	Se podrá acceder directamente a los archivos de código generados.
Flujo de eventos normal	1. El usuario accede al menú Herramientas - Abrir contendor de archivos 2. El sistema abre en el explorador de archivos de Windows la carpeta que contiene todos los archivos del proyecto, tanto los archivos java como las imágenes.
Frecuencia de uso	Normal

2.2.21 Caso de uso: Abrir Archivo en pestaña

Nombre	Abrir archivo en pestaña
Objetivo	Abrir un archivo del árbol de archivos generado en el panel de edición de código.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado o abierto un proyecto. Que se haya generado el código del proyecto abierto.
Pos condiciones	Se podrá acceder al código de los archivos generados para su edición.

- Flujo de eventos normal**
1. El usuario hace doble clic en un nodo del árbol de código, que es un enlace para abrir el fichero en el panel de código.
 2. El sistema abre el código en el panel derecho de código.
 3. El sistema habilita el botón de cerrar pestaña.

Frecuencia de uso Normal

2.2.22 Caso de uso: Imprimir Archivo

Nombre	Imprimir Archivo.
Objetivo	Enviar a la impresora un archivo de código java.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya cargado un proyecto en la herramienta. Que se haya generado el código a imprimir. Que exista una impresora en la que escribir la salida. Que se haya abierto un archivo en una pestaña.
Pos condiciones	La impresora tendrá un nuevo trabajo de impresión.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario selecciona una pestaña del panel central en el que se encuentran cargados los archivos *.java generados previamente. 2. El sistema activa la pestaña. 3. El usuario accede al menú Archivo - Imprimir Archivo 4. El sistema muestra un formulario para configurar los detalles de la página a imprimir (márgenes, tamaño, orientación). 5. El usuario modifica los datos del formulario. 6. El sistema muestra una nueva ventana para seleccionar una impresora del sistema y modificar valores como número de páginas a imprimir. 7. El usuario introduce los datos. 8. El sistema imprime el archivo según las preferencias del usuario.
Flujo de eventos excepcional	No hay.
Frecuencia de uso	Normal

2.2.23 Caso de uso: Guardar Pestaña

Nombre	Guardar Pestaña.
Objetivo	Almacenar en disco el código de una pestaña abierta.
Actores	Usuario

Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya cargado un proyecto en la herramienta. Que se haya generado el código. Que se haya abierto un archivo en una pestaña.
Pos condiciones	La copia del archivo en disco será la última versión.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario selecciona una pestaña del panel central en el que se encuentran cargados los archivos *.java generados previamente. 2. El sistema activa la pestaña. 3. El usuario accede al menú Archivo – Guardar Archivo Actual. 4. El sistema guarda los cambios realizados en el archivo en disco.
Flujo de eventos excepcional	No hay.
Frecuencia de uso	Normal

2.2.24 Caso de uso: Compilar Pestaña

Nombre	Compilar Pestaña.
Objetivo	Abre una consola de Windows en la que se compila el archivo de la pestaña seleccionada.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya cargado un proyecto en la herramienta. Que se haya generado el código. Que se haya abierto un archivo en una pestaña.
Pos condiciones	Se compila el archivo de la pestaña actual.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario selecciona una pestaña del panel central en el que se encuentran cargados los archivos *.java generados previamente. 2. El sistema activa la pestaña. 3. El usuario accede al menú Herramientas – Compilar pestaña actual. 4. El sistema abre una consola de Windows se posiciona en la carpeta en la que se encuentran los archivos y ejecuta el comando javac seguido del nombre del archivo java. En la consola se muestran los errores que puedan existir en el código de esa pestaña.
Flujo de eventos excepcional	No hay.
Frecuencia de uso	Muy Alta.

2.2.25 Caso de uso: Cerrar Pestaña

Nombre	Cerrar Pestaña.
Objetivo	Cierra una pestaña abierta en el panel de código.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya cargado un proyecto en la herramienta. Que se haya generado el código. Que se haya abierto un archivo en una pestaña.
Pos condiciones	Se cierra el archivo de la pestaña actual.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario selecciona una pestaña del panel central en el que se encuentran cargados los archivos *.java generados previamente. 2. El sistema activa la pestaña. 3. El usuario accede al botón cerrar pestaña. 4. El sistema abre una nueva ventana en la que se pregunta al usuario si desea guardar los cambios realizados en la pestaña actual. 5. El usuario decide si desea guardar o no los datos.
Flujo de eventos excepcional	No hay.
Frecuencia de uso	Muy Alta.

2.2.26 Caso de uso: Copiar selección de código

Nombre	Copiar selección de código
Objetivo	Copiar un trozo de código
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya cargado un proyecto en la herramienta. Que se haya generado el código. Que se haya abierto un archivo en una pestaña.
Pos condiciones	Tener el código copiado disponible en memoria.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario realiza una selección de código de una pestaña. 2. El usuario selecciona el menú Edición – Copiar o Ctrl + c. 3. El sistema almacena la selección de código en memoria.
Flujo de eventos excepcional	No hay.

Frecuencia de uso Normal.

2.2.27 Caso de uso: Pegar selección de código

Nombre	Pegar selección de código
Objetivo	Pegar un trozo de código
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya cargado un proyecto en la herramienta. Que se haya generado el código. Que se haya abierto un archivo en una pestaña.
Pos condiciones	Pegar una selección de código almacenada en memoria
Flujo de eventos normal	1. El usuario posiciona el cursor en un punto del panel de código. 2. El usuario selecciona el menú Editar - Pegar
Flujo de eventos excepcional	No hay.
Frecuencia de uso	Normal.

2.2.28 Caso de uso: Cortar selección de código

Nombre	Cortar selección de código
Objetivo	Cortar un trozo de código
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya cargado un proyecto en la herramienta. Que se haya generado el código. Que se haya abierto un archivo en una pestaña.
Pos condiciones	Tener el código cortado disponible en memoria.
Flujo de eventos normal	1. El usuario realiza una selección de código de una pestaña. 2. El sistema almacena la selección de código en memoria.
Flujo de eventos excepcional	No hay.
Frecuencia de uso	Normal.

2.2.29 Caso de uso: Cambiar a Vista Diseño

Nombre	Cambiar a Vista Diseño
Objetivo	Volver a la vista de Diseño para editar el diagrama de clases.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que exista un proyecto creado, abierto o importado. Que se haya generado el código.
Pos condiciones	Volver a la vista de diseño.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario accede al menú Vista – Diseño 2. El sistema muestra un mensaje de confirmación para volver a la vista de diseño. 3. El usuario pulsa el botón de confirmación. 4. El sistema vuelve a mostrar la vista de diseño.
Frecuencia de uso	Normal

2.2.30 Caso de uso: Generar Documentación Javadoc

Nombre	Generar Documentación Javadoc
Objetivo	Tener información de las clases, atributos y métodos de nuestra aplicación, la documentación se genera en base a los comentarios existentes en el código.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que exista un proyecto creado, abierto o importado. Que se haya generado el código. Que se haya documentado el código.
Pos condiciones	Tener en disco los HTML generados de la documentación.
Flujo de eventos normal	<ol style="list-style-type: none"> 1. El usuario accede al menú Herramientas – Generar documentación Javadoc. 2. El sistema abre una nueva consola de Windows e introduce el comando Javadoc, para que se creen los archivos HTML de documentación. 3. El sistema abre en el explorador firefox el archivo de documentación de la clase principal del sistema.
Frecuencia de uso	Normal

2.2.31 Caso de uso: Compilar Proyecto

Nombre	Compilar Proyecto
Objetivo	Comprobar si el código generado y modificado es correcto.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado o abierto un proyecto. Que se haya generado el código.
Pos condiciones	Se muestran los posibles errores de código.
Flujo de eventos normal	1. El usuario accede al menú Herramientas – Compilar Archivos Generados. 2. El sistema abre una nueva consola de Windows, se sitúa en el directorio en el que se encuentran los archivos e introduce el comando javac *.java para compilar todos los archivos del proyecto. Los errores del proyecto también se muestran en esta consola.
Flujo de eventos excepcional	No hay
Frecuencia de uso	Alta

2.2.32 Caso de uso: Ejecutar Proyecto

Nombre	Ejecutar Proyecto
Objetivo	Probar el proyecto generado.
Actores	Usuario
Precondiciones	Que la herramienta esté ejecutándose en el sistema. Que se haya creado o abierto un proyecto. Que se haya generado el código.
Pos condiciones	
Flujo de eventos normal	1. El usuario accede al menú Herramientas – Ejecutar Proyecto generado. 2. El sistema abre una nueva consola de Windows, se sitúa en el directorio en el que se encuentran los archivos e introduce el comando java para compilar todos los archivos del proyecto. Los errores del proyecto también se muestran en esta consola.
Flujo de eventos excepcional	No hay
Frecuencia de uso	Alta

3. DISEÑO

3.1 Patrón utilizado

Los patrones de diseño son un conjunto de prácticas de óptimo diseño que se utilizan para abordar problemas recurrentes en la programación orientada a objetos. Expresan el esquema fundamental de organización para sistemas de software. Proveen un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos, ayudando así a especificar la estructura de una aplicación.

Durante la realización de este proyecto se ha utilizado el patrón Modelo Vista Controlador. El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

El Modelo es el objeto que representa los datos del programa. Maneja los datos y controla todas sus transformaciones. El Modelo no tiene conocimiento específico del controlador o de la Vista. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el Modelo y sus Vistas, y notificar a las Vistas cuando cambia el Modelo.

La Vista es el objeto que maneja la presentación visual de los datos representados por el Modelo. Genera una representación visual del Modelo y muestra los datos al usuario. Interactúa con el Modelo a través de una referencia al propio Modelo.

El Controlador es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo. Cuando se realiza algún cambio, entra en acción, bien sea por cambios en la información del Modelo o por alteraciones de la Vista. Interactúa con el Modelo a través de una referencia al propio Modelo.

En el proyecto la clase Controller recibe una solicitud de alguna de las ventanas del paquete de Vista y coordina su realización para hacer modificaciones en las estructuras del modelo que son las representadas en las clases ClassContainer, AssociationTable, GeneralizationTable...

El patrón MVC representa un mecanismo de mejora de procesos de desarrollo de software, fácil de comprender y aplicar. La principal ventaja de la aplicación del patrón MVC es la reutilización de componentes del modelo, si queremos modificar la interfaz de la aplicación sólo tendremos que modificar las ventanas pertenecientes al paquete Vista.

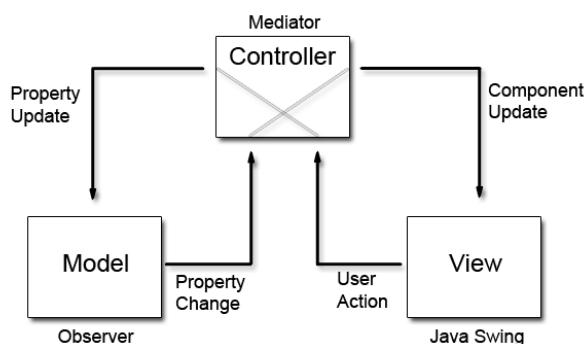


Ilustración 21: Patrón MVC

3.2 Diagramas de secuencia de la herramienta

Los diagramas de secuencia de la herramienta representan la interacción entre los diferentes objetos del sistema. Se visualizan los eventos generados por los actores cuando solicitan alguna operación y el intercambio de mensajes entre los objetos de las clases.

Los mensajes son llamadas a los métodos de la clase y se representan mediante flechas horizontales orientadas del emisor al receptor. El orden de envío de los mensajes viene dado por la posición sobre el eje vertical.

A continuación se muestran los diagramas de secuencia de las dos partes del sistema.

PARTE DE DISEÑO

3.2.1 Diagrama de secuencia: Nuevo Proyecto

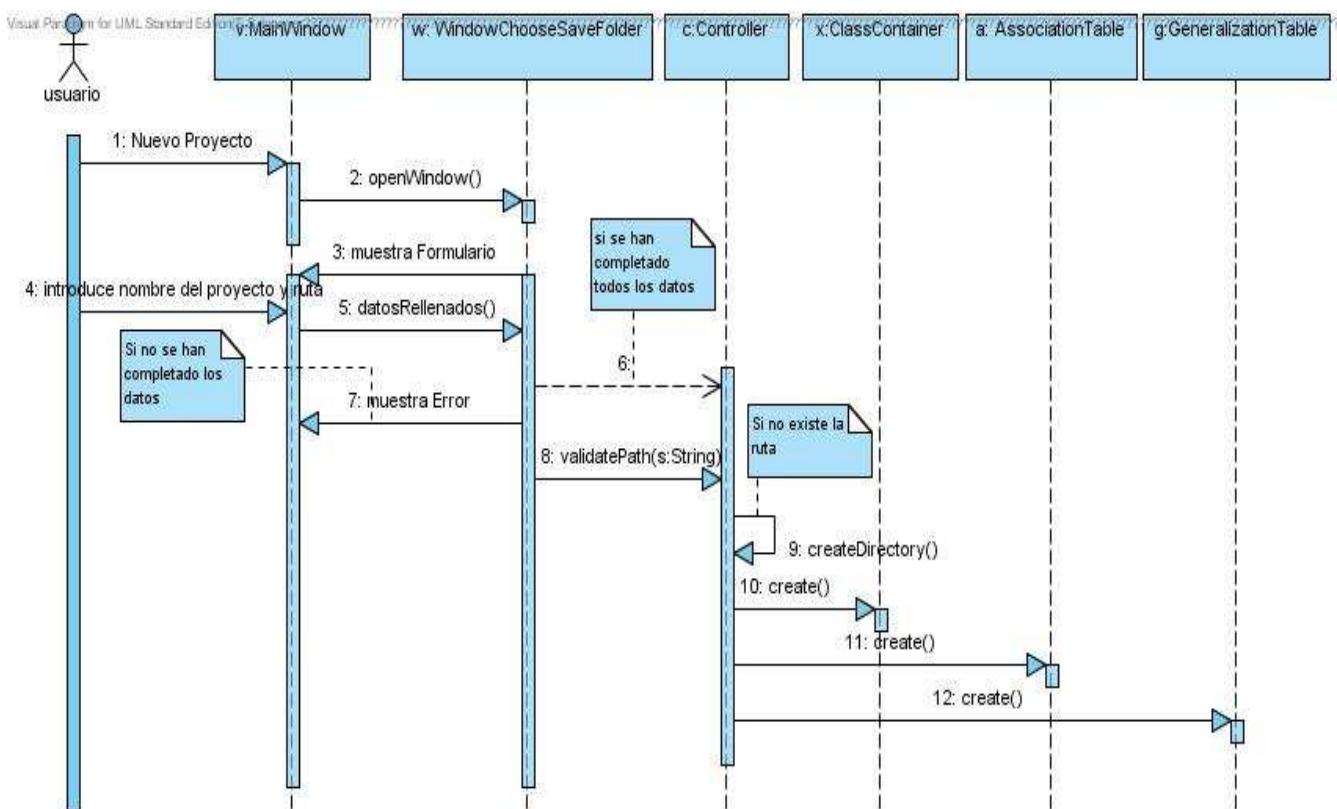


Ilustración 22: Diagrama de Secuencia - Nuevo proyecto

3.2.2 Diagrama de secuencia: Abrir Proyecto

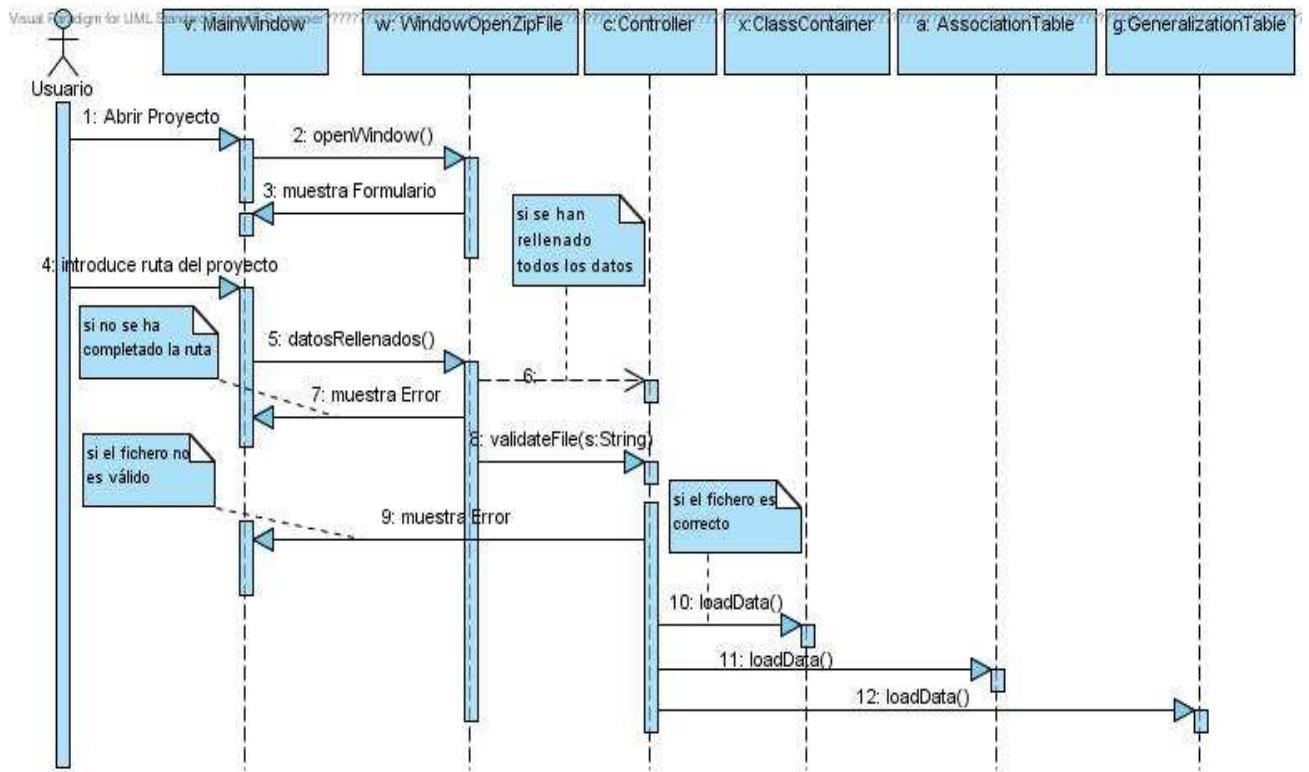


Ilustración 23: Diagrama de secuencia - Abrir proyecto

3.2.3 Diagrama de secuencia: Importar Archivo XMI

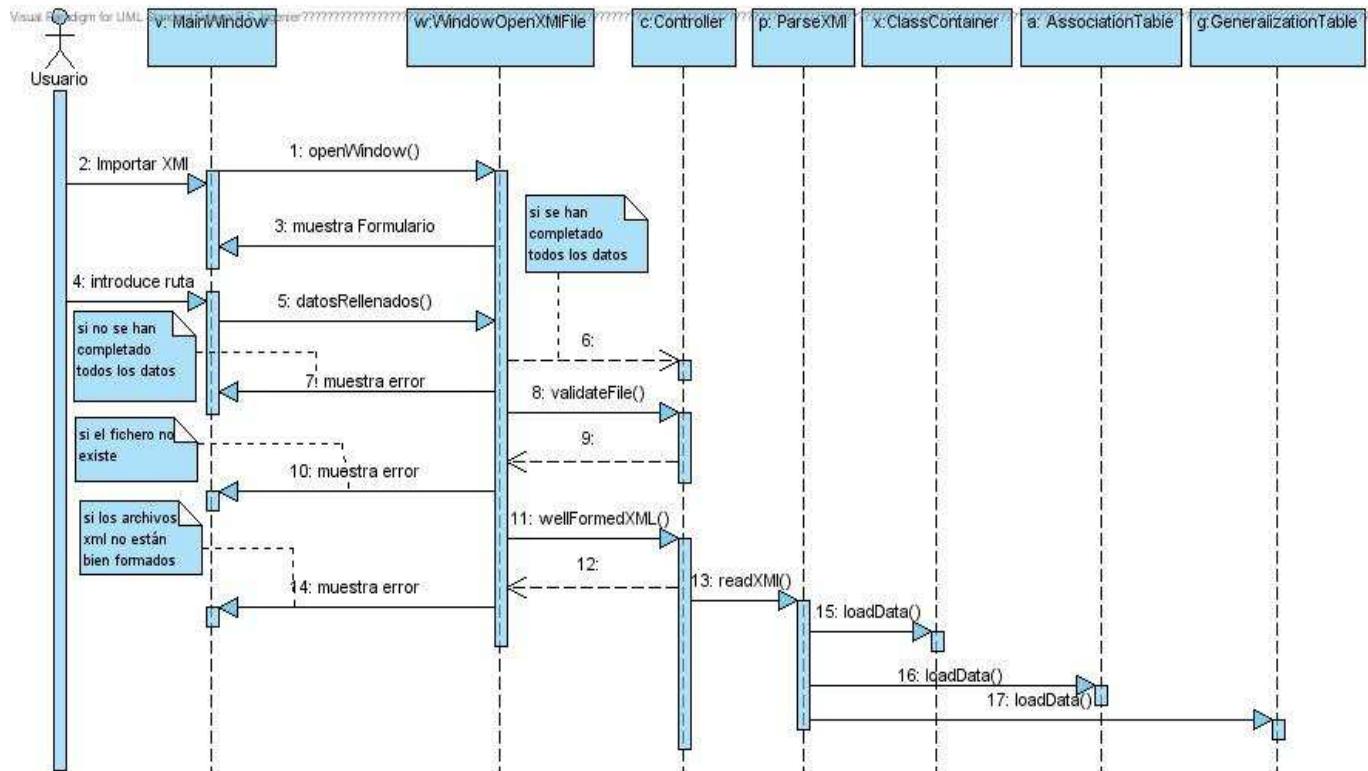


Ilustración 24: Diagrama de secuencia - Importar archivo XMI

3.2.4 Diagrama de secuencia: Generar código Java

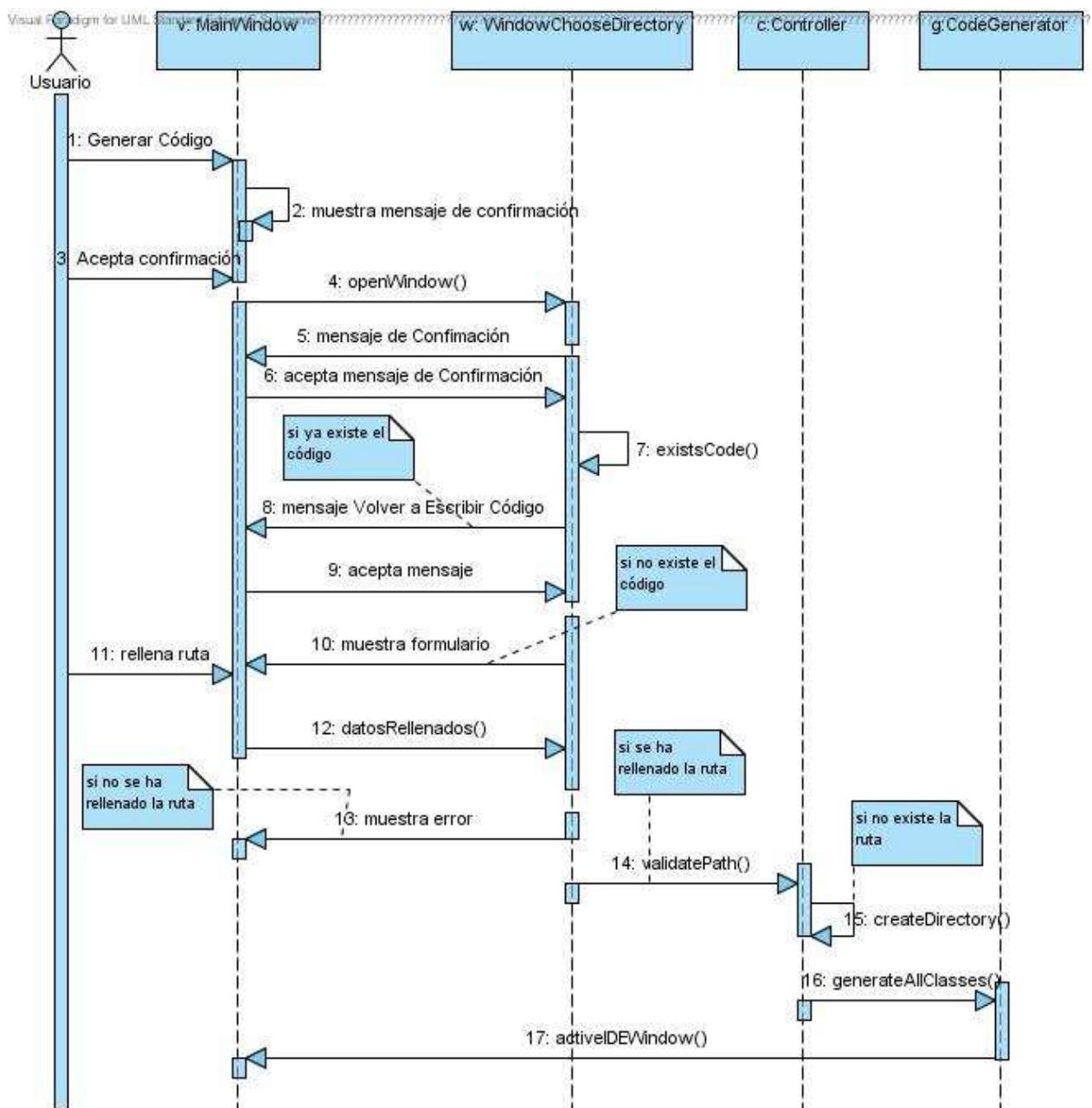


Ilustración 25: Diagrama de secuencia - Generar código

3.2.5 Diagrama de secuencia: Modificar Preferencias

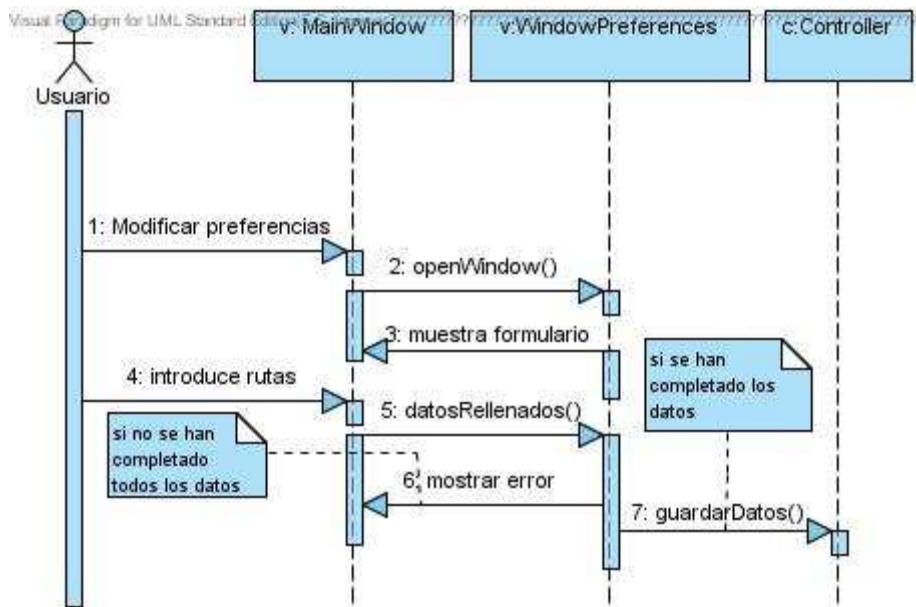


Ilustración 26: Diagrama de secuencia - Modificar preferencias

3.2.6 Diagrama de secuencia: Seleccionar Clase Principal

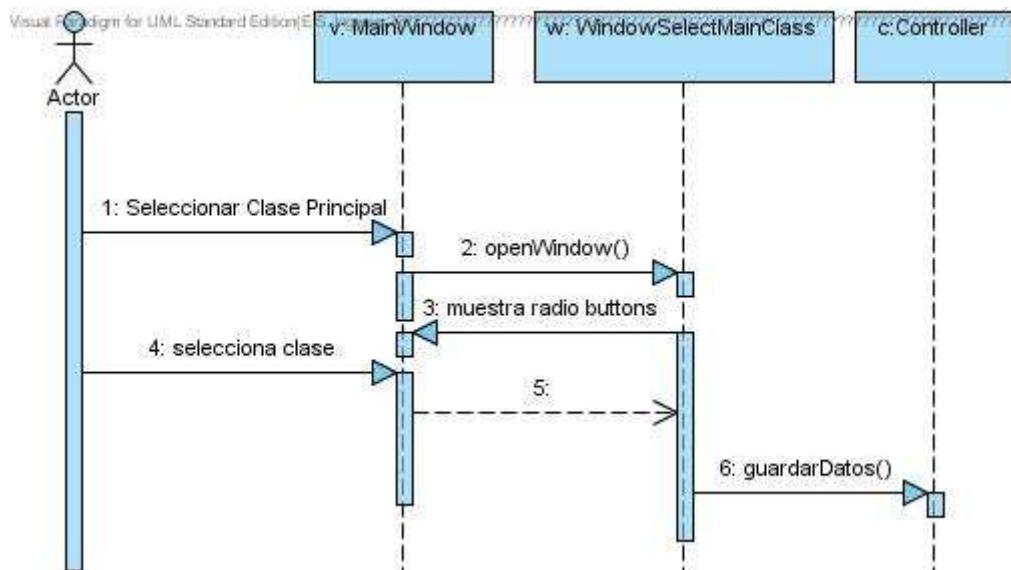


Ilustración 27: Diagrama de secuencia - Seleccionar clase principal

3.2.7 Diagrama de secuencia: Guardar Proyecto

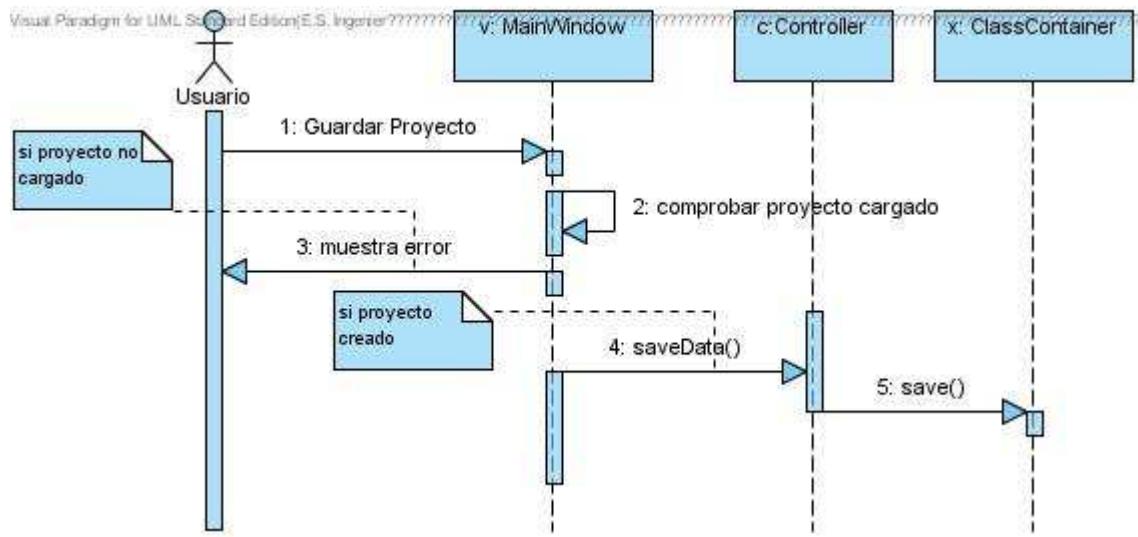


Ilustración 28: Diagrama de secuencia - Guardar proyecto

3.2.8 Diagrama de secuencia: Guardar Proyecto Como

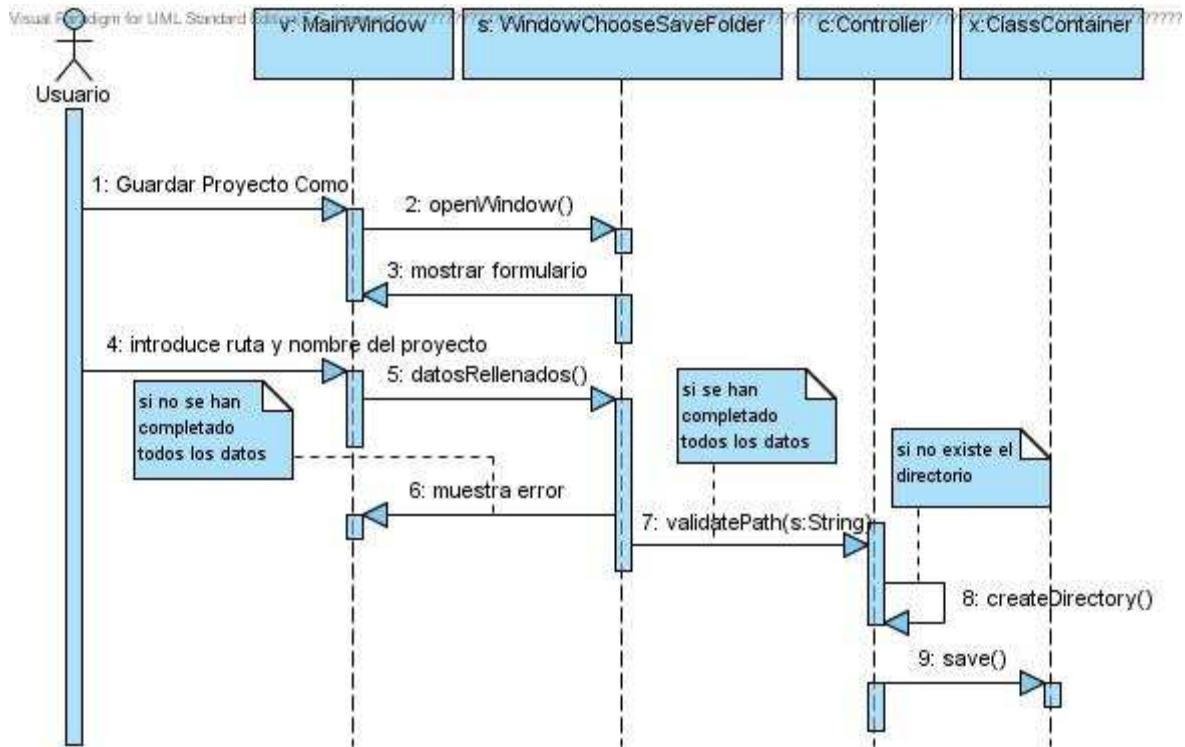


Ilustración 29: Diagrama de secuencia - Guardar Proyecto como

3.2.9 Diagrama de secuencia: Refrescar contenidos

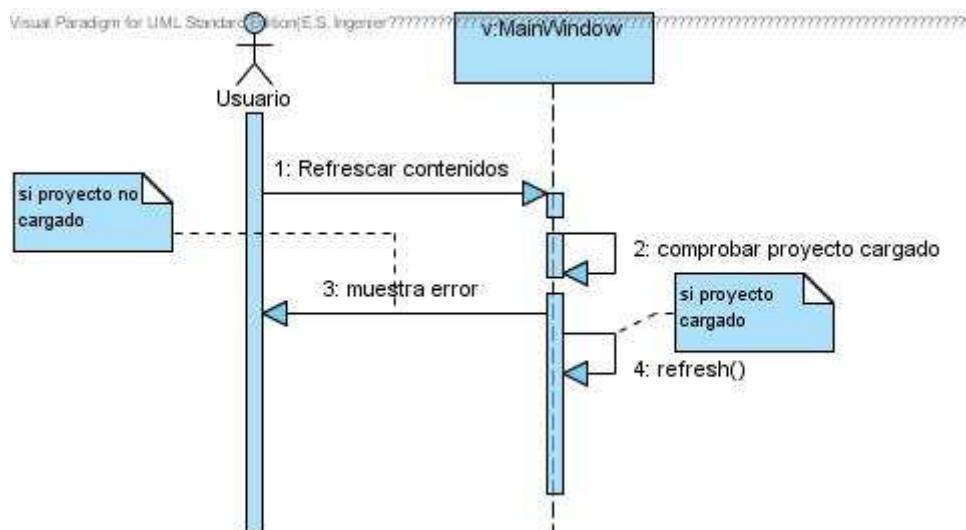


Ilustración 30: Diagrama de secuencia - Refrescar contenidos

3.2.10 Diagrama de secuencia: Fijar Diagrama

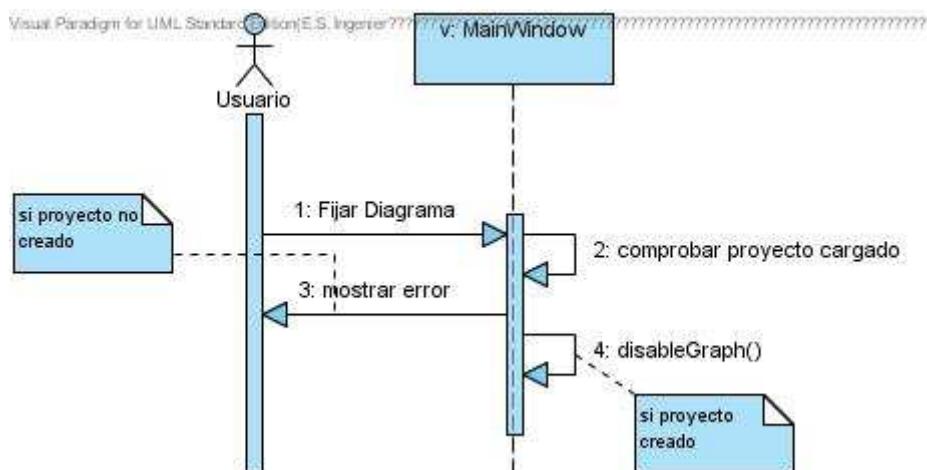


Ilustración 31: Diagrama de secuencia - Fijar diagrama

3.2.11 Diagrama de secuencia: Ocultar Menú Principal

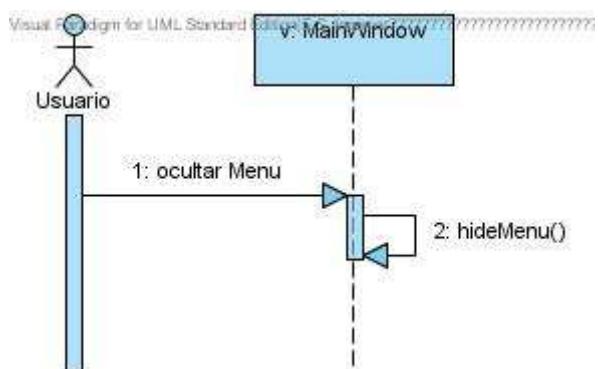


Ilustración 32: Diagrama de secuencia - Ocultar menú principal

3.2.12 Diagrama de secuencia: Borrar todos los elementos

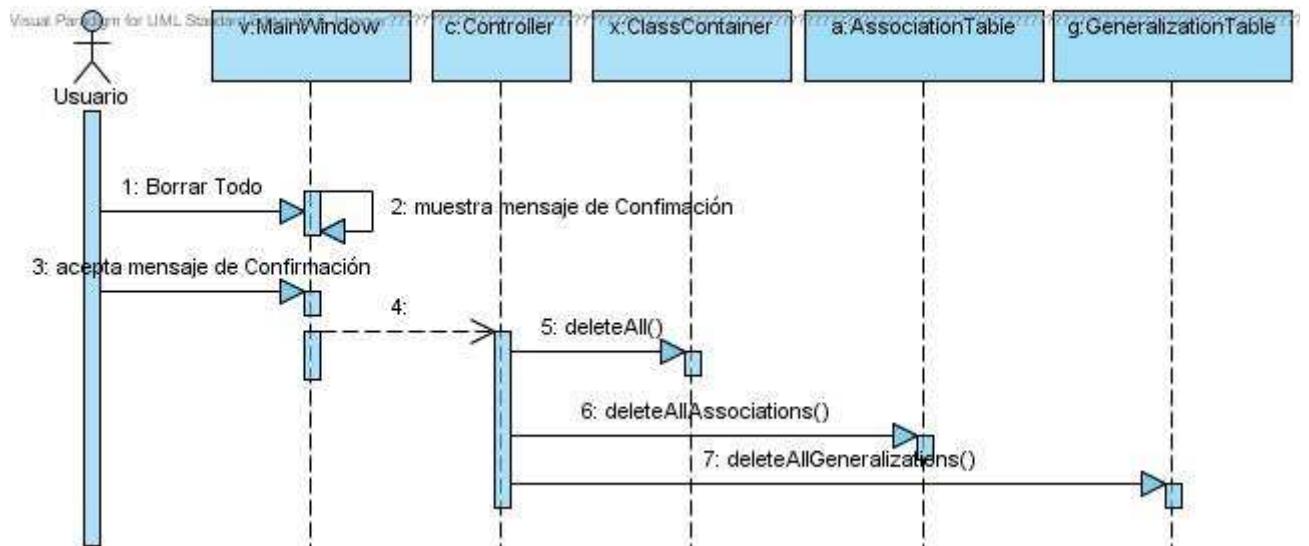


Ilustración 33: Diagrama de secuencia - Borrar todos los elementos

3.2.13 Diagrama de secuencia: Gestionar Clase

Escenario: Añadir Clase

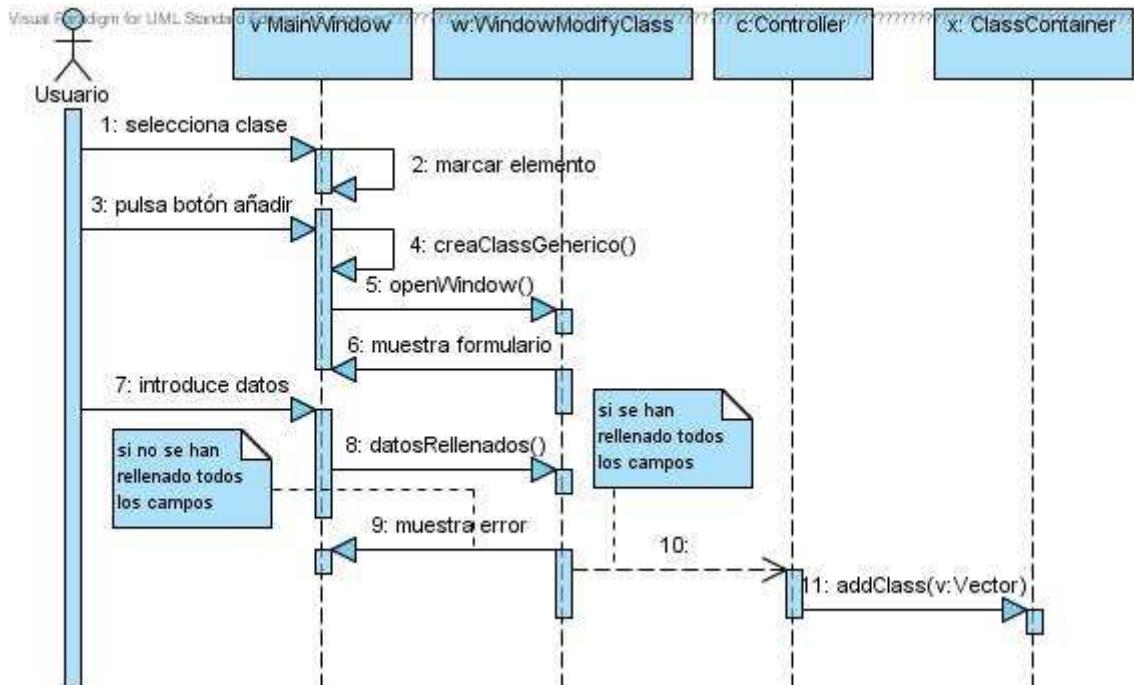


Ilustración 34: Diagrama de secuencia - Añadir Clase

Escenario: Borrar Clase

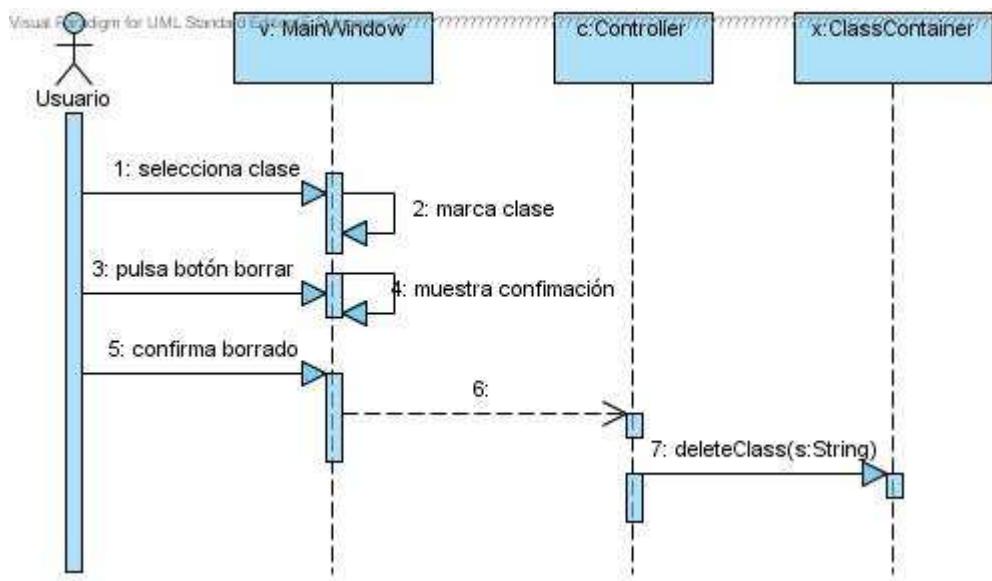


Ilustración 35: Diagrama de secuencia - Borrar Clase

Escenario: Modificar Clase

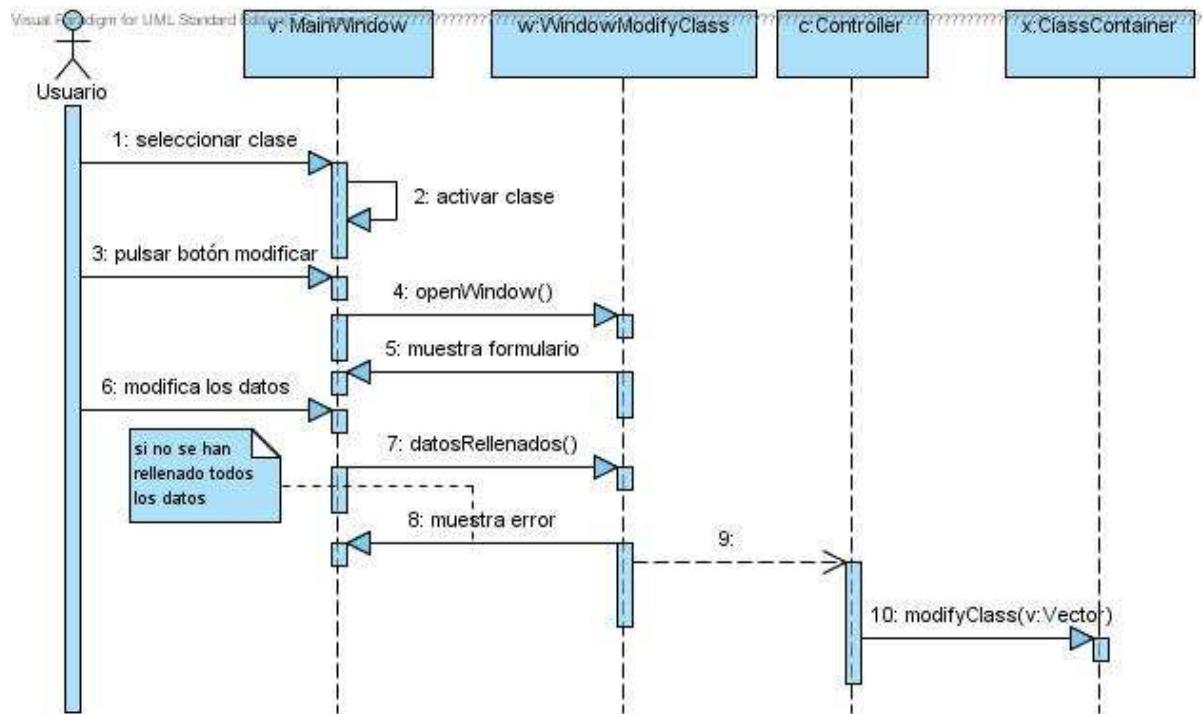


Ilustración 36: Diagrama de secuencia - Modificar clase

3.2.14 Diagrama de secuencia: Gestionar Atributos

Escenario: Añadir Atributo

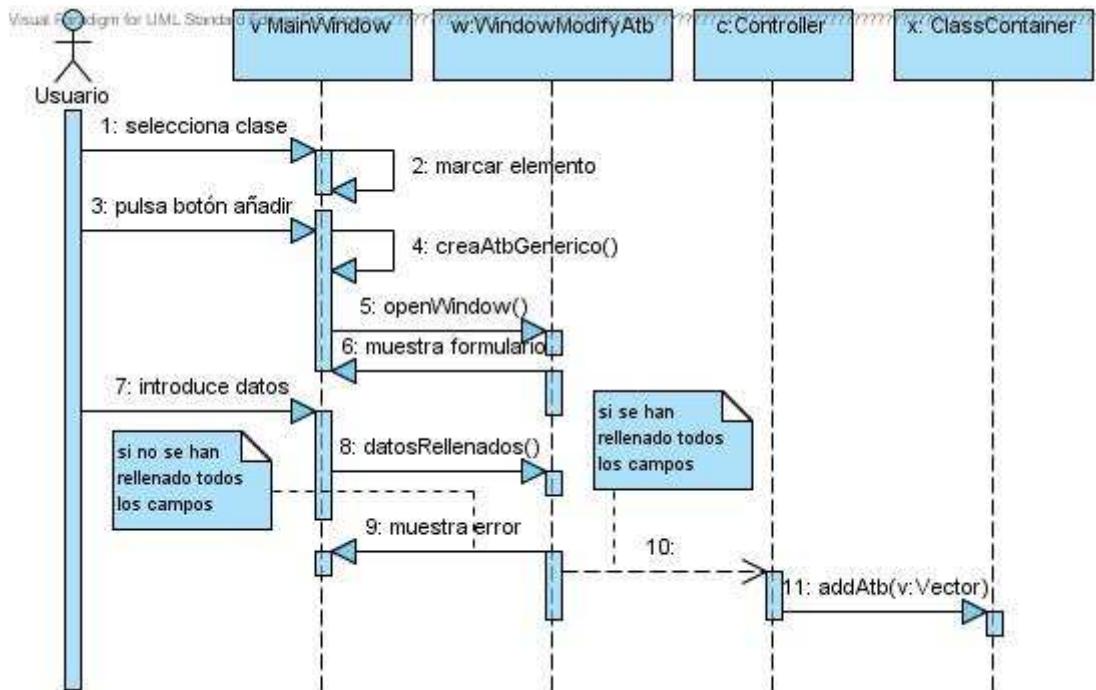


Ilustración 37: Diagrama de secuencia - Añadir atributo

Escenario: Borrar Atributo

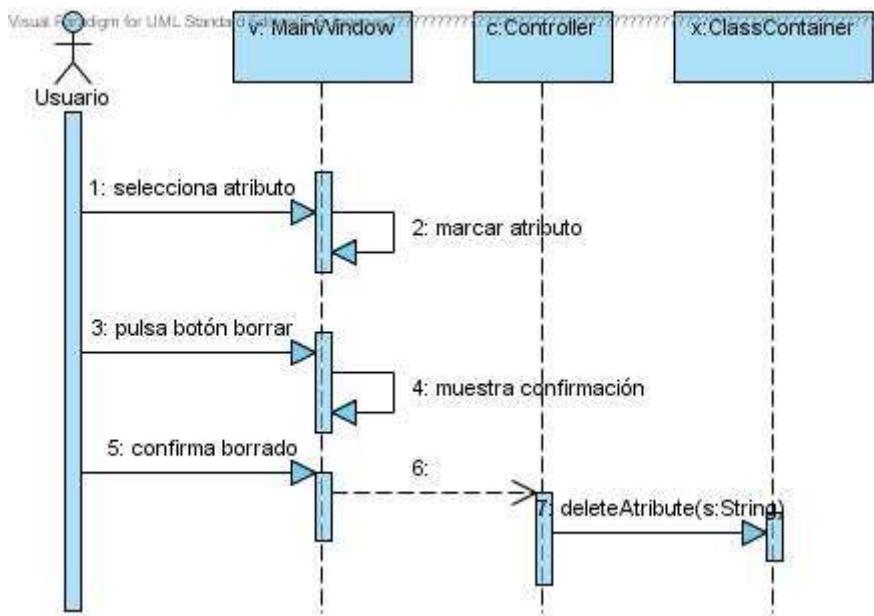


Ilustración 38: Diagrama de secuencia - Borrar atributo

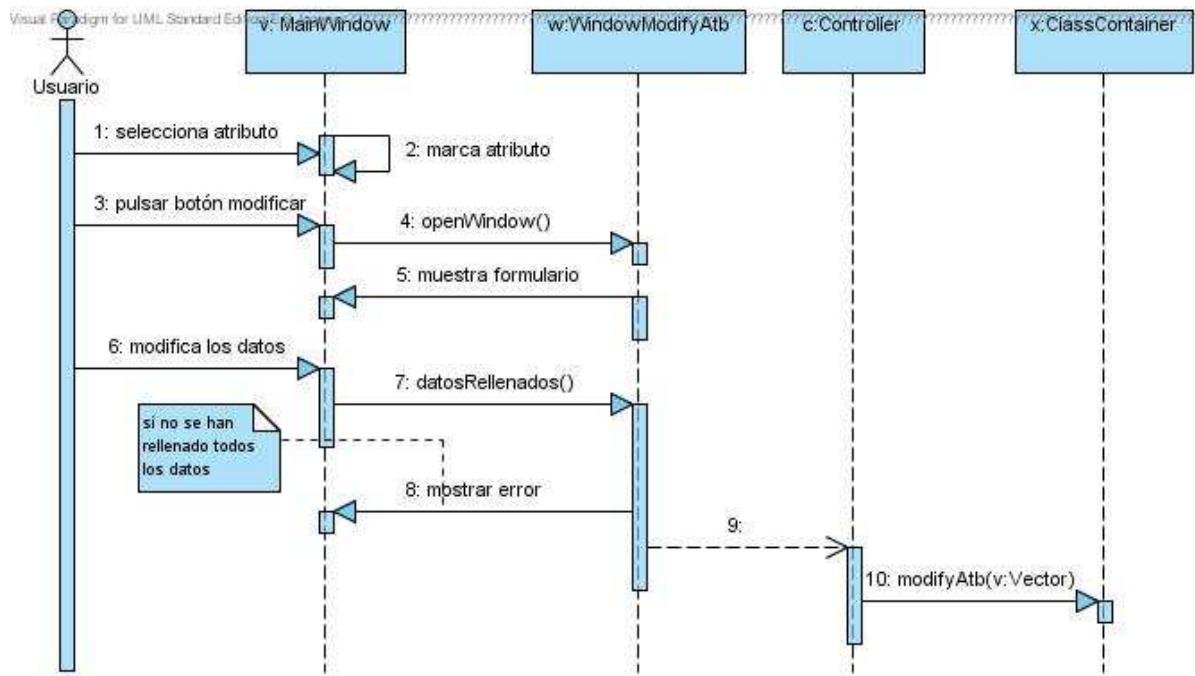
Escenario: Modificar Atributo

Ilustración 39: Diagrama de secuencia - Modificar atributo

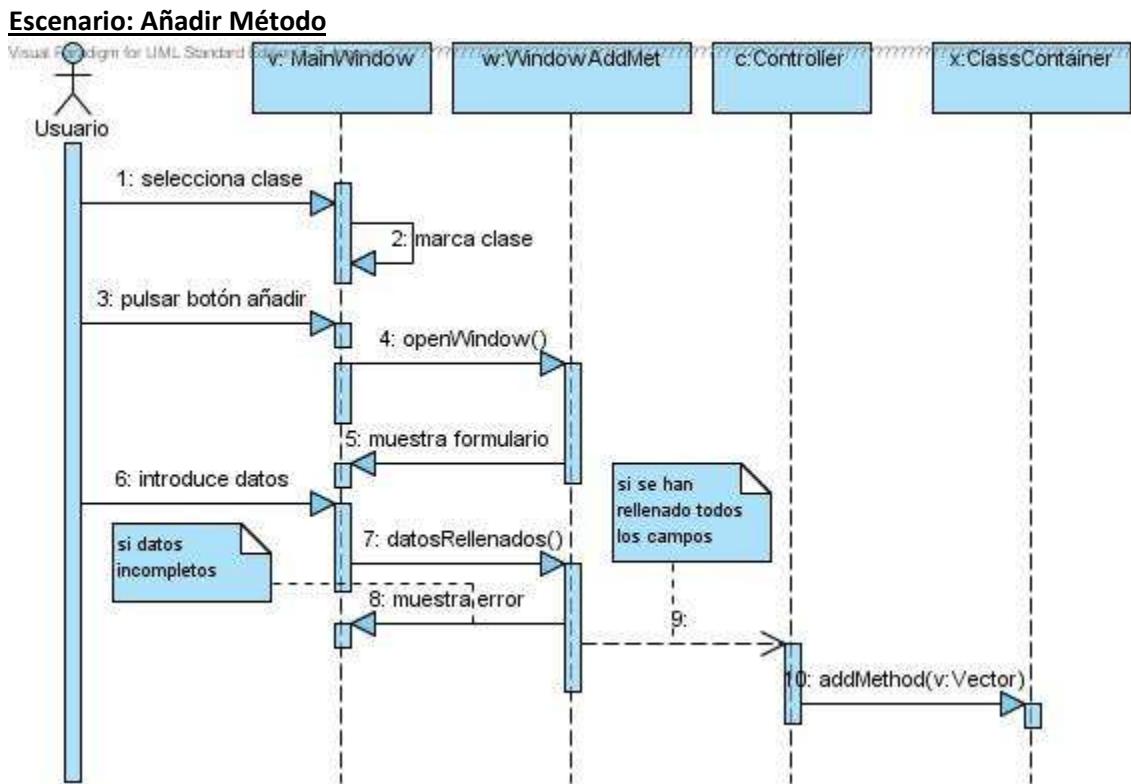
3.2.15 Diagrama de secuencia: Gestionar Métodos

Ilustración 40: Diagrama de secuencia - Añadir método

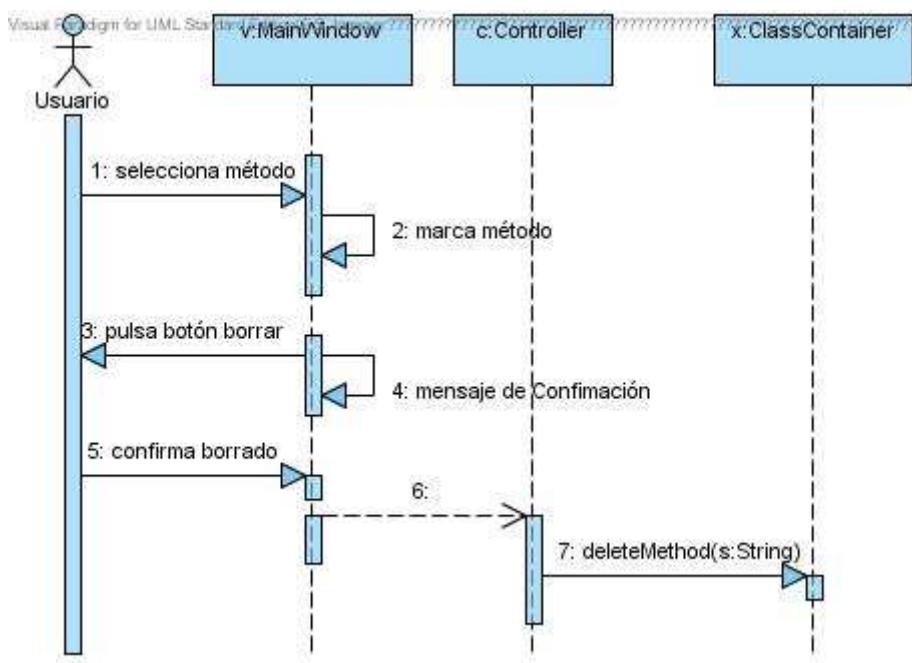
Escenario: Borrar Método

Ilustración 41: Diagrama de secuencia - Borrar método

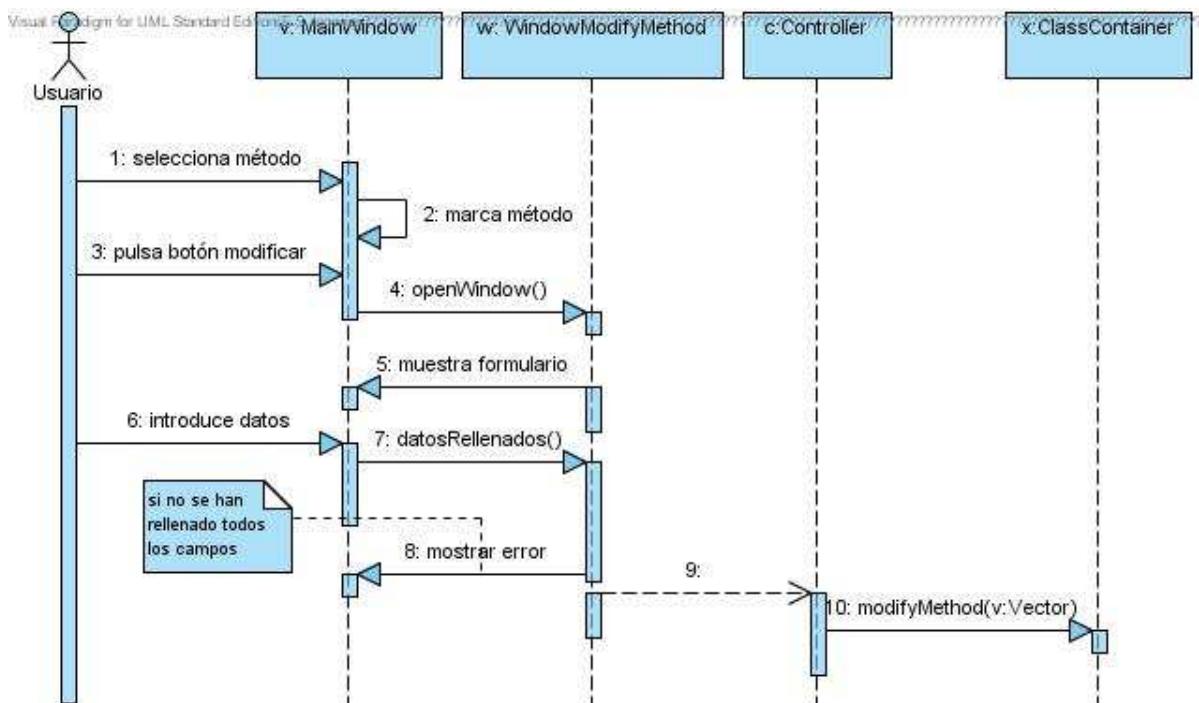
Escenario: Modificar Método

Ilustración 42: Diagrama de secuencia - Modificar método

PARTE IDE DE JAVA

3.2.16 Diagrama de secuencia: Nuevo Archivo Java

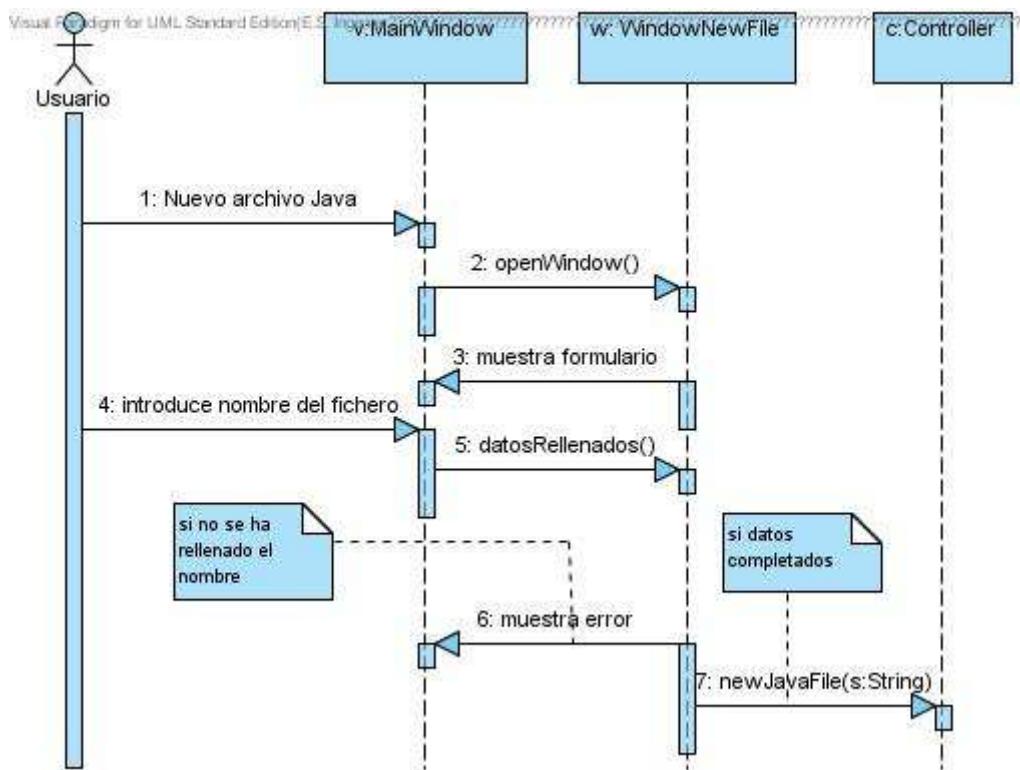


Ilustración 43: Diagrama de secuencia - Nuevo archivo java

3.2.17 Diagrama de secuencia: Abrir Archivo Java

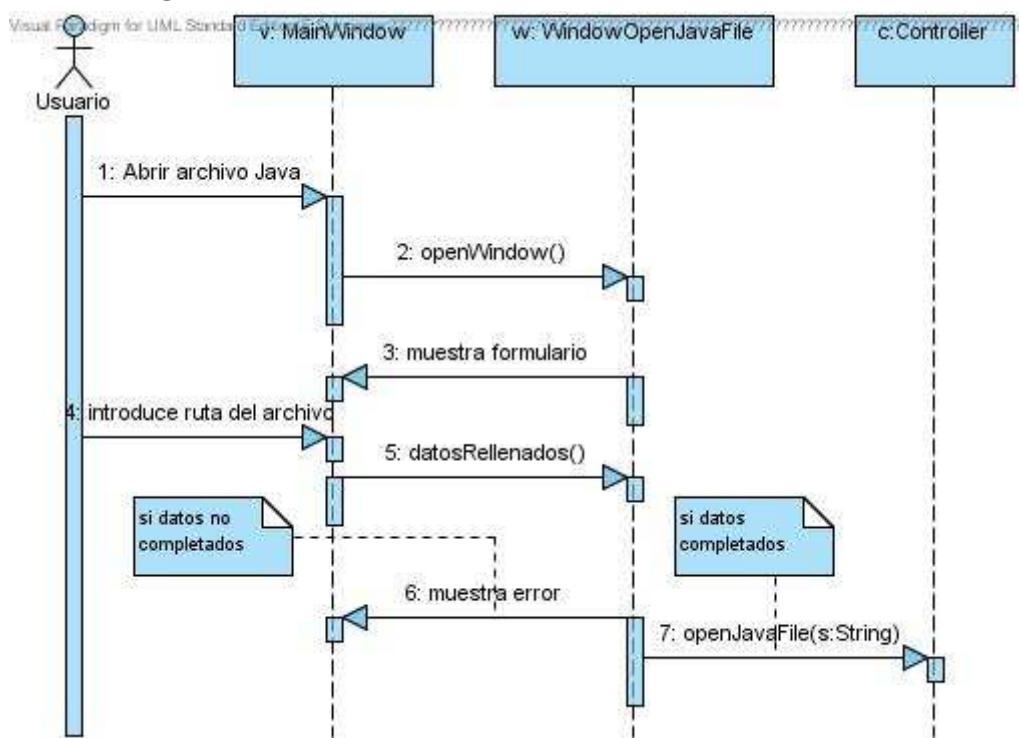


Ilustración 44: Diagrama de secuencia - Abrir archivo Java

3.2.18 Diagrama de secuencia: Borrar Archivo Java

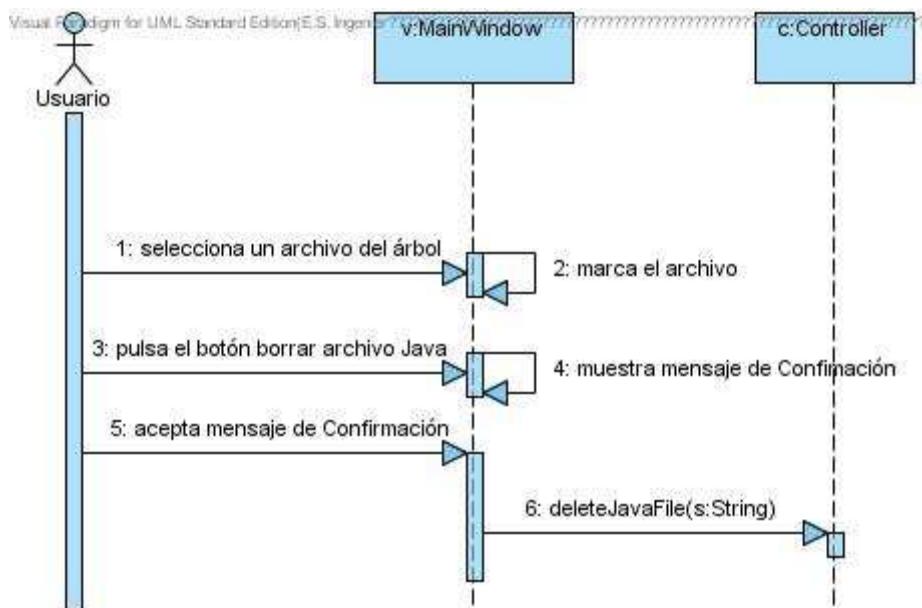


Ilustración 45: Diagrama de secuencia - Borrar archivo java

3.2.19 Diagrama de secuencia: Guardar Todo

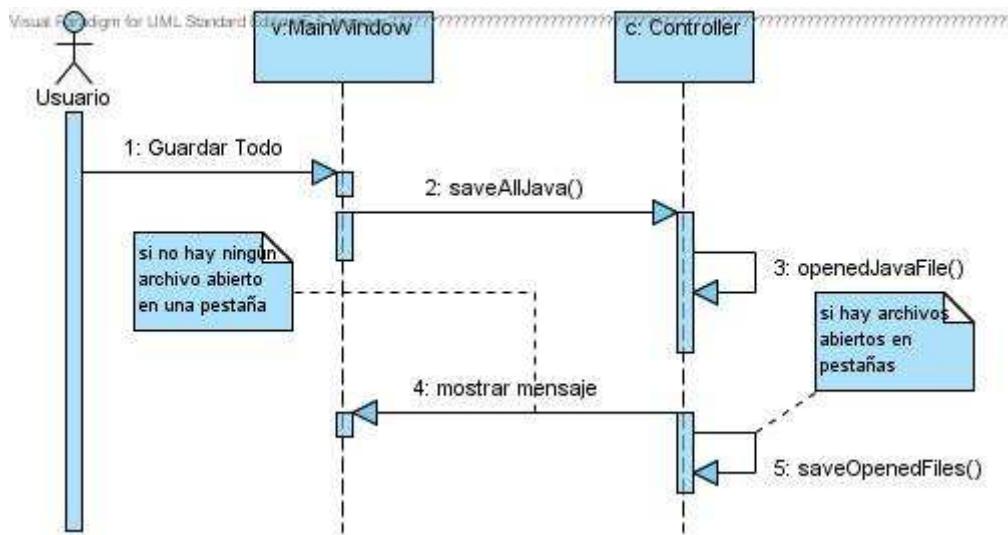


Ilustración 46: Diagrama de secuencia - Guardar Todo

3.2.20 Diagrama de secuencia: Abrir contenedor de archivos

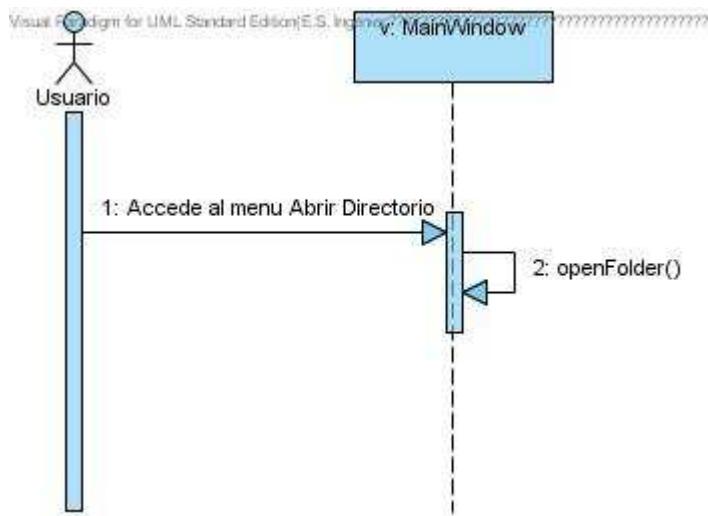


Ilustración 47: Diagrama de secuencia - Abrir contenedor de archivos

3.2.21 Diagrama de secuencia: Imprimir Archivo

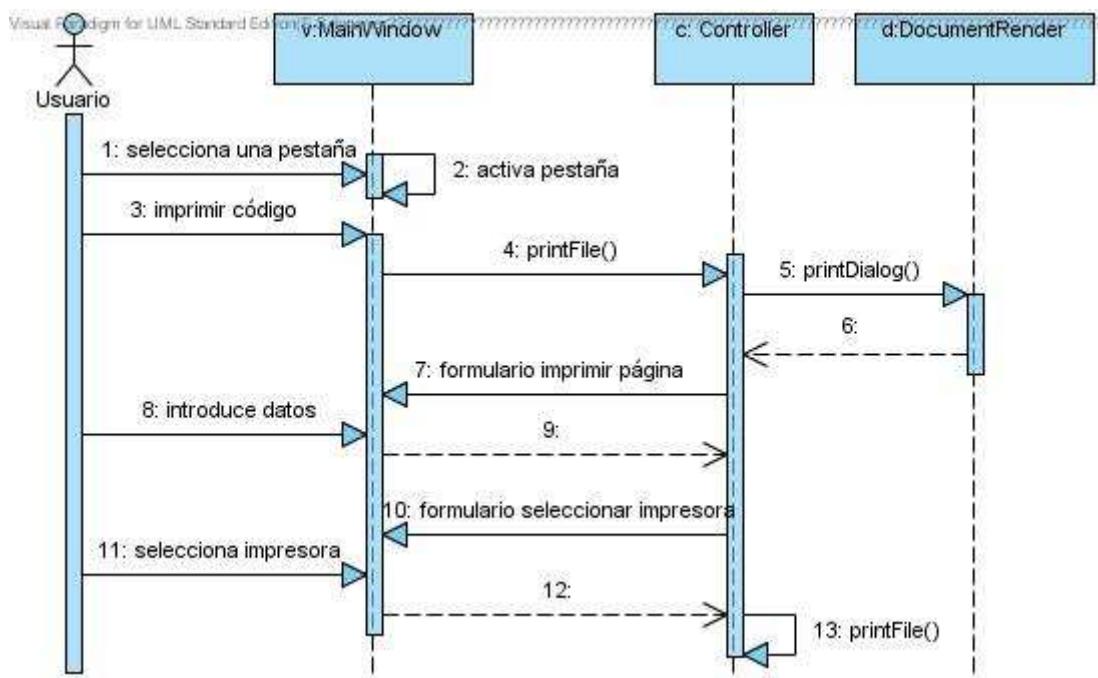


Ilustración 48: Diagrama de secuencia - Imprimir archivo

3.2.22 Diagrama de secuencia: Abrir archivo en pestaña



Ilustración 49: Diagrama de secuencia - Abrir archivo en pestaña

3.2.23 Diagrama de secuencia: Guardar Pestaña

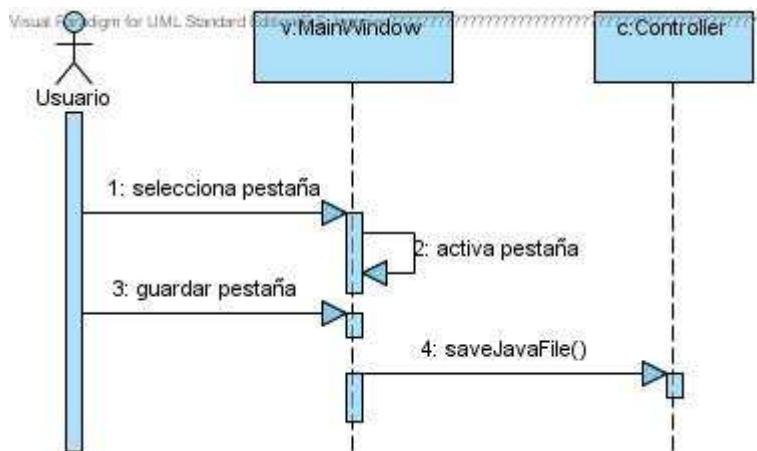


Ilustración 50: Diagrama de secuencia - Guardar Pestaña

3.2.24 Diagrama de secuencia: Compilar Pestaña

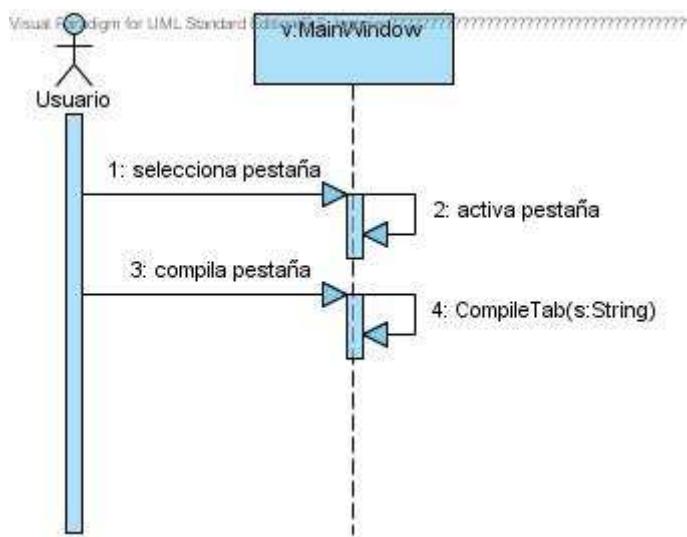


Ilustración 51: Diagrama de secuencia - Compilar pestaña

3.2.25 Diagrama de secuencia: Cerrar Pestaña

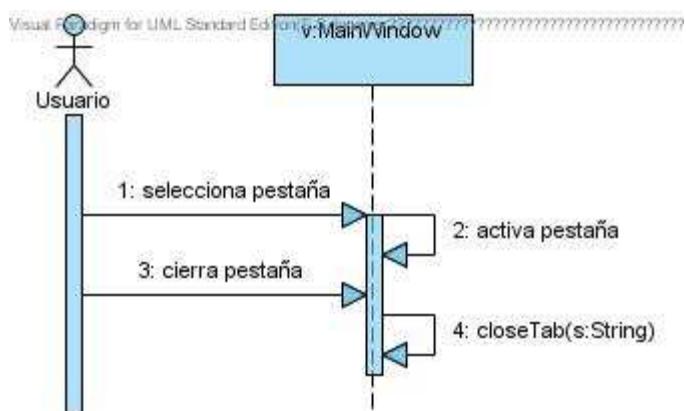


Ilustración 52: Diagrama de secuencia - Cerrar pestaña

3.2.26 Diagrama de secuencia: Copiar selección de código

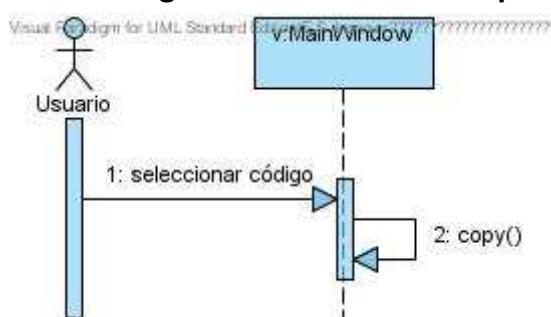


Ilustración 53: Diagrama de secuencia - Copiar selección de código

3.2.27 Diagrama de secuencia: Cortar selección de código

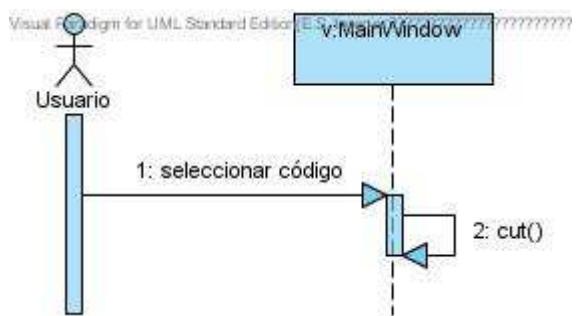


Ilustración 54: Diagrama de secuencia - Cortar selección de código

3.2.28 Diagrama de secuencia: Pegar selección de código

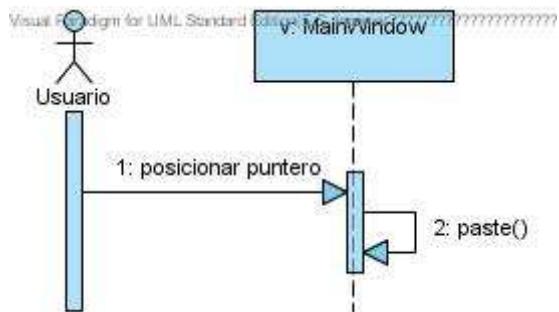


Ilustración 55: Diagrama de secuencia - Pegar selección de código

3.2.29 Diagrama de secuencia: Cambiar a Vista Diseño

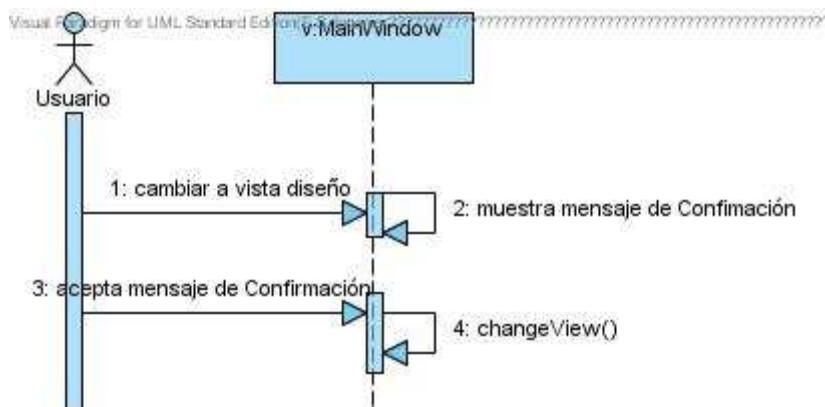


Ilustración 56: Diagrama de secuencia - Cambiar a Vista Diseño

3.2.30 Diagrama de secuencia: Generar Documentación Javadoc

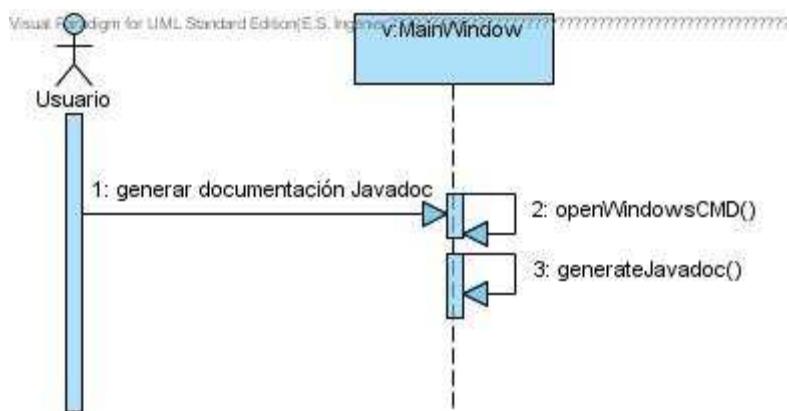


Ilustración 57: Diagrama de Secuencia - Generar documentación Javadoc

3.2.31 Diagrama de secuencia: Compilar Proyecto

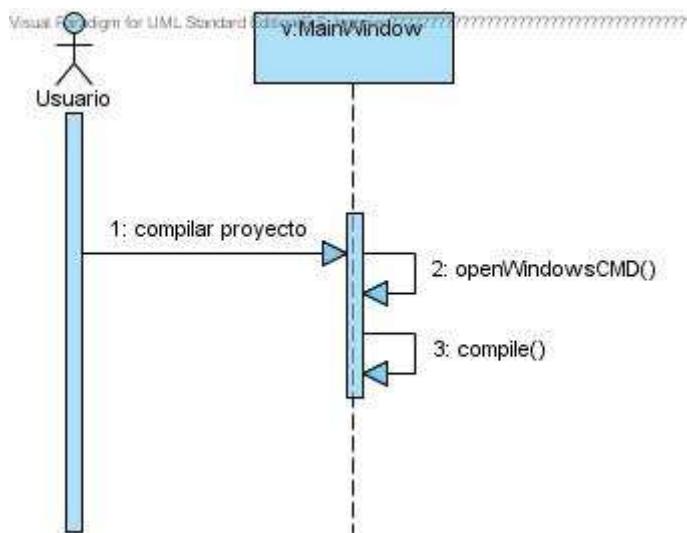


Ilustración 58: Diagrama de Secuencia - Compilar Proyecto

3.2.32 Diagrama de secuencia: Ejecutar Proyecto

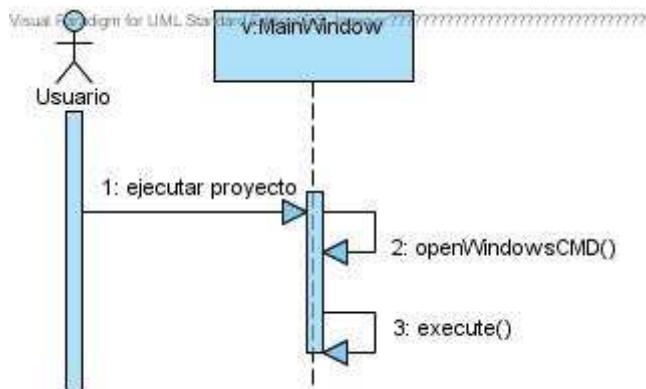


Ilustración 59: Diagrama de Secuencia - Ejecutar Proyecto

3.3 Diagrama de clases

El diagrama de clases expresa de manera general la estructura estática del sistema, en términos de clases y sus relaciones. Es decir, muestra la estructura de información del sistema y la visibilidad que tiene cada una de las clases dada por sus relaciones con las demás en el modelo. Cada clase tiene un comportamiento y un estado, las clases interactúan entre ellas para lograr un comportamiento mayor.

Además este diagrama es más inteligible para cualquier persona porque las clases que se crean en él se corresponden con objetos del mundo real, con sus características, sus acciones y su capacidad de interactuar con otros objetos.

A continuación se muestra el diagrama de clases total del sistema, sólo se han mostrado las clases que se han considerado más importantes para hacerlo más legible.

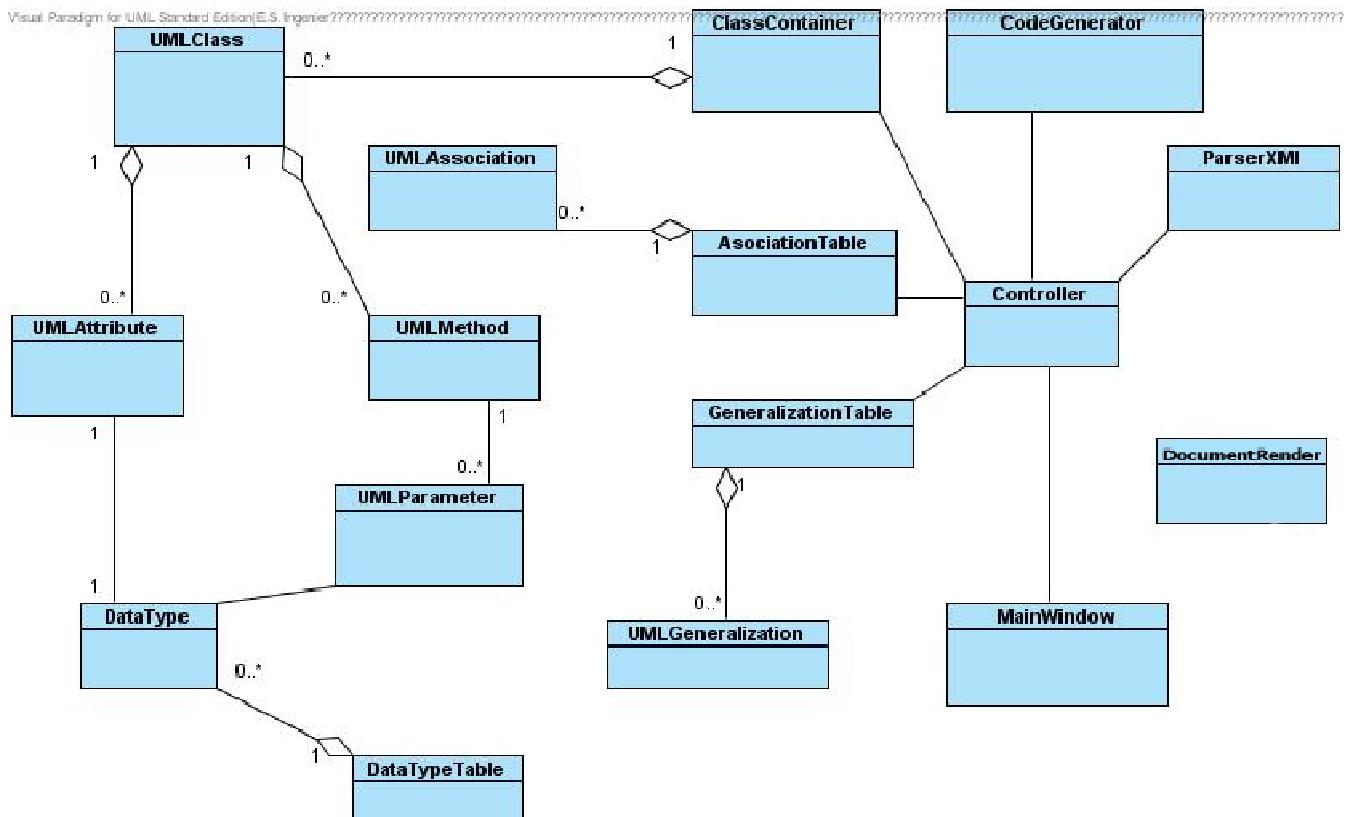


Ilustración 60: Diagrama de clases del sistema

3.4 Descripción de las clases

Para una mayor comprensión de los atributos y del funcionamiento de los métodos existe en la carpeta del proyecto un directorio denominado doc, en el que está especificada la documentación javadoc de todas las clases.

3.4.1 Clase UMLClass

Visual Paradigm Standard Edition(E.S)	
UML Class	
-name :	String
-visibility :	int
-isAbstract :	boolean
-vAttributes :	Vector
-vOperations :	Vector
+UMLClass()	: UMLClass
+setAbstract()	: void
+isAbstract()	: boolean
+loadGesXMLTemplate()	: void
+loadImportTemplate()	: void
+writeClass()	: void
+writeClassXML()	: void
+writeXMLHead()	: void

Almacena los datos de una clase UML, encapsula toda la información de un objeto (visibilidad, nombre, métodos, atributos...) a través de ella podemos modelar el entorno en estudio.

3.4.2 Clase UMLAttribute

Visual Paradigm Standard Edition(E.S)	
UML Attribute	
-name :	String
-type :	String
-vectorType :	String
-visibility :	int
-isstatic :	boolean
+UMLAttribute()	: UMLAttribute
+getStatic()	: boolean
+setStatic()	: void
+writeAttribute()	: void
+writeXMLAttribute()	: void

Clase que representa a un atributo de una clase UML. Un atributo es una característica de una clase. En esta clase se almacenan el nombre del atributo, la visibilidad y demás modificadores. En caso de que el atributo sea de tipo Vector, se tendrá en cuenta el atributo vectorType que almacenará el tipo de los elementos del vector.

3.4.3 Clase UMLMethod

Visual Paradigm Standard Edition(E.S)	
UML Method	
-name :	String
-parameters :	Vector
-typeReturn :	String
-visibility :	int
-isAbstract :	boolean
-isStatic :	boolean
+UMLMethod()	: void
+copyParameters()	: void
+getAbstract()	: boolean
+getStatic()	: boolean
+getType()	: String
+setAbstract()	: void
+setStatic()	: void
+setType()	: void
+writeMethodXML()	: void
+writeMethod()	: void

Clase que representa a un método de una clase UML. Los métodos describen el comportamiento de la clase. En esta clase se almacenan el nombre, los parámetros y el valor de retorno del método además de otros modificadores.

3.4.4 Clase UMLParameter

UMLParameter	
-name : String	
-type : String	
+UMLParameter() : UMLParameter	
+isReturnType() : boolean	
+writeParameter() : void	
+writeParameterXML() : void	

Clase que representa al parámetro de un método de una clase UML. Almacena el tipo de dato del parámetro y su nombre.

3.4.5 Clase UMLAssociation

UMLAssociation	
-id : String	
-multiplicity : Vector	
-vBeginEnd : Vector	
+UMLAssociation() : void	
+copyAllMembers() : void	
+printAssociation() : void	

Clase que representa una relación de asociación entre dos clases UML. En ella se almacena el nombre y multiplicidad de las clases conectadas por la relación. Así como el identificador de la relación que existe en el archivo xmi del que se recupera.

3.4.6 Clase UMLGeneralization

UMLGeneralization	
-id : String	
-vBeginEnd : Vector	
+UMLGeneralization() : void	
+copyAllMembers() : void	
+print() : void	

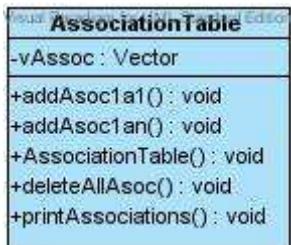
Representa una relación de herencia entre clases. En ella se almacenan los nombre de la clase padre e hija. Y también el identificador de la relación que existe en el archivo xmi del que se recupera.

3.4.7 Clase ClassContainer

ClassContainer	
-vClasses : Vector	
+ClassContainer() : void	
+addClass() : void	
+addAtb() : void	
+addIMethod() : void	
+consultAttribute() : Vector	
+consultClass() : Vector	
+consultOperation() : Vector	
+consultParameter() : Vector	
+createFileXMLProject() : String	
+createXMLFile() : Vector	
+deleteAllClases() : void	
+deleteMember() : void	
+existsAtb() : boolean	
+existsClass() : boolean	
+modifyAttribute() : void	
+modifyClass() : void	
+modifyOperation() : void	
+printAllClasses() : void	
+save() : void	
+unzip() : void	
+zip() : void	

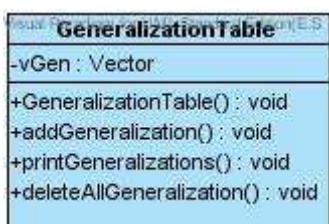
Clase que almacena todos los objetos UMLClass del sistema. Contiene todos los métodos de acceso, modificación de los elementos almacenados.

3.4.8 Clase AssociationTable



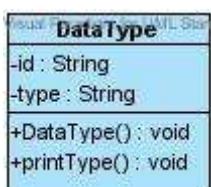
Clase que almacena todas las relaciones de asociación del diagrama de clases del sistema a generar. Contiene métodos para añadir y eliminar asociaciones.

3.4.9 Clase GeneralizationTable



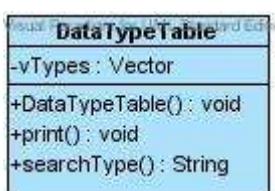
Clase que almacena todas las relaciones de herencia del diagrama de clases del sistema a generar. Contiene métodos para añadir y eliminar relaciones.

3.4.10 Clase DataType



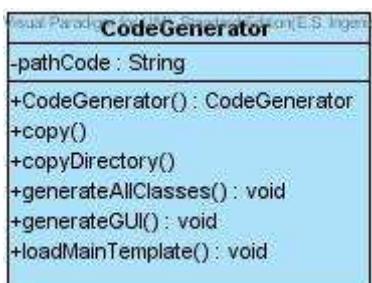
Clase que representa los tipos de datos que pueden existir en la aplicación. Almacena el nombre y un identificador de los tipos de datos primitivos y de las clases creadas

3.4.11 Clase DataTypeTable



Clase que almacena todos los DataType de la aplicación.

3.4.12 Clase CodeGenerator



Clase que se encarga de generar el código relativo a todas las clases del sistema, usando plantillas de texto almacenadas en disco. También genera el código de las clases de la interfaz gráfica de la aplicación.

3.4.13 Clase ParseXMI

Visual Paradigm for UML Standard Edition (E.S. Inc.)	
ParserXMI	
-rutaXMI : String	
-tdt : DataTypeTable	
+ParserXMI() : void	
+changeType() : String	
+getAllAssociations() : void	
+getAllGeneralizations() : void	
+getAllTypes() : void	
+getVisibility() : int	
+main()	
+readAssociations() : AssociationTable	
+readGeneralizations() : GeneralizationTable	
+readTypes() : DataTypeTable	
+readXMI() : ClassContainer	
+setXMIPath() : void	
+readXMLNode() : void	

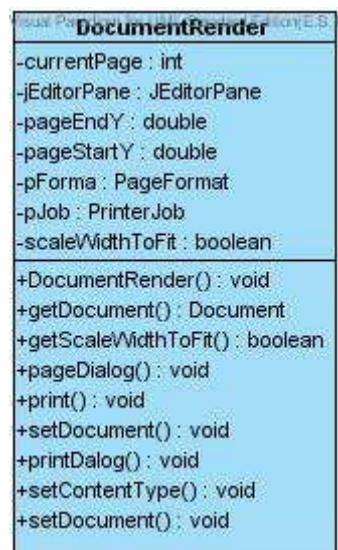
Clase que se encarga de leer el archivo xmi de entrada, tratando todos los nodos del árbol xmi, creando a partir de esta lectura las estructuras del modelo de la herramienta: el contendor de clases, la tabla de relaciones de herencia, la tabla de asociaciones y la tabla de tipos de datos.

3.4.14 Clase Controller

Visual Paradigm for UML Standard Edition (E.S. Inc.)	
Controller	
-asoc : AssociationTable	
-contenido : ClassContainer	
-defaultJavaPath : String	
-defaultSavePath : String	
-errorBienFormado : String	
-gener : GeneralizationTable	
-mainClass : String	
-vecTipo : String[]	
-vecTipoReturn : String[]	
+Controller() : void	
+addTypes() : void	
+addTypesWithVoid() : void	
+createAssociationTable() : void	
+createDirectory() : String	
+deleteAssociation() : void	
+getAllNodes() : void	
+getOnlyName() : String	
+isMethod() : boolean	
+loadData() : void	
+modifyAssociationTable() : void	
+newJavaFile() : String	
+openFile() : String	
+printAll() : void	
+printFile() : void	
+readBasicXML() : Vector	
+readCompletXML() : Vector	
+validateAllXML() : String	
+validateFile() : boolean	
+validatePath() : boolean	
+wellFormedXML() : boolean	

La clase Controller almacena referencias a las clases del modelo, recibe una solicitud de alguna de las ventanas del paquete de Vista y coordina su realización para hacer modificaciones en las estructuras del modelo que son las representadas en las clases ClassContainer, AssociationTable, GeneralizationTable...

3.4.15 Clase DocumentRender



La clase `DocumentRender` imprime objetos de tipo documento (atributos de texto, incluyendo fuentes, color, iconos...) calcula los saltos de línea, página, y realiza otros formateos.

3.5 Diagramas de clases parciales

3.5.1 Diagrama de clases parciales: Nuevo Proyecto

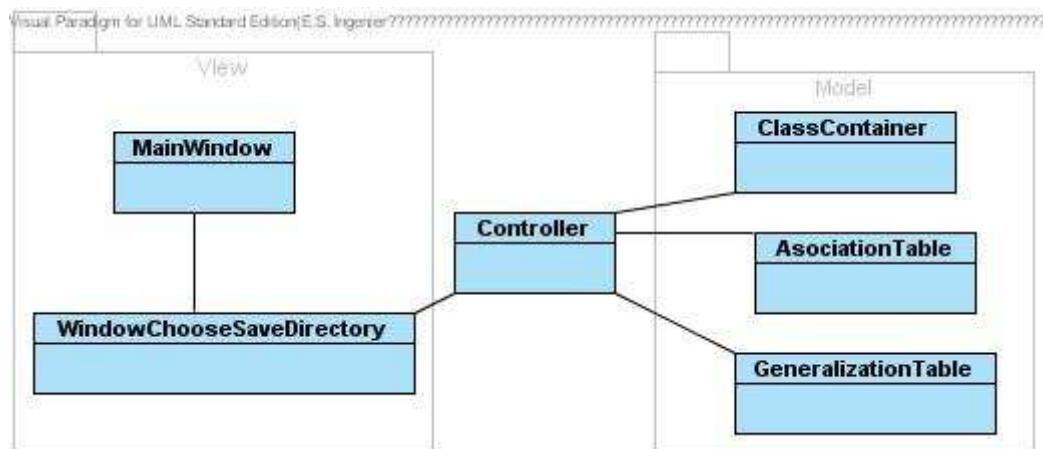


Ilustración 61: Diagrama de clases parciales - Nuevo Proyecto

3.5.2 Diagrama de clases parciales: Abrir Proyecto

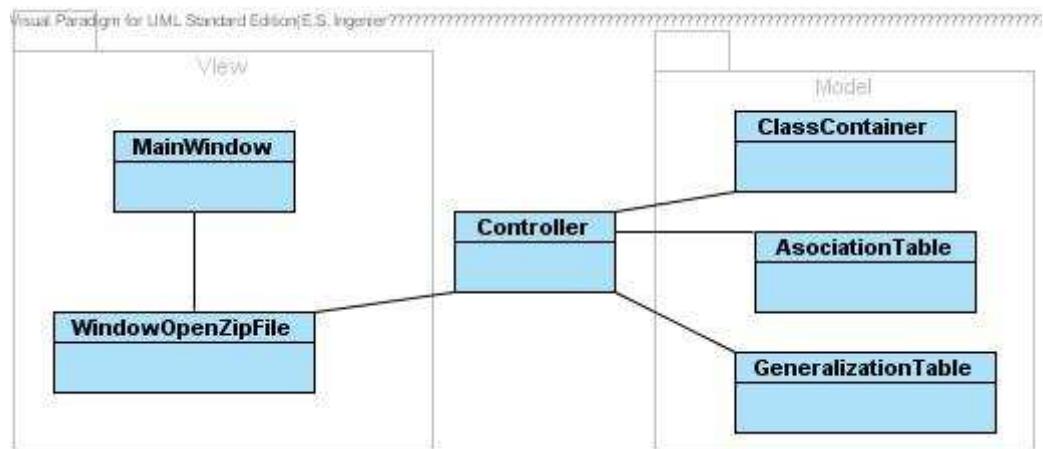


Ilustración 62: Diagrama de clases parciales - Abrir Proyecto

3.5.3 Diagrama de clases parciales: Importar archivo XMI

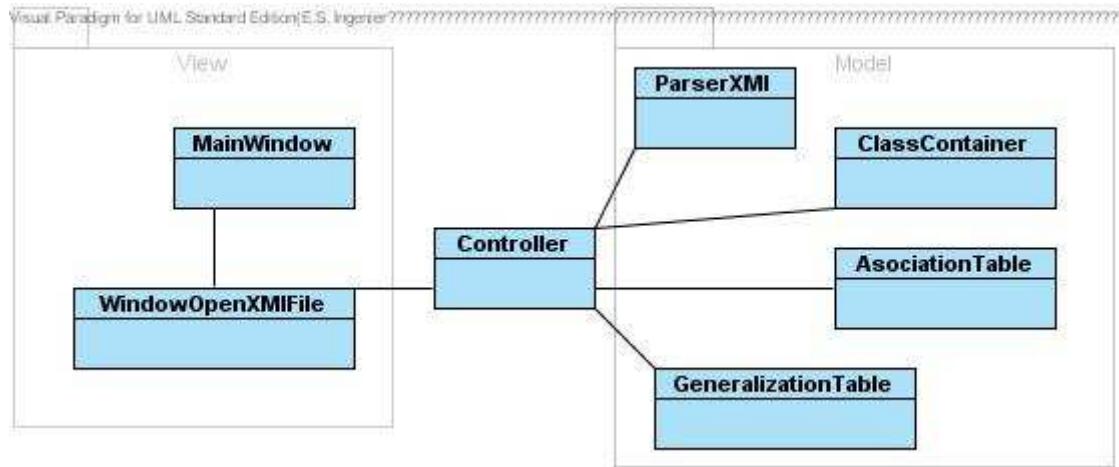


Ilustración 63: Diagrama de clases parciales - Importar archivo XMI

3.5.4 Diagrama de clases parciales: Generar código Java

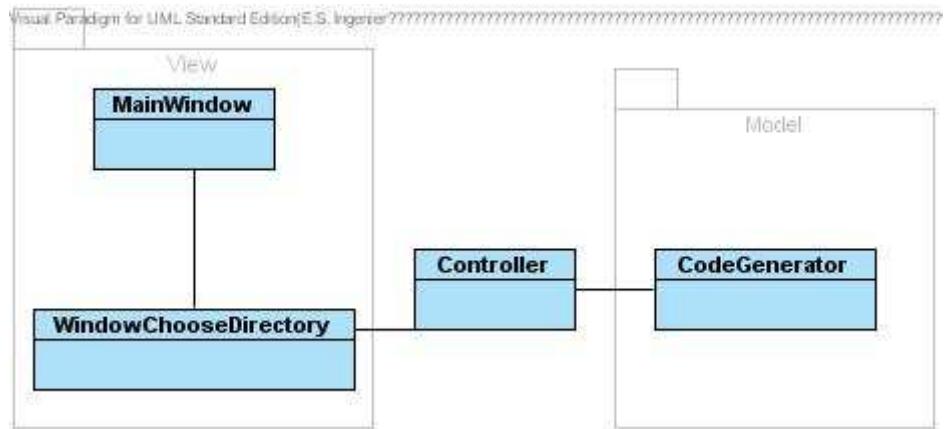


Ilustración 64: Diagrama de clases parciales - Generar código java

3.5.5 Diagrama de clases parciales: Guardar Proyecto

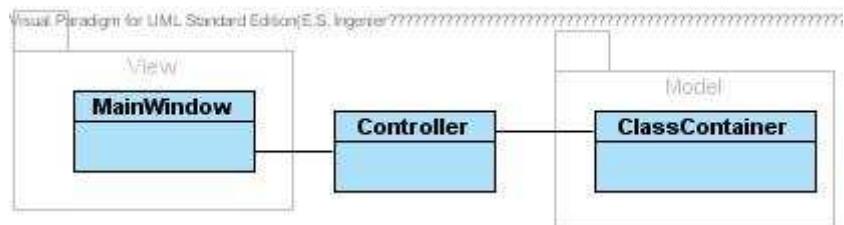


Ilustración 65: Diagrama de clases parciales - Guardar proyecto

3.5.6 Diagrama de clases parciales: Guardar Proyecto Como

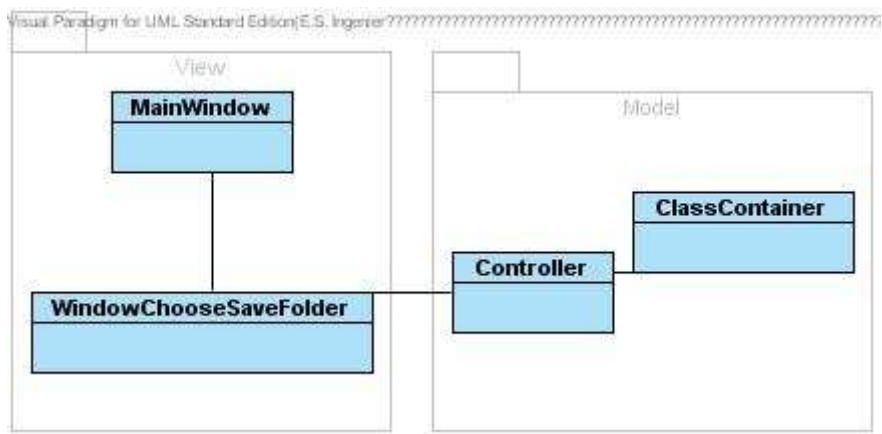


Ilustración 66: Diagrama de clases parciales - Guardar proyecto como

3.5.7 Diagrama de clases parciales: Borrar todos los elementos

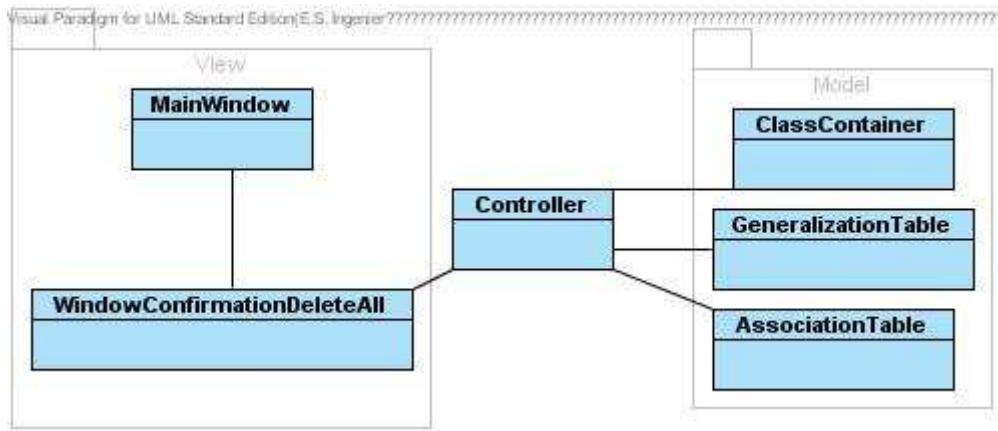


Ilustración 67: Diagrama de clases parciales - Borrar todos los elementos

3.5.8 Diagrama de clases parciales: Gestionar Clase

Añadir clase

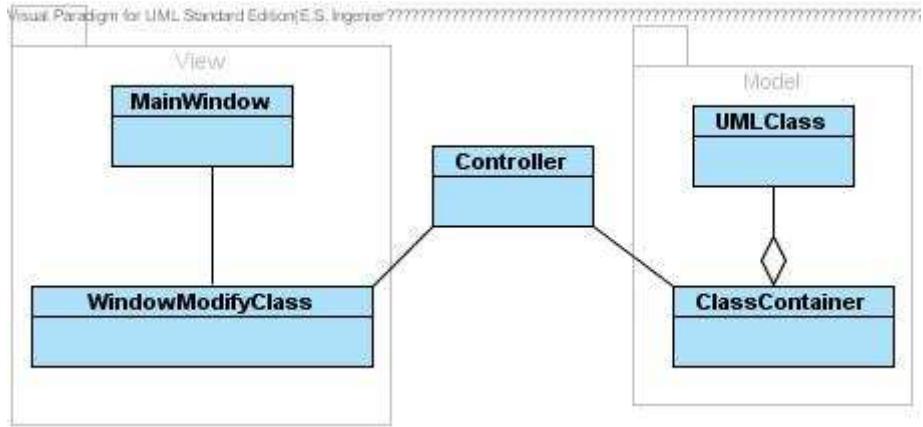


Ilustración 68: Diagrama de clases parciales - Añadir clase

Modificar Clase

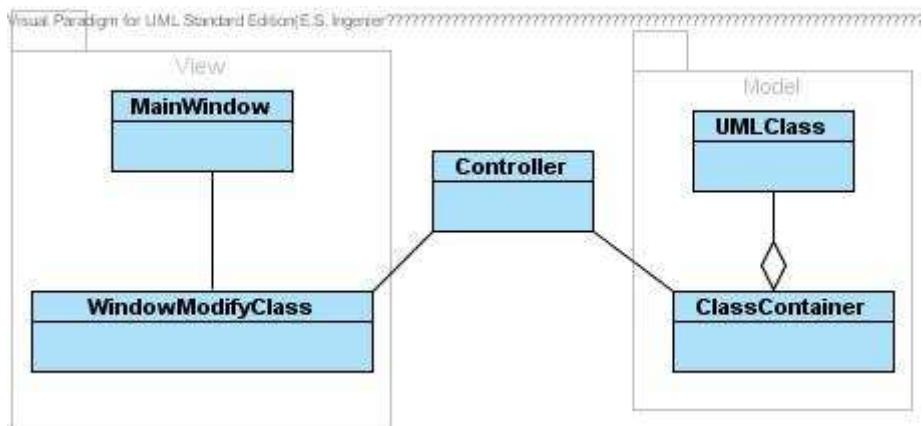


Ilustración 69: Diagrama de clases parciales - Modificar Clase

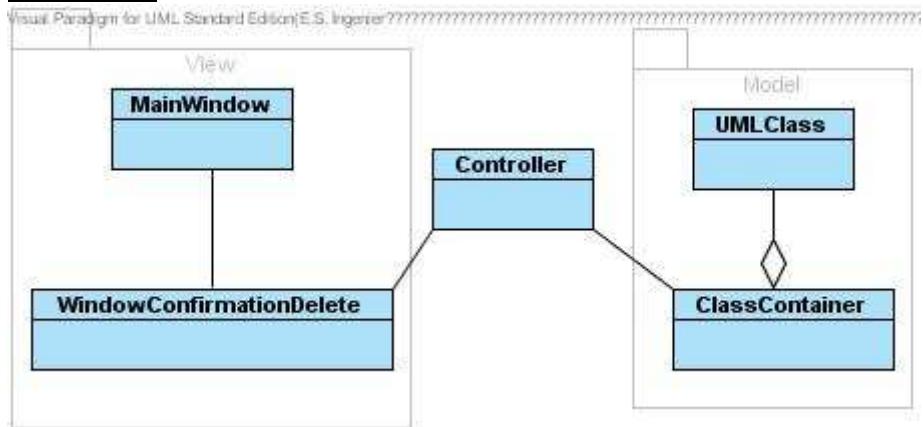
Borrar Clase

Ilustración 70: Diagrama de clases parciales - Borrar clase

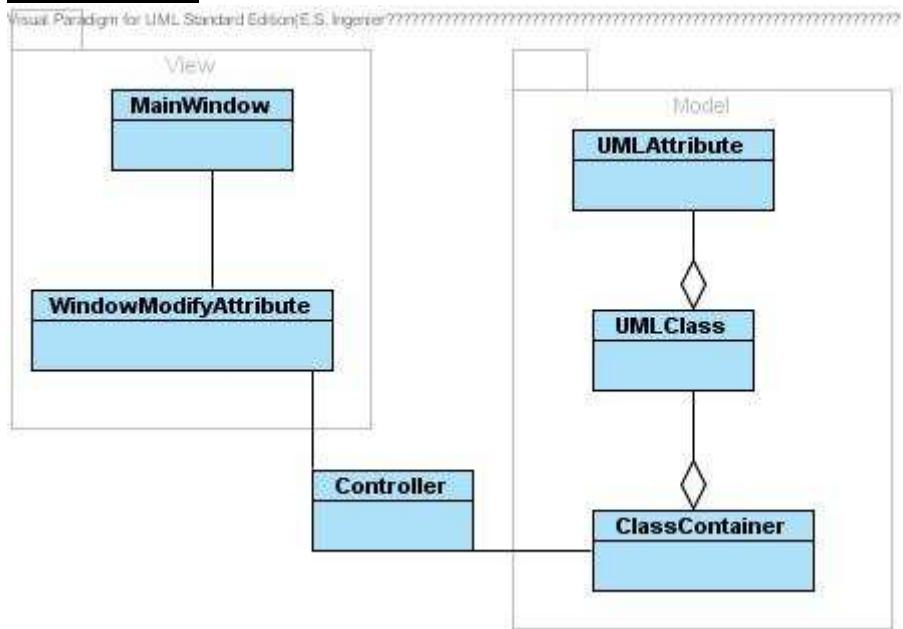
3.5.9 Diagrama de clases parciales: Gestionar Atributos**Añadir Atributo**

Ilustración 71: Diagrama de clases parciales - Añadir Atributo

Modificar Atributo

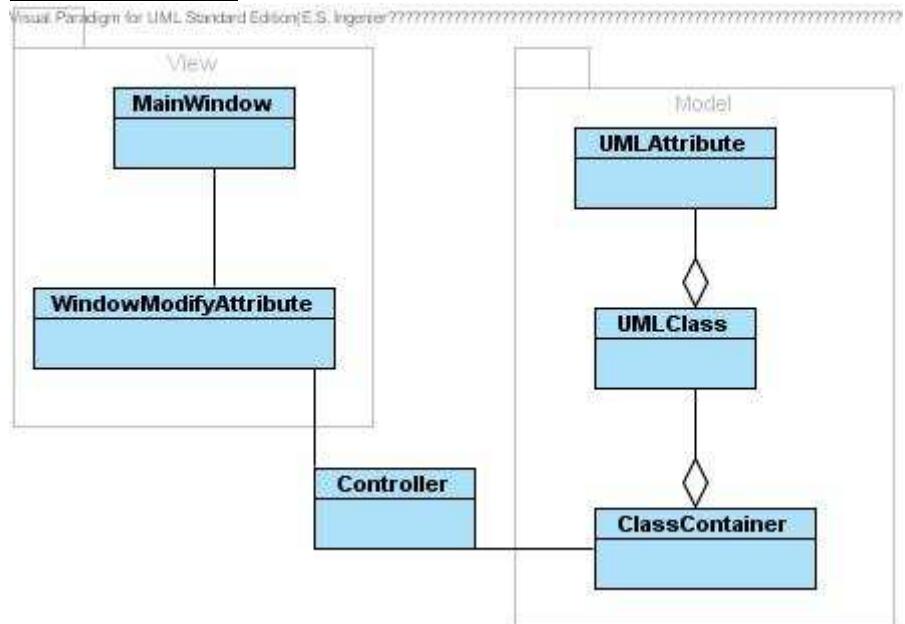


Ilustración 72: Diagrama de clases parciales - Modificar Atributo

Borrar Atributo

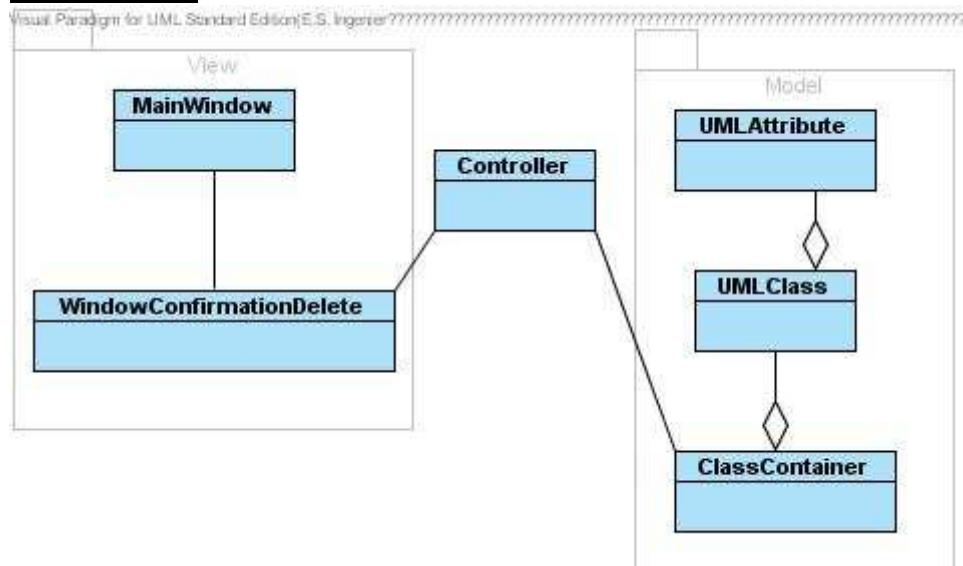


Ilustración 73: Diagrama de clases parciales - Borrar Atributo

3.5.10 Diagrama de clases parciales: Gestionar Métodos

Añadir Método

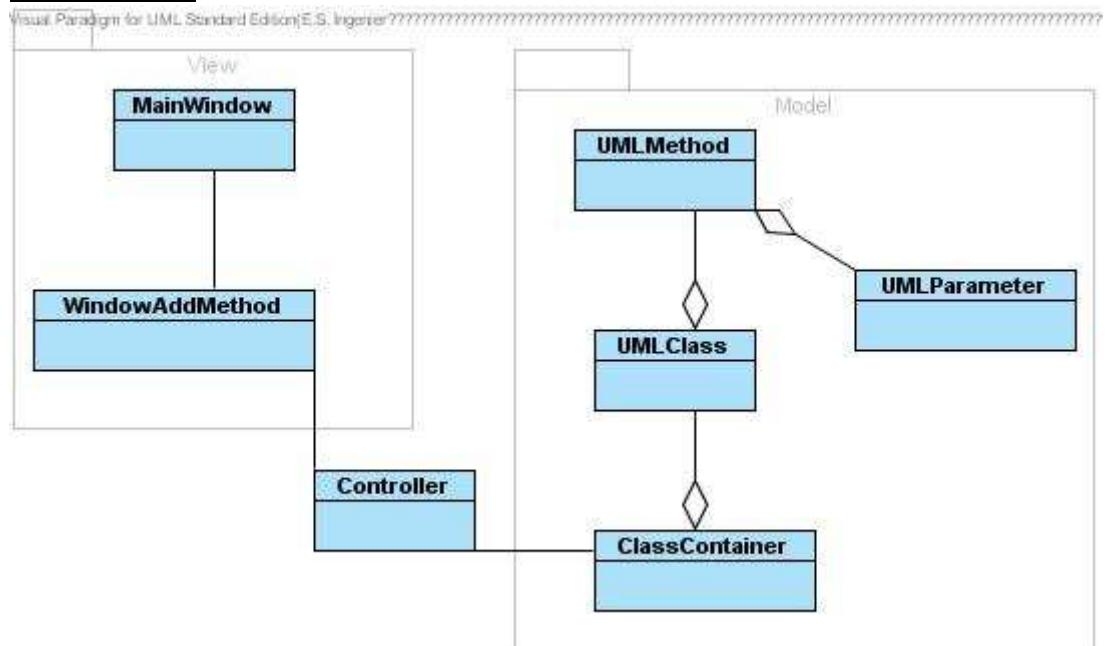


Ilustración 74: Diagrama de clases parciales - Añadir método

Modificar Método

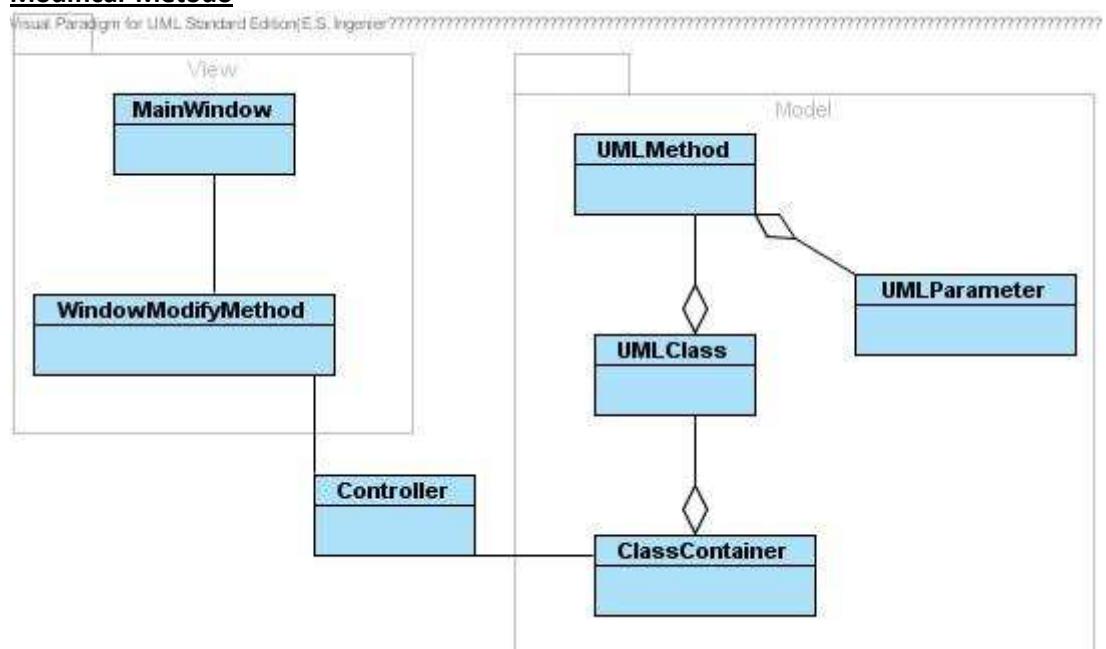


Ilustración 75: Diagrama de clases parciales - Modificar método

Borrar Método

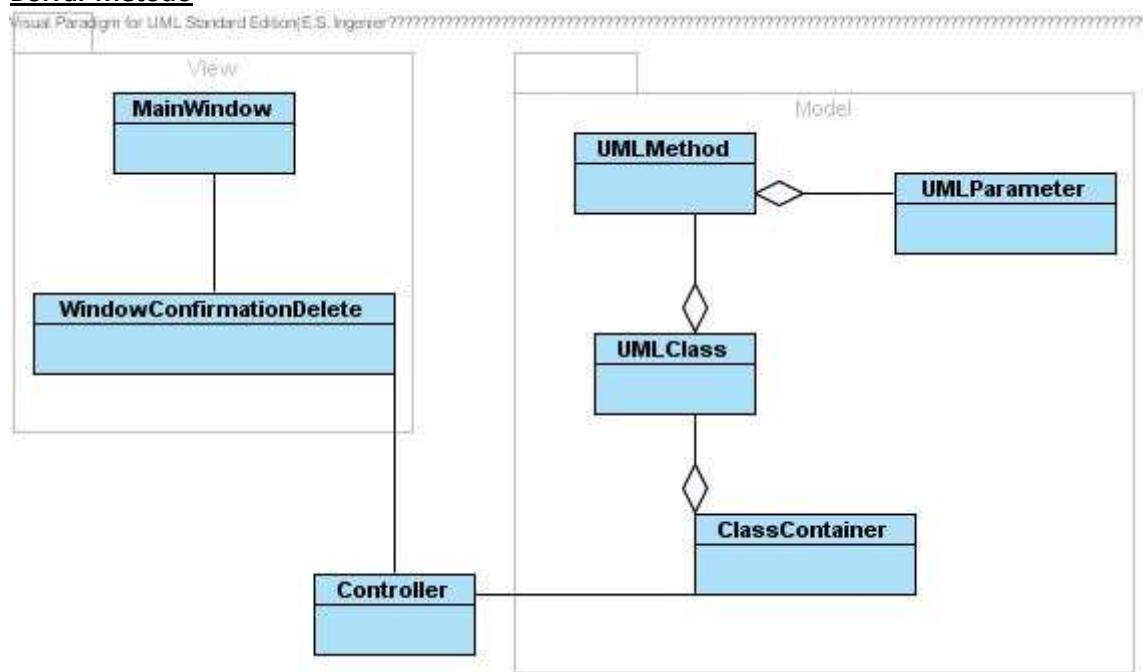


Ilustración 76: Diagrama de clases parciales: Borrar método

3.5.11 Diagrama de clases parciales: Nuevo Archivo Java

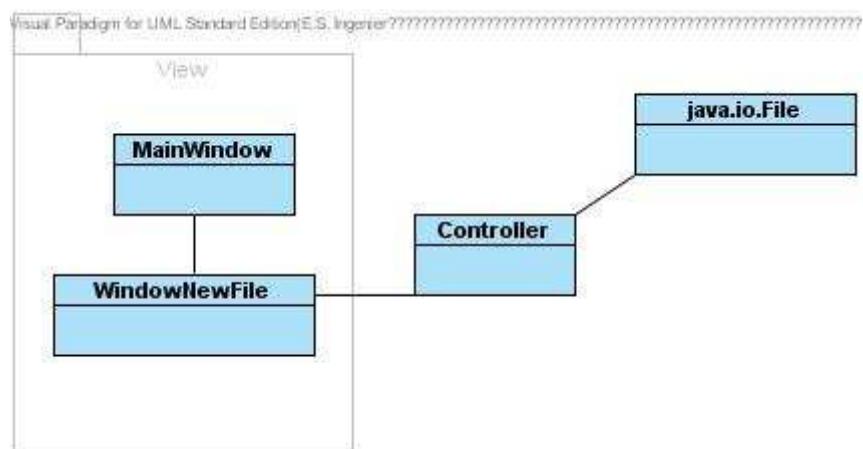


Ilustración 77: Diagrama de clases parciales - Nuevo Archivo Java

3.5.12 Diagrama de clases parciales: Abrir Archivo Java

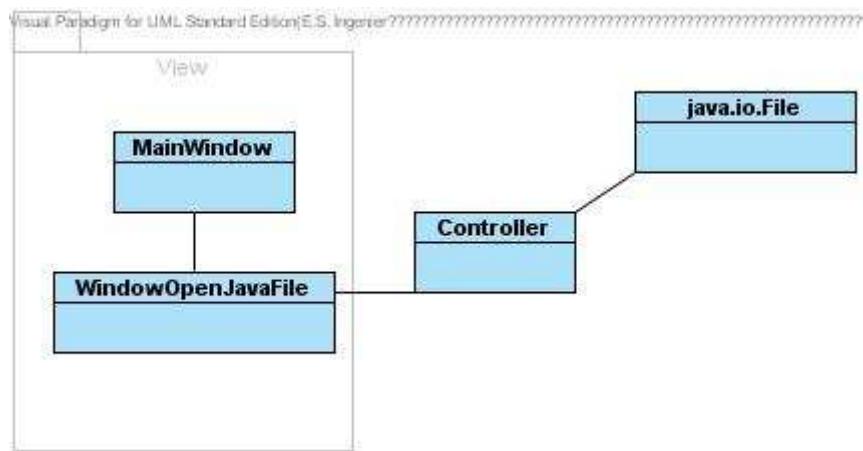


Ilustración 78: Diagrama de clases parciales - Abrir Archivo Java

3.5.13 Diagrama de clases parciales: Borrar archivo Java

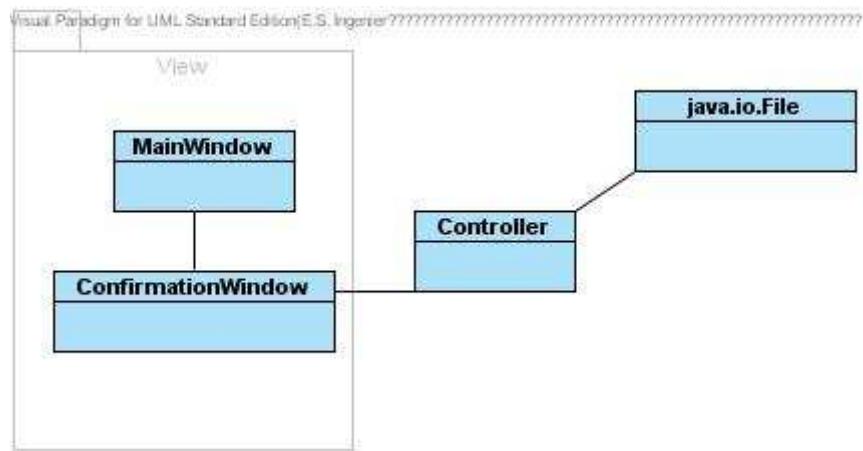


Ilustración 79: Diagrama de clases parciales - Borrar archivo Java

3.5.14 Diagrama de clases parciales: Guardar Todo

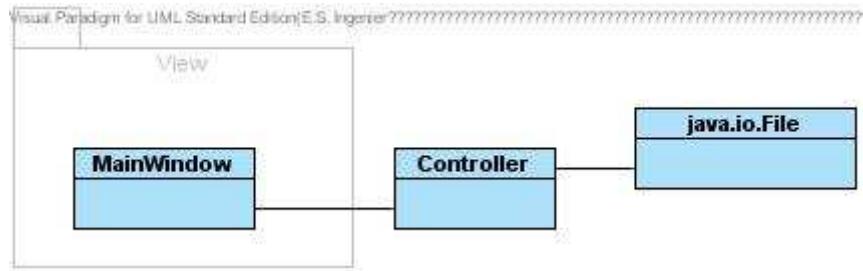


Ilustración 80: Diagrama de clases parciales - Guardar todo

3.5.15 Diagrama de clases parciales: Imprimir Archivo

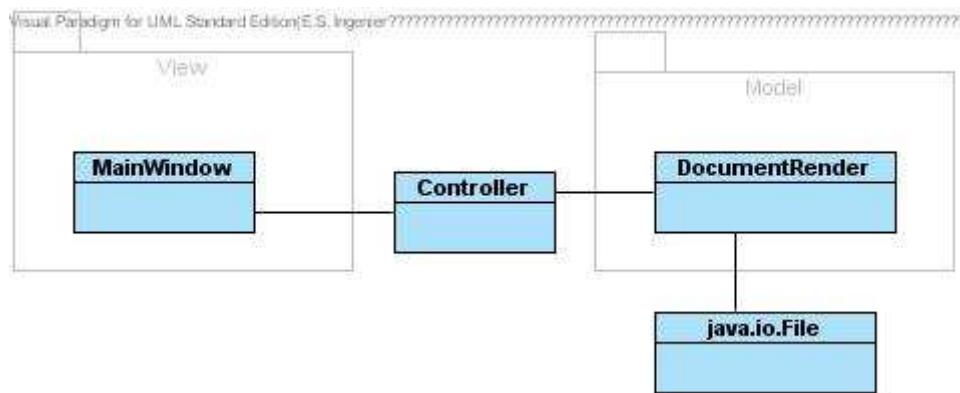


Ilustración 81: Diagrama de clases parciales - Imprimir archivo

3.5.16 Diagrama de clases parciales: Guardar Pestaña

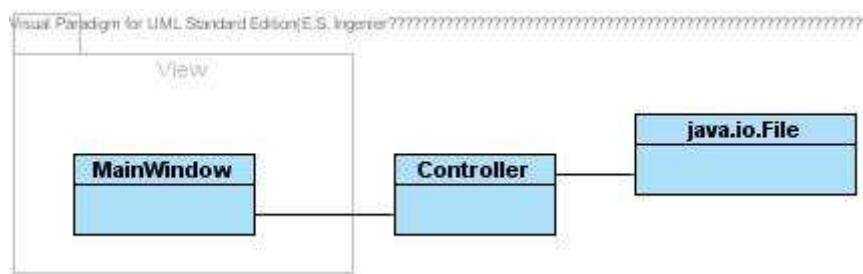


Ilustración 82: Diagrama de clases parcial - Guardar pestaña

4. IMPLEMENTACIÓN

En esta fase se estudiará la arquitectura para el sistema y se llevará a cabo la implementación del mismo.

4.1 Diagrama de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Los componentes representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas, pueden ser simples archivos, paquetes, interfaces o bibliotecas. Las relaciones de dependencia se utilizan en los diagramas de componentes para indicar que un componente utiliza los servicios ofrecidos por otro componente. Los paquetes están organizados para maximizar la coherencia interna dentro de cada paquete y minimizar el acoplamiento externo entre paquetes.

A continuación se muestra una vista general de los componentes de la aplicación.

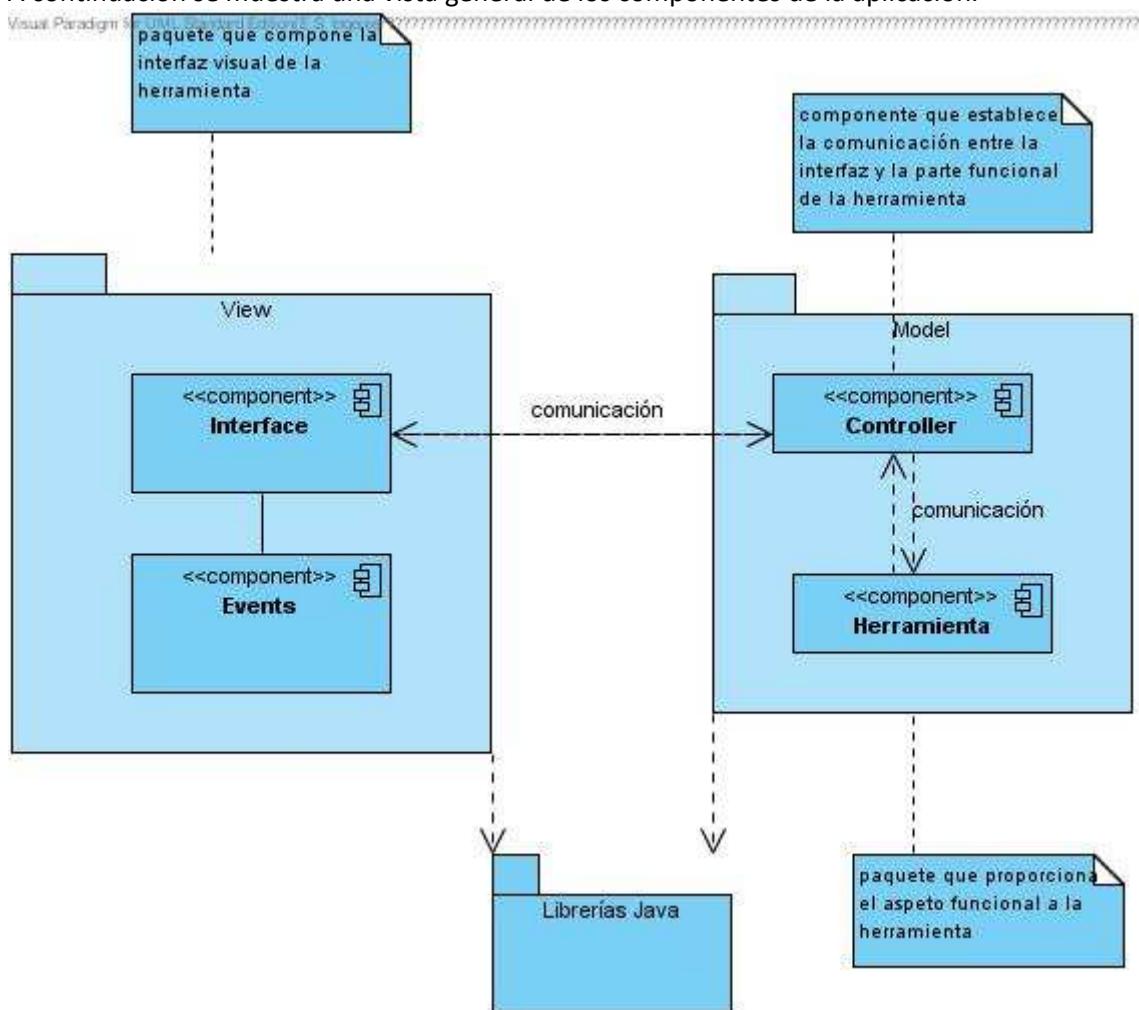


Ilustración 83: Diagrama de componentes

Cada uno de los elementos contiene la siguiente información:

Herramienta: Este componente representa al conjunto de clases que componen la parte lógica de la aplicación.

Vista: representa el conjunto de componentes que forman la Interfaz de Usuario de la Herramienta.

Interface: Elemento que representa a todos las ventanas de la interfaz de la aplicación.

Controller: Componente que se encarga de establecer la comunicación entre la interfaz y los componentes que aportan el aspecto funcional a la herramienta. De esta forma se separa el componente visual del funcional.

Eventos: conjunto de componentes que se encargan de capturar y gestionar los eventos que se producen en la interfaz de la Herramienta.

Librerías Java: Conjunto de componentes y librerías que la Herramienta necesita para su ejecución y funcionamiento.

4.2 Diagramas de despliegue

El diagrama de despliegue es un tipo de diagrama que se utiliza para modelar el hardware utilizado en la implementación del sistema y las relaciones entre sus componentes.

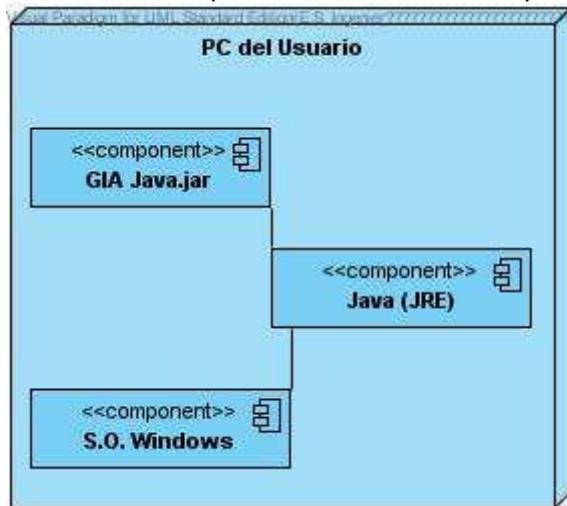


Ilustración 84: Diagrama de despliegue

Como puede verse el diagrama está compuesto por un único bloque que constituye el PC del usuario sobre el que se ejecuta la herramienta. Al ser una herramienta java podrá ser ejecutada en cualquier PC que disponga del entorno de ejecución Java (JRE). El sistema operativo sobre el que se ejecute la aplicación deberá ser Windows porque la herramienta hace uso de algunos recursos de este sistema como la línea de comandos o el explorador de archivos.

5. PRUEBAS

5.1 Pruebas del sistema

Las pruebas sobre el sistema se han realizado para descubrir errores no detectados con anterioridad, medir la fiabilidad y la calidad de la aplicación, así como garantizar un correcto funcionamiento del sistema, se han realizado una serie de pruebas sobre la aplicación durante toda la fase de desarrollo.

Estas pruebas permiten validar los siguientes aspectos:

- Comprobar la especificación de requerimientos y verificar que éstas coinciden con el diseño en las dos partes que componen el sistema. Prueba de que las funcionalidades para las que se diseñó se cumplen perfectamente tanto en la parte de diseño como en la de IDE de Java.
- Corrección del código implementado mediante el lenguaje de programación java, para que cumpla toda la funcionalidad.
- Asegurar que la herramienta genera aplicaciones con una interfaz agradable sencilla.
- Asegurar que el código generado es legible y fácil de entender.
- Garantizar robustez ante un uso equivocado por parte de los usuarios del sistema

Finalmente, se ha validado el sistema completo para evitar posibles efectos laterales.

5.2 Pruebas de software

Las pruebas de software detectan el correcto funcionamiento del sistema teniendo en cuenta el software utilizado. El sistema se ha probado con resultados óptimos en las siguientes condiciones:

- Sistema operativo: La herramienta está específicamente diseñada para el sistema operativo Windows. Se ha probado bajo Windows xp sin encontrar problemas.
- Plataforma Java de Ejecución (JRE): Se ha utilizado la última versión que había disponible al iniciar el desarrollo, esta versión es la (JRE) J2SE 6.0.

5.3 Pruebas de unidad

Este tipo de pruebas intenta verificar que todos los componentes internos del sistema funcionan correctamente. Para ello, se diseñan dos tipos de pruebas, primero las pruebas de caja blanca, y a continuación las pruebas de caja negra. Ambas son complementarias y están orientadas a descubrir distintos tipos de errores.

5.3.1 Pruebas de caja blanca

Las pruebas de Caja Blanca intentan verificar o hacer un examen minucioso de los distintos caminos lógicos del software, comprobando que se recorren todas las ramas o caminos posibles, por lo menos una vez. Este tipo de pruebas se realizan al mismo tiempo que el desarrollo del sistema. Las pruebas de Caja Blanca intentan chequear que:

- Se ejecutan por lo menos una vez todos los caminos independientes de cada módulo.
- Se ejecutan todas las decisiones lógicas en sus vértices verdadero y falso.
- Se ejecutan todos los bucles en sus límites.

5.3.2 Pruebas de caja negra

Las pruebas de caja negra comprueban la validez de los datos de entrada al sistema y que el sistema responda a ellos de la forma esperada.

En estas pruebas se realizarán los siguientes pasos:

1. Para cada dato introducido en el sistema se identificarán sus clases de equivalencia.
2. Una vez definidas estas clases se agruparán en clases válidas e inválidas, es decir, se definirán las condiciones por las cuales una determinada entrada de datos en un campo del sistema se considera válido o inválido.
3. Definir los casos de prueba en los que se proponen valores de entrada, se establecen los resultados esperados y se muestran los obtenidos realmente.

Para realizar estas pruebas de una forma más sencilla, en la primera tabla se reunirán los dos primeros pasos, entonces, al mismo tiempo que se identifican las clases de equivalencia con un número, se clasifican en válidas e inválidas. En las clases de equivalencia se tendrá en cuenta el tipo de dato que se debe insertar y en algunas ocasiones caracteres obligatorios que tienen que aparecer en los datos de entrada.

En la segunda tabla se definirán los casos de prueba, las clases de equivalencia que cumplen cada caso, los resultados esperados y los obtenidos realmente.

Debido a la coincidencia de los datos solicitados para los formularios de alta y modificación, estas pruebas se simplifican en un solo análisis para el ámbito de las inserciones.

5.3.2.1 Formulario Nuevo Proyecto

Clases de equivalencia

Condiciones de Entrada		Clases Equivalencia Válidas	Clases Equivalencia Inválidas
Nombre del archivo	Valor	1. alfanumérico	2. vacío 3. utilizar los caracteres /\?* "<>
	Longitud	4. la que tenga el nombre	5. no hay
Directorio de trabajo	Valor	6. alfanumérico con espacios y signos.	7. vacío 8. ruta incorrecta
	Longitud	9. la que tenga la ruta	10. no hay

Casos de prueba

Campo	Valor	Clases que cumple	Resultados Esperados	Resultados obtenidos
Nombre del archivo	nuevo	1	Aceptar	Aceptado
	nuevo.txt	1	Aceptar	Aceptado
		2	Mensaje de error	Error: Dato no rellenado
	Nuevo\cinco	3	Mensaje de error	Error: No se pueden usar los caracteres /\?* "<>
	Nuevo22?	3	Mensaje de error	Error: No se pueden usar los caracteres /\?* "<>
Directorio de trabajo	D:\Workspace	6	Aceptar	Aceptado
		7	Mensaje de error	Error: Dato obligatorio no rellenado.
	:\Workspace	8	Mensaje de error	Error: La ruta no es válida.

5.3.2.2 Formulario Abrir Proyecto

Clases de equivalencia

Condiciones de Entrada		Clases Equivalencia Válidas	Clases Equivalencia Inválidas
Ruta del archivo ZIP	Valor	1. alfanumérico con espacios y signos. 5. la que tenga la ruta	2. vacío 3. que no tenga extensión .Zip 4. que no exista el archivo 6. no hay
	Longitud		

Casos de prueba

Campo	Valor	Clases que cumple	Resultados Esperados	Resultados obtenidos
Ruta del archivo ZIP	C:\Proyecto\ nuevo.zip	1	Aceptar	Aceptado
	nuevo.zip	1	Aceptar	Aceptado siempre que el archivo exista en el directorio del proyecto.
		2	Mensaje de error	Error: Dato obligatorio no rellenado
	C:\Proyecto\ nuevo	3	Mensaje de error	Error: El archivo debe de tener extensión .Zip
	:\Proyecto\ nuevo.zip	4	Mensaje de error	Error: Debe introducir una ruta correcta.

5.3.2.3 Formulario Importar Proyecto

Clases de equivalencia

Condiciones de Entrada		Clases Equivalencia Válidas	Clases Equivalencia Inválidas
Ruta del archivo XMI	Valor	1. alfanumérico con espacios y signos. 5. la que tenga la ruta	2. vacío 3. que no tenga extensión .xmi 4. que no exista el archivo 6. no hay
	Longitud		

Casos de prueba

Campo	Valor	Clases que cumple	Resultados Esperados	Resultados obtenidos

Ruta del archivo XMI	C:\Archivos\ nuevo.xmi	1	Aceptar	Aceptado
	nuevo.xmi	1	Aceptar	Aceptado siempre que el archivo exista en el directorio del proyecto.
		2	Mensaje de Error	Error: Dato obligatorio no rellenado
	C:\Proyecto\nuevo	3	Mensaje de Error	Error: El archivo debe de tener extensión .xmi
	:\Proyecto\nuevo.xmi	4	Mensaje de Error	Error: Debe introducir una ruta correcta.

5.3.2.4 Formulario de Preferencias

Clases de equivalencia

Condiciones de Entrada		Clases Equivalencia Válidas	Clases Equivalencia Inválidas
Directorio de trabajo	Valor	1. alfanumérico con espacios y signos.	2. vacío
	Longitud	3. la que tenga el nombre	4. no hay
Directorio de archivos generados	Valor	5. alfanumérico con espacios y signos.	6. vacío
	Longitud	7. la que tenga la ruta	8. no hay

Casos de prueba

Campo	Valor	Clases que cumple	Resultados Esperados	Resultados obtenidos
Directorio de trabajo	D:\Generated GIA files	1	Aceptar	Aceptado
	Nueva\miProyecto	1	Aceptar	Aceptado
		2	Mensaje de error	Error: Dato no rellenado
Directorio de archivos generados	D:\Workspace	5	Aceptar	Aceptado
		6	Mensaje de error	Error: Dato obligatorio no rellenado.
	Nuevo\Workspace	5	Aceptar	Aceptado

5.3.2.5 Formulario Guardar Como

Clases de equivalencia

Condiciones de Entrada		Clases Equivalencia Válidas	Clases Equivalencia Inválidas
Nombre del archivo	Valor	1. alfanumérico	2. vacío 3. utilizar los caracteres /\?* "<>
	Longitud	4. la que tenga el nombre	5. no hay
Directorio de trabajo	Valor	6. alfanumérico con espacios y signos.	7. vacío 8. ruta incorrecta
	Longitud	9. la que tenga la ruta	10. no hay

Casos de prueba

Campo	Valor	Clases que cumple	Resultados Esperados	Resultados obtenidos
Nombre del archivo	nuevo	1	Aceptar	Aceptado
	nuevo.txt	1	Aceptar	Aceptado
		2	Mensaje de error	Error: Dato no rellenado
	Nuevo\cinco	3	Mensaje de error	Error: No se pueden usar los caracteres /\?* "<>
	Nuevo22?	3	Mensaje de Error	Error: No se pueden usar los caracteres /\?* "<>
Directorio de trabajo	D:\Workspace	6	Aceptar	Aceptado
		7	Mensaje de error	Error: Dato obligatorio no rellenado.
	:\Workspace	8	Mensaje de error	Error: La ruta no es válida.

5.3.2.6 Formulario Generar Código

Clases de equivalencia

Condiciones de Entrada		Clases Equivalencia Válidas	Clases Equivalencia Inválidas
Ruta del directorio en el que se generan los archivos java	Valor	1. alfanumérico con espacios y signos.	2. vacío 3. que la ruta sea incorrecta
	Longitud	4. la que tenga la ruta	5. no hay

Casos de prueba

Campo	Valor	Clases que cumple	Resultados Esperados	Resultados obtenidos
Ruta del directorio en el que se generan los archivos java	C:\Generated Files	1	Aceptar	Aceptado
		2	Mensaje de Error	Error: Dato obligatorio no rellenado
	:\Generated files	3	Mensaje de Error	Error: Debe introducir una ruta correcta.

5.3.2.7 Formulario Añadir/Modificar Clase

Clases de equivalencia

Condiciones de Entrada		Clases Equivalencia Válidas	Clases Equivalencia Inválidas
Nombre de la clase	Valor	1. alfanumérico sin espacios	2. vacío 3. empezar por un número 4. contener espacios 5. el nombre de la clase ya existe en el sistema
	Longitud	6. >3 caracteres	7. < 3 caracteres
visibilidad	Valor	8. grupo de RadioButton: private, public o protected	9. no hay porque siempre hay un campo seleccionado por defecto.
	Longitud	10. no definida	11. no hay

Abstracta	Valor	12. checkBox: activado o desactivado	13. No hay porque activado o desactivado el valor es válido.
	Longitud	14. No definida	15. No hay

Casos de prueba

Campo	Valor	Clases que cumple	Resultados Esperados	Resultados obtenidos
Nombre de la clase	Padre	1	Aceptar	Aceptado
	Padre_adoptivo	1	Aceptar	Aceptado
		2	Mensaje de error	Error: Dato no rellenado
	9Padre	3	Mensaje de error	Error: El nombre de una clase no puede empezar por un número
	Padre adoptivo	4	Mensaje de error	Error: El nombre de una clase no puede tener espacios
	pad	7	Mensaje de error	Error: El nombre de una clase debe de tener más de 3 caracteres.
visibilidad	Seleccionar public	8	Aceptar	Aceptado
Modificador abstract	Activar check abstract	12	Aceptar	Aceptado

5.3.2.8 Formulario Añadir/Modificar Atributo

Clases de equivalencia

Condiciones de Entrada		Clases Equivalencia Válidas	Clases Equivalencia Inválidas
Nombre del atributo	Valor	1. alfanumérico sin espacios	2. vacío 3. empezar por un número 4. contener espacios 5. atributo que ya existe para esa clase.
	Longitud	6. >3 caracteres	7. <3 caracteres
Visibilidad	Valor	8. grupo de RadioButton: private, public o protected	9. no hay porque siempre hay un campo seleccionado por defecto.

	Longitud	10. no definida	11. no hay
Modificador static	Valor	12. checkBox: activado o desactivado	13. no hay porque activado o desactivado el valor es válido.
	Longitud	14. No definida	15. no hay
Tipo de dato	Valor	16. comboBox: elección de un elemento de la lista.	17. No hay, siempre hay un elemento de la lista seleccionado.
	Longitud	18. no definida	19. no hay

Casos de prueba

Campo	Valor	Clases que cumple	Resultados Esperados	Resultados obtenidos
Nombre del atributo	ruedas	1	Aceptar	Aceptado
	numero_ruedas	1	Aceptar	Aceptado
		2	Mensaje de error	Error: Dato no rellenado
	4ruedas	3	Mensaje de error	Error: El nombre de una clase no puede empezar por un número
	numero ruedas	4	Mensaje de error	Error: El nombre de un atributo no puede tener espacios
	num		Mensaje de error	Error: la longitud del nombre del atributo debe de ser mayor de 3 caracteres.
Visibilidad	Seleccionar public	8	Aceptar	Aceptado
Modificador static	Activar check abstract	12	Aceptar	Aceptado
Tipo de dato	Seleccionar int	16	Aceptar	Aceptado

5.3.2.9 Formulario Añadir/Modificar Método

Clases de equivalencia

Condiciones de Entrada	Clases Equivalencia Válidas	Clases Equivalencia Inválidas
------------------------	-----------------------------	-------------------------------

Nombre del método	Valor	1. alfanumérico sin espacios	2. vacío 3. empezar por un número 4. contener espacios
	Longitud	5. la que tenga el nombre	6. no hay
Visibilidad	Valor	7. grupo de RadioButton: private, public o protected	8. no hay porque siempre hay un campo seleccionado por defecto.
	Longitud	9. no definida	10. no hay
Modificador abstract	Valor	11. checkBox: activado o desactivado	12. no hay porque activado o desactivado el valor es válido.
	Longitud	13. no definida	14. no hay
Modificador static	Valor	15. checkBox: activado o desactivado	16. no hay porque activado o desactivado el valor es válido.
	Longitud	17. no definida	18. no hay
Tipo de retorno	Valor	19. comboBox: elección de un elemento de la lista.	20. no hay, siempre hay un elemento de la lista seleccionado.
	Longitud	21. no definida	22. no hay
Número de parámetros	Valor	23. >=0	24. <0 25. vacio 26. valor no numérico
	Longitud	26. no definida	27. no hay

Casos de prueba

Campo	Valor	Clases que cumple	Resultados Esperados	Resultados obtenidos
Nombre del método	agrupar	1	Aceptar	Aceptado
	agrupar_por_signo	1	Aceptar	Aceptado
		2	Mensaje de Error	Error: Dato no rellenado
	5agrupar	3	Mensaje de Error	Error: El nombre de un método no puede empezar por un número.
	agrupar por signo	4	Mensaje de Error	Error: El nombre de un método no puede tener espacios

Visibilidad	Seleccionar public	7	Aceptar	Aceptado
Modificador static	Seleccionar check static	11	Aceptar	Aceptado
Modificador abstract	Seleccionar check abstract	15	Aceptar	Aceptado
Tipo de retorno	Seleccionar void	19	Aceptar	Aceptado
Número de parámetros	2	23	Aceptar	Aceptado
	-1	24	Error	El valor del número de parámetros debe de estar entre (0-9)
		25	Error	No puede haber ningún campo vacío
	t	26	Error	El número de parámetros debe de estar entre (0-9)

5.3.2.10 Formulario Añadir/Modificar Parámetro

Clases de equivalencia

Condiciones de Entrada		Clases Equivalencia Válidas	Clases Equivalencia Inválidas
Nombre del parámetro	Valor	1. alfanumérico sin espacios	2. vacío 3. empezar por un número 4. contener espacios 5. no puede haber dos nombres de parámetro iguales para el mismo método.
	Longitud	5. la que tenga el nombre	6. no hay
Tipo del parámetro	Valor	7. comboBox: elección de un elemento de la lista.	8. no hay porque siempre hay un campo seleccionado por defecto.
	Longitud	9. no definida	10. no hay

Casos de prueba

Campo	Valor	Clases que cumple	Resultados Esperados	Resultados obtenidos
Nombre del parámetro	x	1	Aceptar	Aceptado

	nuevo_num	1	Aceptar	Aceptado
		2	Mensaje de Error	Error: Dato no rellenado
	3x	3	Mensaje de Error	Error: El nombre de un parámetro no puede empezar por un número
	nuevo valor	4	Mensaje de Error	Error: El nombre de un parámetro no puede tener espacios
Tipo del parámetro	Seleccionar private	7	Aceptar	Aceptado

5.3.2.11 Formulario Añadir Archivo java

Clases de equivalencia

Condiciones de Entrada		Clases Equivalencia Válidas	Clases Equivalencia Inválidas
Ruta del archivo Java	Valor	1. Alfanumérico con espacios y signos.	2. vacío 3. que no tenga extensión .java 4. que no exista el archivo
	Longitud	4. la que tenga la ruta	5. no hay

Casos de prueba

Campo	Valor	Clases que cumple	Resultados Esperados	Resultados obtenidos
Ruta del archivo Java	C:\Proyecto\Notas.java	1	Aceptar	Aceptado
	Notas.java	1	Aceptar	Aceptado siempre que el archivo exista en el directorio del proyecto.
		2	Mensaje de Error	Error: Dato obligatorio no rellenado
	C:\Proyecto\nuevo	3	Mensaje de Error	Error: El archivo debe de tener extensión .java
	:\\Proyecto\\nuevo.java	3	Mensaje de Error	Error: Debe introducir una ruta correcta.

5.3.2.12 Formulario Nuevo Archivo java

Clases de equivalencia

Condiciones de Entrada		Clases Equivalencia Válidas	Clases Equivalencia Inválidas
Nombre del archivo	Valor	1. Alfanumérico 2. vacío 3. utilizar los caracteres \?* <>	5. no hay
	Longitud	4. la que tenga el nombre	

Casos de prueba

Campo	Valor	Clases que cumple	Resultados Esperados	Resultados obtenidos
Nombre del archivo	nuevo	1	Aceptar	Aceptado
		2	Mensaje de Error	Error: Dato no rellenado
	Nuevo\cinco	3	Mensaje de Error	Error: No se pueden usar los caracteres \?* <>
	Nuevo22?	3	Mensaje de Error	Error: No se pueden usar los caracteres \?* <>

6. ANÁLISIS Y EJEMPLOS DE LA APLICACIÓN GENERADA

A continuación se muestra el diagrama de caso genérico de la aplicación generada, así como su diagrama de clases. Se explican las clases que son comunes a todas las aplicaciones generadas.

6.1 Diagrama de casos de uso de la aplicación generada

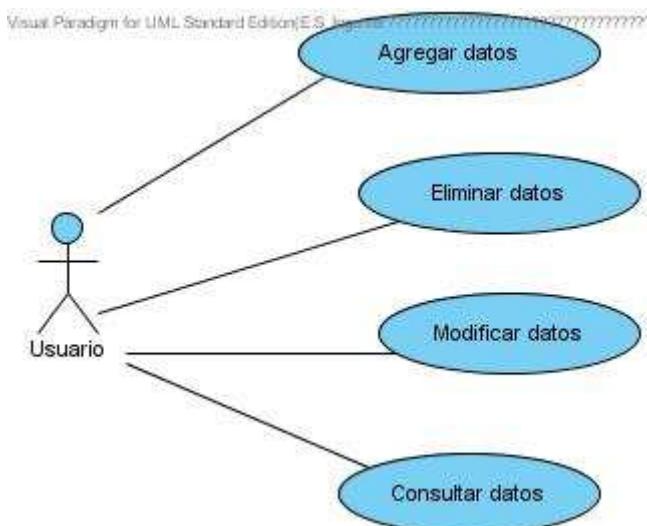


Ilustración 85: Diagrama de casos de uso de la aplicación generada

[Agregar datos](#): El usuario podrá añadir los datos de un objeto de una clase que se almacenarán en un archivo XML con el nombre de la clase.

[Eliminar datos](#): El usuario podrá eliminar los datos de un objeto de una clase almacenado en el XML de la clase. Los datos se borrarán de la interfaz además del XML.

[Modificar datos](#): El usuario podrá modificar los datos de un objeto de una clase almacenados en un archivo XML. Los datos se modificarán en el XML y se volverán a cargar en la interfaz de usuario.

[Consultar datos](#): El usuario podrá modificar los datos de un objeto de una clase leyéndolos de un archivo XML con el nombre de la clase.

6.2 Diagrama de clases de la aplicación generada

A continuación se muestra la estructura de información del sistema, las clases y las relaciones entre clases del proyecto generado.

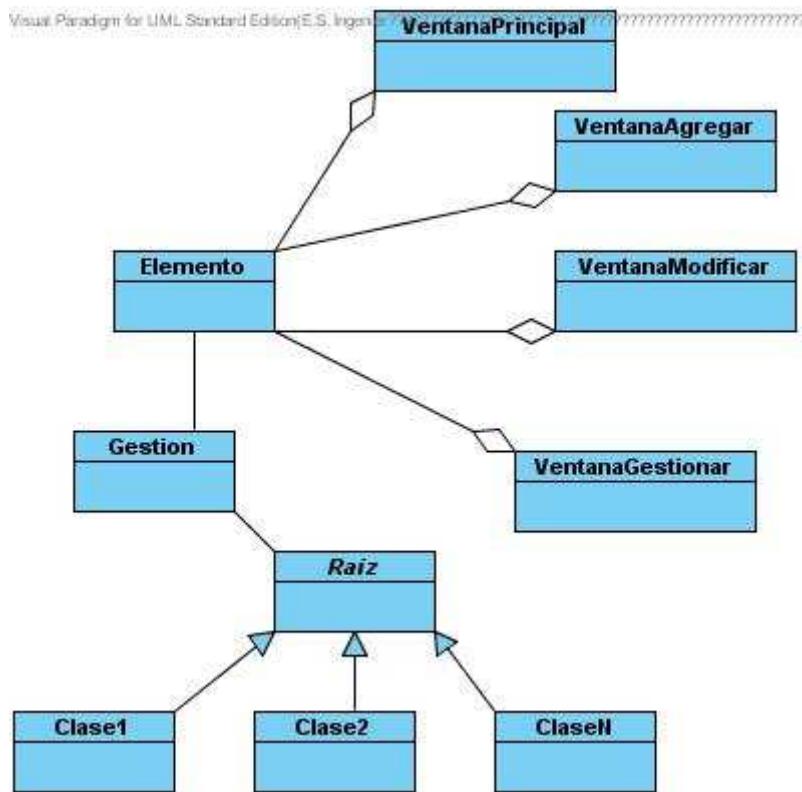


Ilustración 86: Diagrama de clases de la aplicación generada

Clase VentanaPrincipal: Ventana desde la que se accede a los datos de la clase que se ha marcado en la aplicación GiA Java como principal.

Clase VentanaAgregar: Ventana que implementa la interfaz que permite añadir un nuevo objeto a una clase de un proyecto.

Clase VentanaModificar: Ventana que implementa la interfaz que permite modificar los datos de un objeto de una clase de un proyecto.

Clase VentanaGestionar: Ventana desde la que se gestiona una lista de objetos que intervienen en una relación 1 a n.

Clase Elemento: Panel que representa un panel formado por dos elementos: una etiqueta (atributo de una clase) y un cajetín para introducir los datos de cualquier atributo de la clase. Cada ventana de la interfaz gráfica contiene Elementos.

Clase Gestion: Clase que almacena las operaciones (alta, baja y modificación) sobre los objetos de cualquier clase que derive de la clase Raíz.

Clase Raiz: Clase abstracta que almacena el comportamiento común a todas las clases del sistema creadas en el modelo de datos de la aplicación.

Clases1..n: Representa a cualquier clase creada en el modelo de la aplicación GiA Java.

6.3 Ejemplos

A continuación se mostrarán varios ejemplos de proyectos creados con la herramienta GIA Java. Se ilustrarán todo el proceso desde el modelado hasta la edición del código generado.

6.3.1 Ejemplo Agenda de contactos

Siguiendo el proceso habitual debemos de crear con ArgoUML un diagrama de clases de toda la aplicación. El sistema que se pretende modelar es una aplicación que permita gestionar los contactos de una agenda personal.

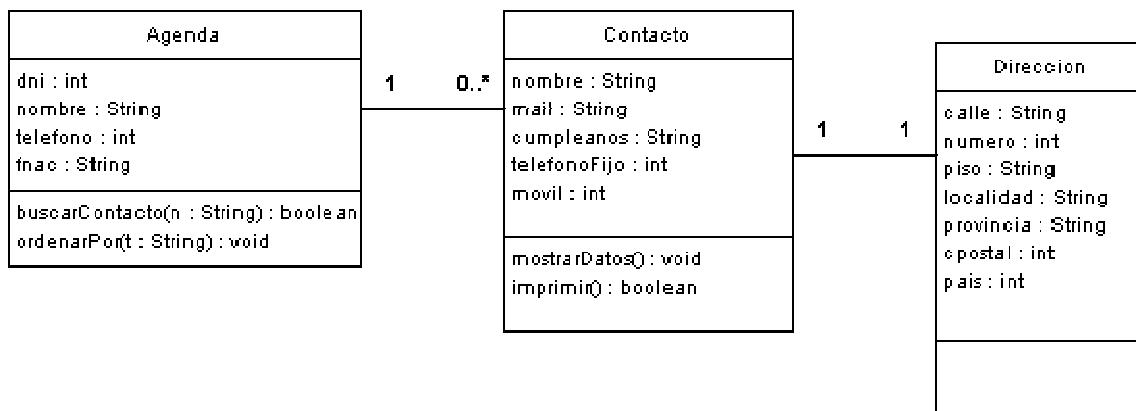


Ilustración 87: Diagrama de clases Agenda - Modelado con ArgoUML

Una vez creado el diagrama de clases se exporta a formato XMI, mediante la opción de Argo Archivo - Exportar proyecto como archivo XMI.

A continuación mostramos una imagen en la que se puede ver una pequeña parte del archivo xmi exportado. Si se mira con detalle podemos ver la estructura del xmi. Todos los elementos están englobados en un elemento raíz denominado UML:Model. De este elemento son hijos directos los elementos uml:Class, en el trozo de código se puede comprobar que se muestra la clase Agenda en el atributo name de uml:Class. También se muestra en el elemento UML:Attribute el atributo dni. Y en el elemento UML:Operation el primer método buscarContacto.

```

<?xml version = '1.0' encoding = 'UTF-8' ?>
<XMI xmi.version = '1.2' xmlns:UML = 'org.omg.xmi.namespace.UML' tim
<XMI.header>      <XMI.documentation>
    <XMI.exporter>ArgoUML (using Netbeans XMI Writer version 1.0)<
    <XMI.exporterVersion>0.24(5) revised on $Date: 2006-11-06 19:5
</XMI.documentation>
<XMI.metamodel xmi.name="UML" xmi.version="1.4"/></XMI.header>
<XMI.content>
<UML:Model xmi.id = '10-100-38--23-4866c288:11d1ae9d973:-8000:00
    name = 'untitledModel' isSpecification = 'false' isRoot = 'fal
    isAbstract = 'false'>
<UML:Namespace.ownedElement>
    <UML:Class xmi.id = '10-100-38--23-4866c288:11d1ae9d973:-800
        name = 'Agenda' visibility = 'public' isSpecification = 'f
        isLeaf = 'false' isAbstract = 'false' isActive = 'false'>
        <UML:Classifier.feature>
            <UML:Attribute xmi.id = '10-100-38--23-4866c288:11d1ae9d
                name = 'dni' visibility = 'protected' isSpecification
                changeability = 'changeable' targetScope = 'instance'>
                <UML:StructuralFeature.multiplicity>
                    <UML:Multiplicity xmi.id = '10-100-50-102-a05257d:11
                        <UML:Multiplicity.range>
                            <UML:MultiplicityRange xmi.id = '10-100-50-102-a
                                lower = '1' upper = '1'>
                            </UML:Multiplicity.range>
                        </UML:Multiplicity>
                    </UML:StructuralFeature.multiplicity>
                    <UML:StructuralFeature.type>
                        <UML:DataType xmi.idref = '10-100-38--23-4866c288:11
                        </UML:StructuralFeature.type>
                </UML:Attribute>
                <UML:Operation xmi.id = '10-100-38--23-4866c288:11d1ae9d
                    name = 'buscarContacto' visibility = 'public' isSpecif

```

III

» Language file	nb char : 29307	Ln : 1 Col : 1
-----------------	-----------------	----------------

Ilustración 88: Archivo XMI ejemplo Agenda de contactos

A continuación importaremos el archivo XMI en la herramienta GIA Java, este archivo será la entrada a nuestro sistema.

Una vez importado en la herramienta se verá como muestra la ilustración 89 Como se puede comprobar la clase Agenda tiene un atributo \$v_Contactos, este vector es la forma en la que el sistema representa la relación 1 a n existente entre la clase Agenda y Contacto. Lo mismo ocurre con la relación 1 a 1 que existe entre la clase Contacto y Dirección, la clase contacto guarda una referencia a la clase dirección \$r_Direccion.

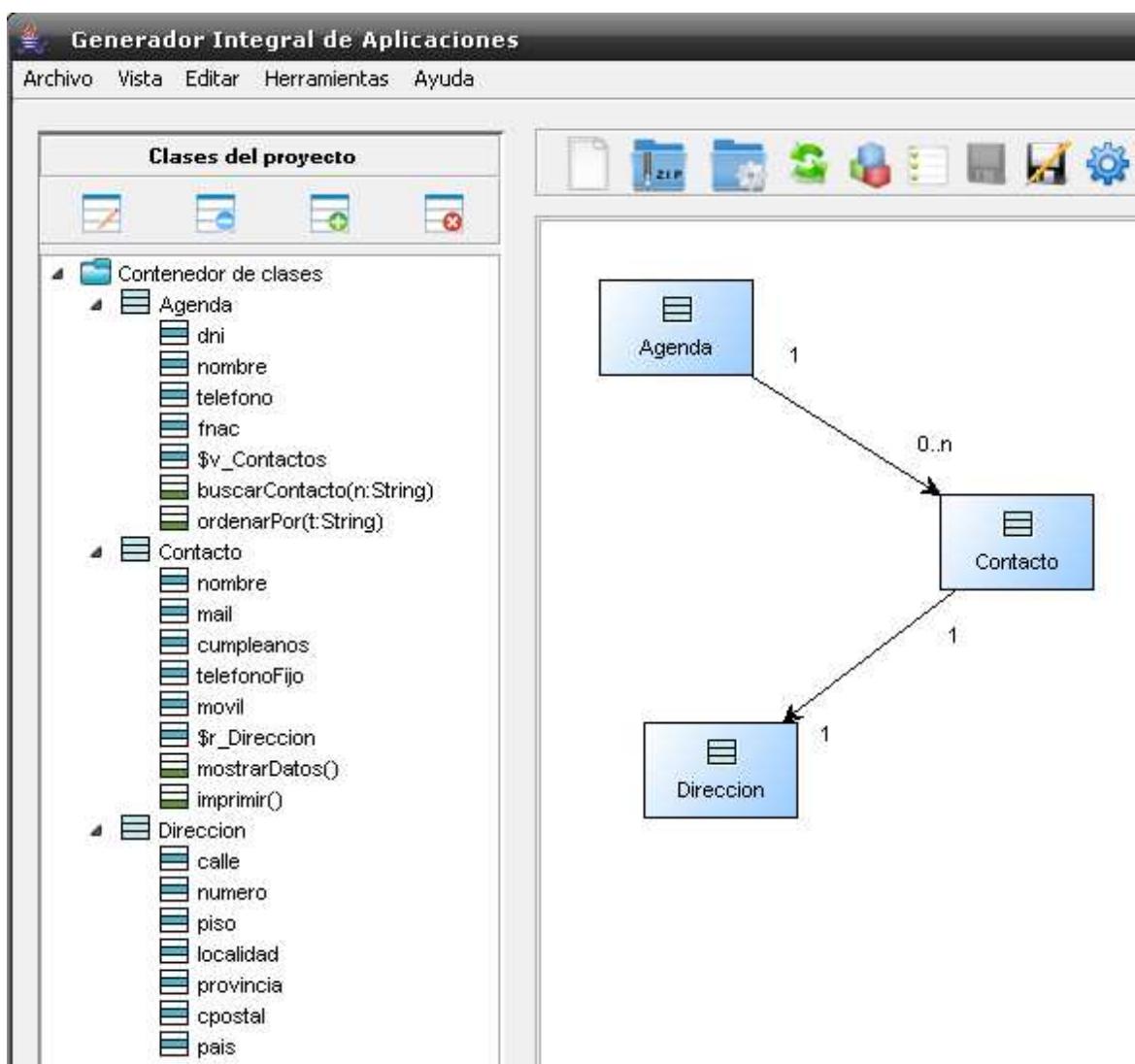
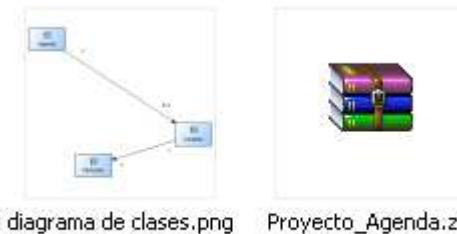


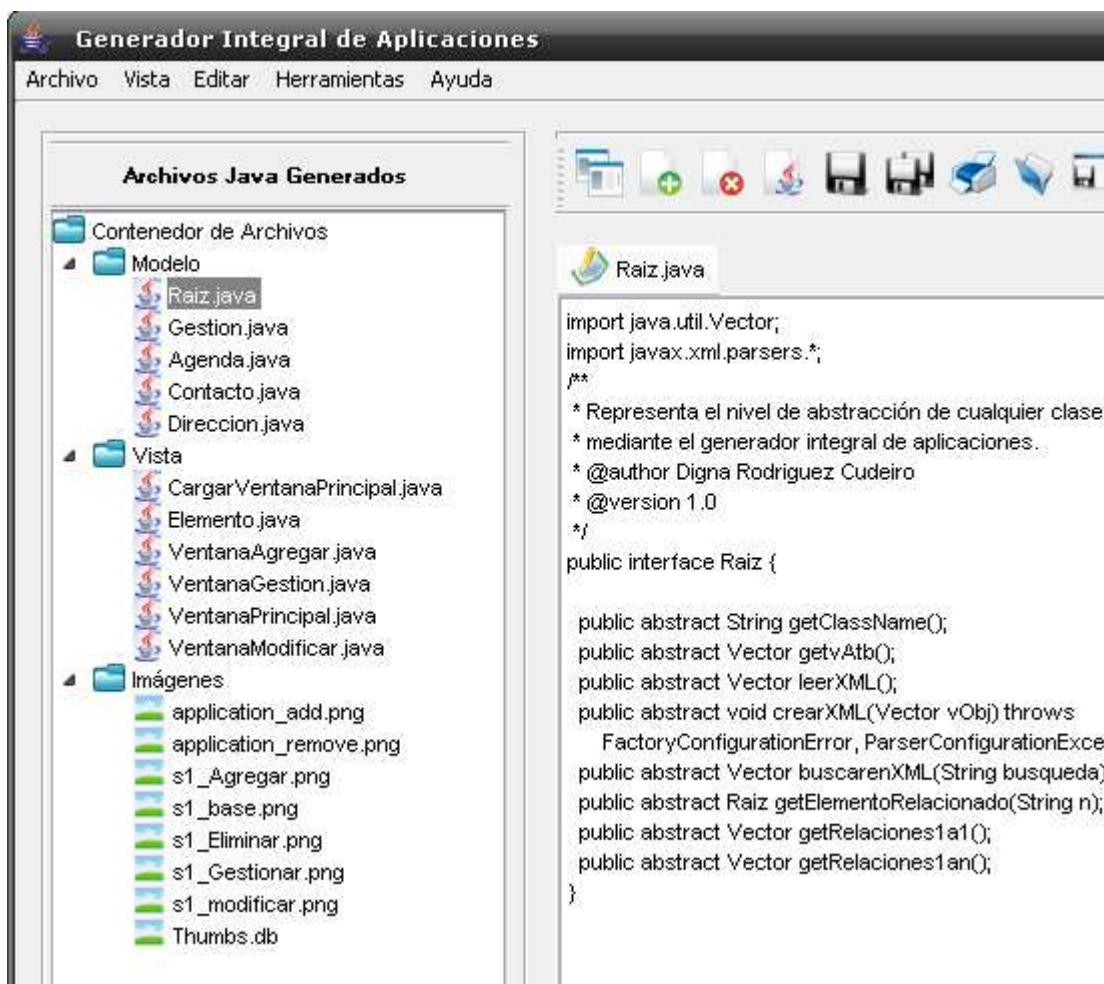
Ilustración 89: Herramienta GiA Java - Ejemplo Agenda

Desde la parte de diseño podremos editar el diagrama. En este caso pasaremos a seleccionar la clase principal directamente que para este ejemplo será la clase Agenda. Una vez seleccionada procederemos a generar el código java. Y pasaremos a la parte de IDE de java, desde la que se procederá a compilar y ejecutar el proyecto creado.

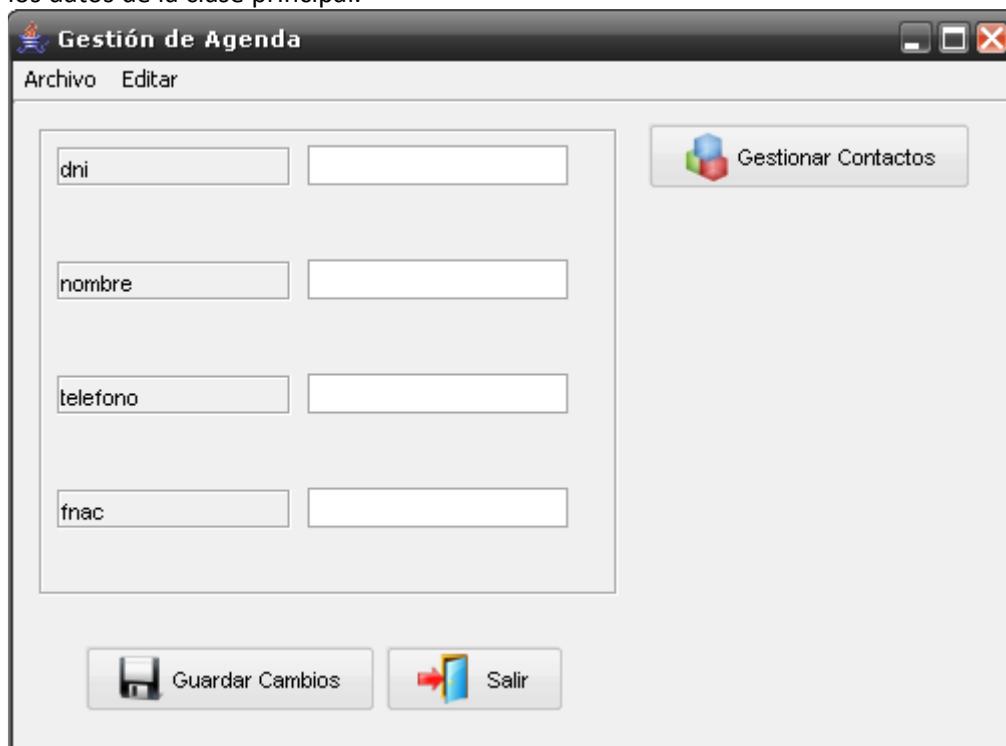
Pero antes guardaremos una copia del proyecto actual, pulsando el botón guardar como. Al guardarlo se creará un archivo ZIP que contendrá los XML de todas las clases y se exportará una imagen con el diagrama modelado en ese momento.

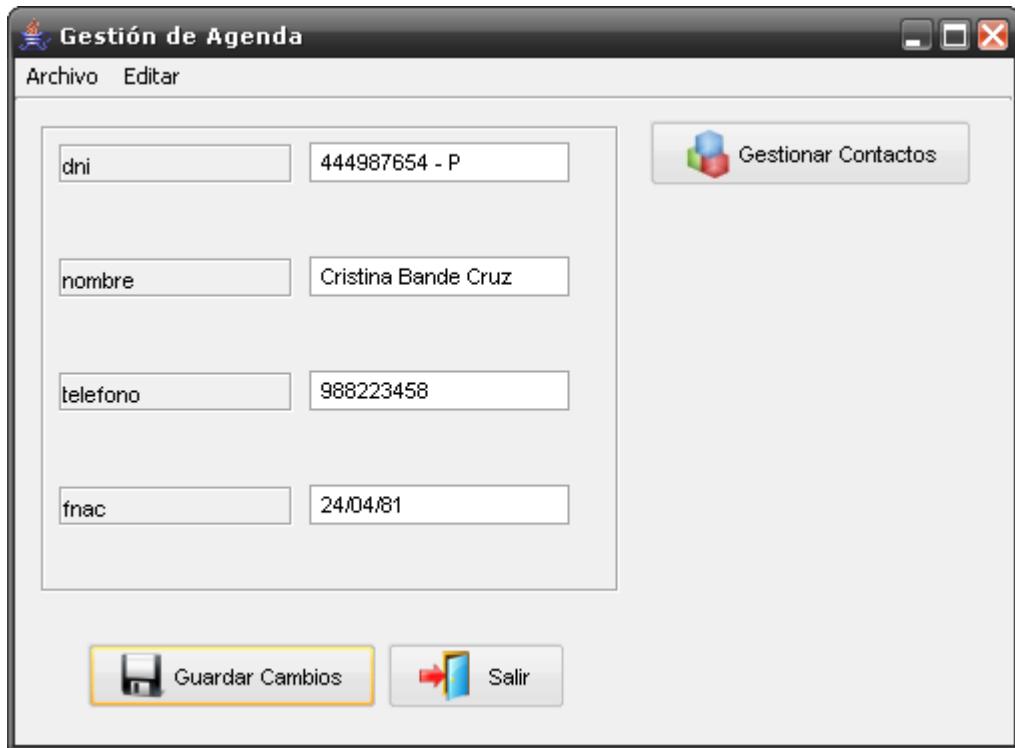


Después generamos el código y se abre la ventana de IDE de Java:



Cuando el proyecto creado es ejecutado se muestra la ventana principal desde la que se editan los datos de la clase principal.





Una vez rellenados todos los campos se pulsará el botón guardar cambios y en ese momento el sistema creará un XML con los atributos de la clase agenda.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<raiz>
<Agenda enlace="">
<ident>Ag_1252625933984</ident>
<dni>444987654 - P</dni>
<nombre>Cristina Bande Cruz</nombre>
<telefono>988223458</telefono>
<fnac>24/04/81</fnac>
</Agenda>
</raiz>
```

Si pulsamos ahora el botón gestionar Contactos se abrirá una nueva ventana desde la que se procederá a añadir, modificar y eliminar contactos.



Añadiremos un nuevo contacto y crearemos su dirección:



El resultado se visualizará en la ventana de gestión de contactos.



Y se habrán creado los archivos Contacto.xml y Direccion.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<raiz>
<Contacto enlace="Ag_1252625933984">
<ident>Co_1252763782078</ident>
<nombre>Juan Perez Ruiz</nombre>
<mail>juanpe@gmail.com</mail>
<cumpleanos>21/01/80</cumpleanos>
<telefonoFijo>988234567</telefonoFijo>
<movil>678435677</movil>
</Contacto>
</raiz>
```

El contacto tiene un enlace al identificador de la clase principal agenda 'Ag_1252625933984'. Lo mismo ocurre con la Dirección tiene un enlace al contacto 'C01'.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Direccion enlace="Co_1252763782078">
<ident>Di_1252763852937</ident>
<calle>Ervedelo</calle>
<numero>22</numero>
<piso>4º Dcha</piso>
<localidad>Ourense</localidad>
<provincia>Ourense</provincia>
<cpPostal>32002</cpPostal>
<pais>España</pais>
</Direccion>
</raiz>
```

Después de añadir varios contactos tendremos una agenda más completa. Desde la que podremos consultar los datos de cualquier contacto.



6.3.2 Ejemplo Videoclub

Un nuevo ejemplo es la gestión de un Videoclub, se pretenden gestionar las películas del videoclub, los clientes y los alquileres de las películas. Creamos el diagrama de clases con Argo.

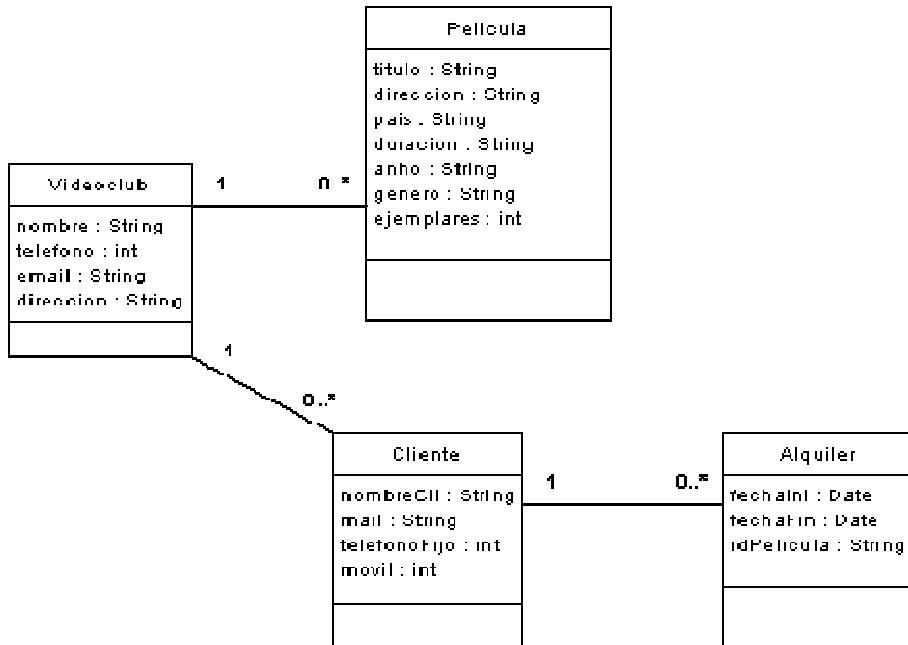


Ilustración 90: Diagrama de clases Ejemplo Videoclub - Modelado con ArgoUML

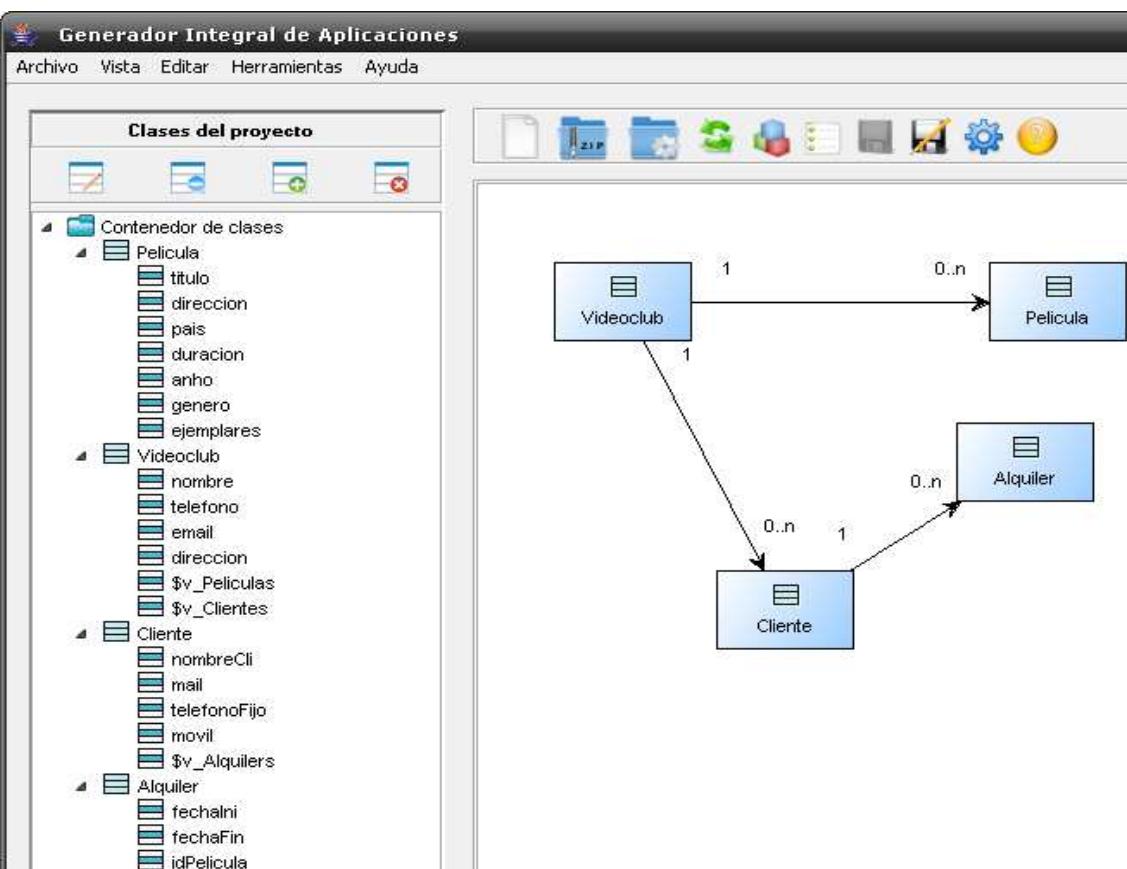
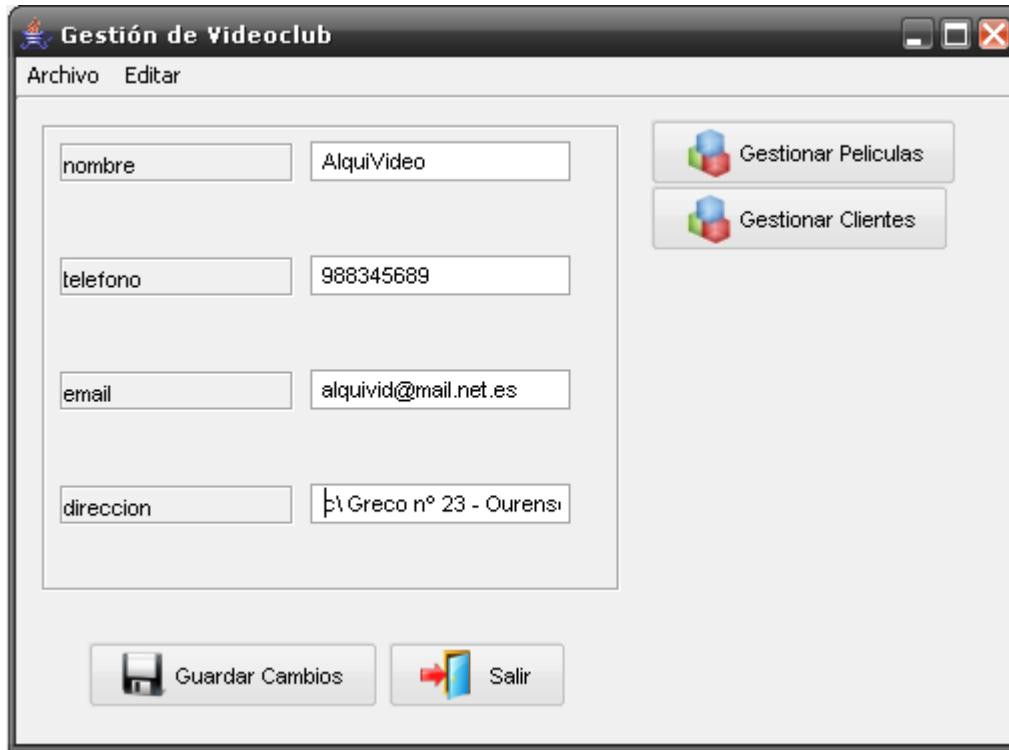


Ilustración 91: Herramienta GiA Java - Ejemplo Videoclub

Importamos el archivo XMI con la herramienta GiA Java. Y seleccionamos como clase principal la clase Videoclub. Generamos el código, lo compilamos y ejecutamos la aplicación generada:

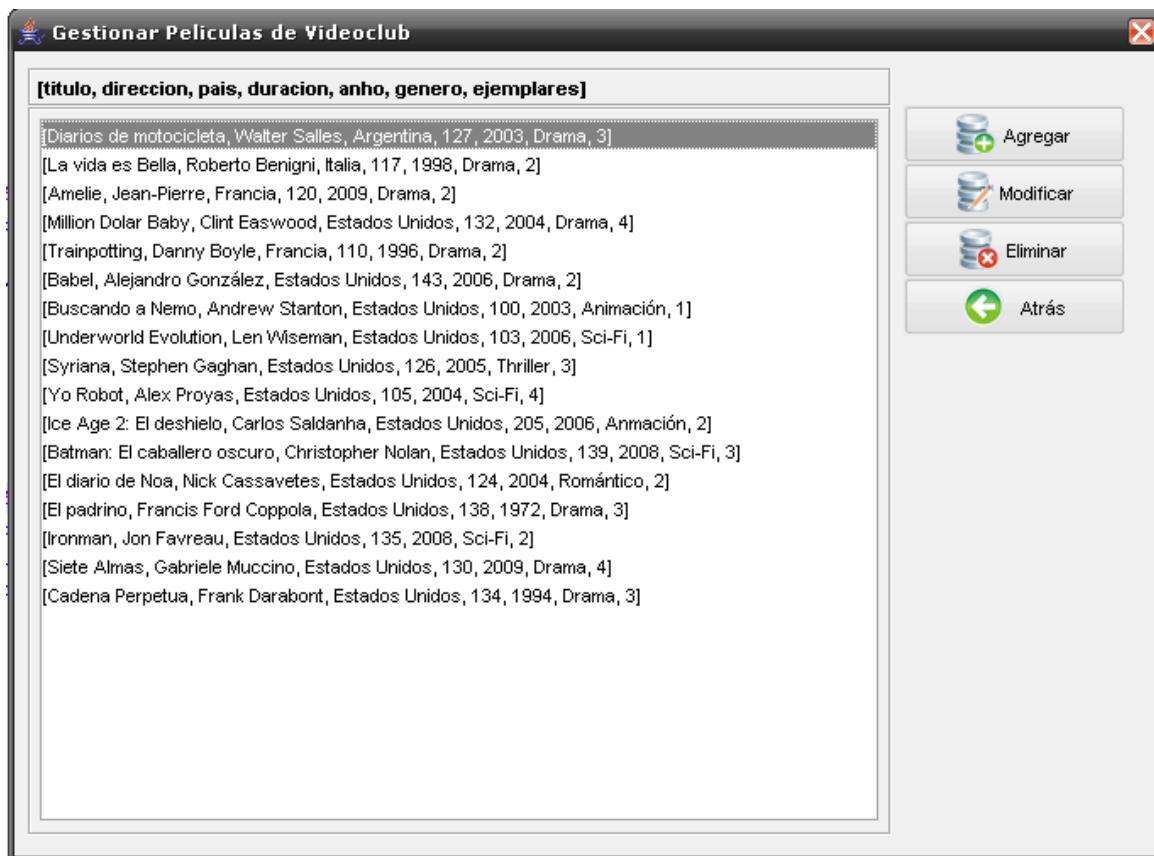


Rellenamos las características de un videoclub y guardamos los cambios, y automáticamente se crea un archivo XML para la clase Videoclub, como se puede ver los atributos visibles en la ventana principal se han creado en el xml, y además se crea internamente un atributo denominado ident que es un identificador único del videoclub:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<raiz>
<Videoclub enlace="">
<ident>Vi_1252766682750</ident>
<nombre>AlquiVideo</nombre>
<telefono>988345689</telefono>
<email>alquivid@mail.net.es</email>
<direccion>cc\ Greco nº 23 - Ourense</direccion>
</Videoclub>
</raiz>
```

Como se puede ver el atributo enlace de videoclub está vacío esto es porque esta clase no tiene ninguna dependencia con otra anterior, es decir, a esta clase no llega ninguna asociación sino que salen de ella.

A continuación procedemos a gestionar las películas. Añadimos varias películas, las editamos y el resultado es el siguiente:



El archivo XML en el que se almacenan los datos, se ha creado en una carpeta llamada XML situada en el directorio de la aplicación generada. Como se puede observar en la imagen posterior todas las películas tienen un enlace al videoclub, este enlace 'Vi_1252766682750' es el identificador de la clase videoclub almacenado internamente en el XML de la clase principal Videoclub, mostrado en una imagen anterior.

En este XML cada clase tiene también su propio identificador que es el código mediante el cual se buscan el XML para su edición o borrado.

El identificador se forma siempre con las dos primeras letras de la clase y a continuación un guión bajo seguido por una serie de números que representan la fecha y hora del sistema.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<raiz>
  <Pelicula enlace="Vi_1252766682750">
    <ident>Pe_1252766957859</ident>
    <titulo>Diarios de motocicleta</titulo>
    <direccion>Walter Salles</direccion>
    <pais>Argentina</pais>
    <duracion>127</duracion>
    <anho>2003</anho>
    <genero>Drama</genero>
    <ejemplares>3</ejemplares>
  </Pelicula>
  <Pelicula enlace="Vi_1252766682750">
    <ident>Pe_1252766957854</ident>
    <titulo>La vida es Bella</titulo>
    <direccion>Roberto Benigni</direccion>
    <pais>Italia</pais>
    <duracion>117</duracion>
    <anho>1998</anho>
    <genero>Drama</genero>
    <ejemplares>2</ejemplares>
  </Pelicula>
  <Pelicula enlace="Vi_1252766682750">
    <ident>Pe_1252766942847</ident>
    <titulo>Amelie</titulo>
    <direccion>Jean-Pierre</direccion>
    <pais>Francia</pais>
    <duracion>120</duracion>
    <anho>2009</anho>
    <genero>Drama</genero>
    <ejemplares>2</ejemplares>
  </Pelicula>
```

Podemos continuar añadiendo los clientes y sus alquileres de la misma manera.

6.3.3 Ejemplo Colegio

A continuación tomaremos como ejemplo un colegio y crearemos el diagrama de clases

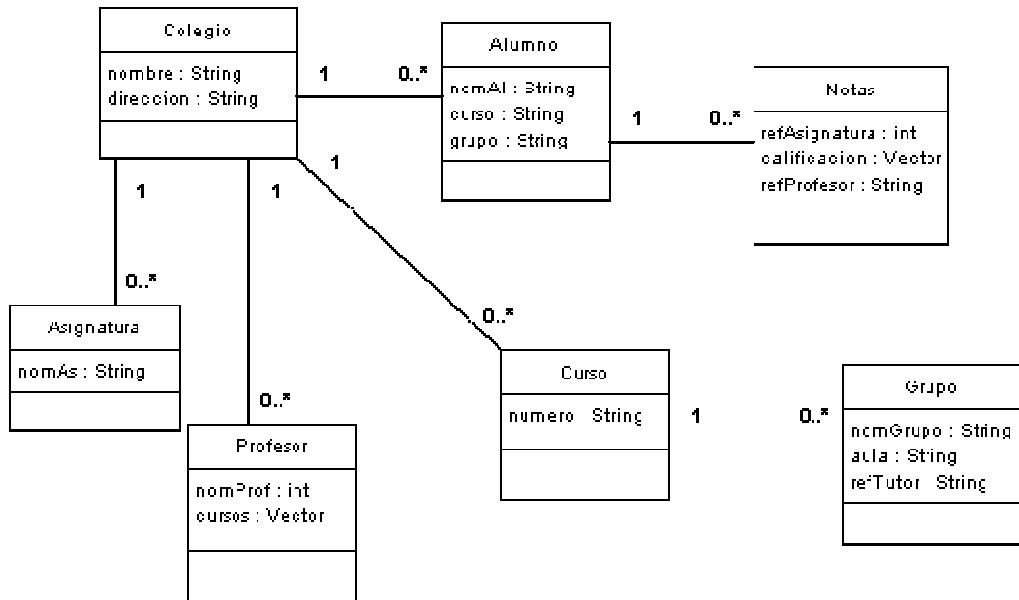


Ilustración 92: Diagrama de clase Ejemplo Colegio - Modelado con ArgoUML

Después de crear el diagrama con ArgoUML, lo importaremos con la herramienta GiA Java.

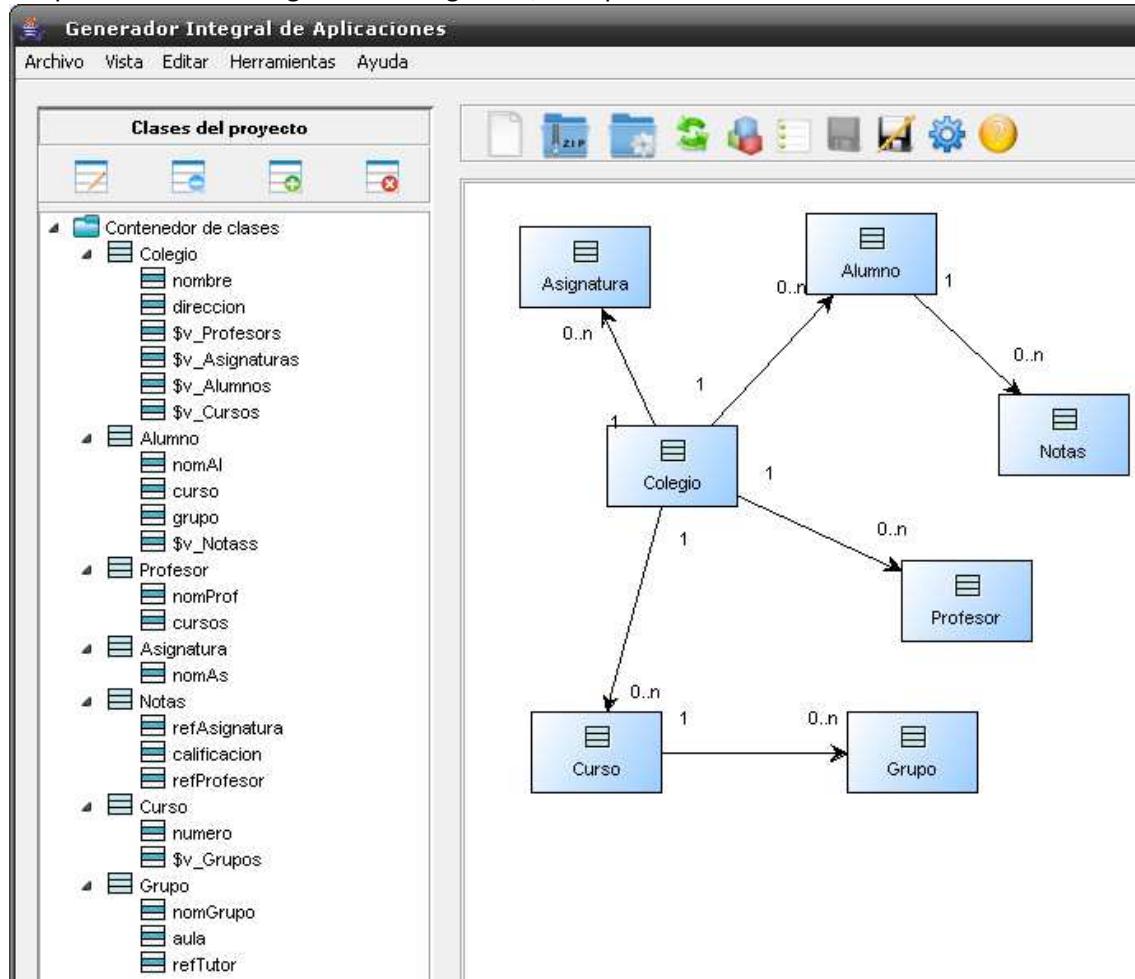
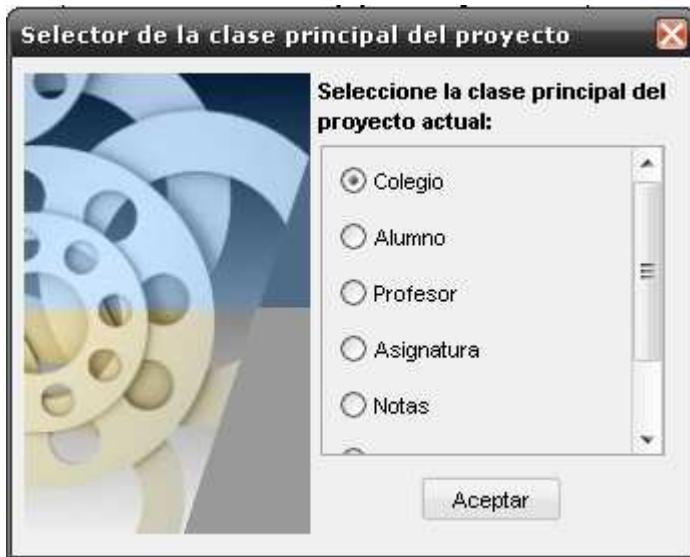


Ilustración 93: Herramienta GiA Java - Ejemplo Agenda

Como se puede comprobar el diagrama de clases es similar y las clases tienen atributos relacionados con las relaciones de asociación que existen entre las clases. Por ejemplo la clase Alumno tiene un vector en el que están contenidas todas sus Notas.

Procedemos a marcar la clase Colegio como clase principal puesto que es una clase a la que no llega ninguna flecha, y es el origen de todo el diagrama.



A posteriori generamos el código, lo compilamos y lo ejecutamos, el resultado es una ventana nueva principal desde la que se editan los datos del colegio, rellenamos los campos y guardamos los cambios:



A continuación Gestionamos los Profesores, los Cursos y las Asignaturas.



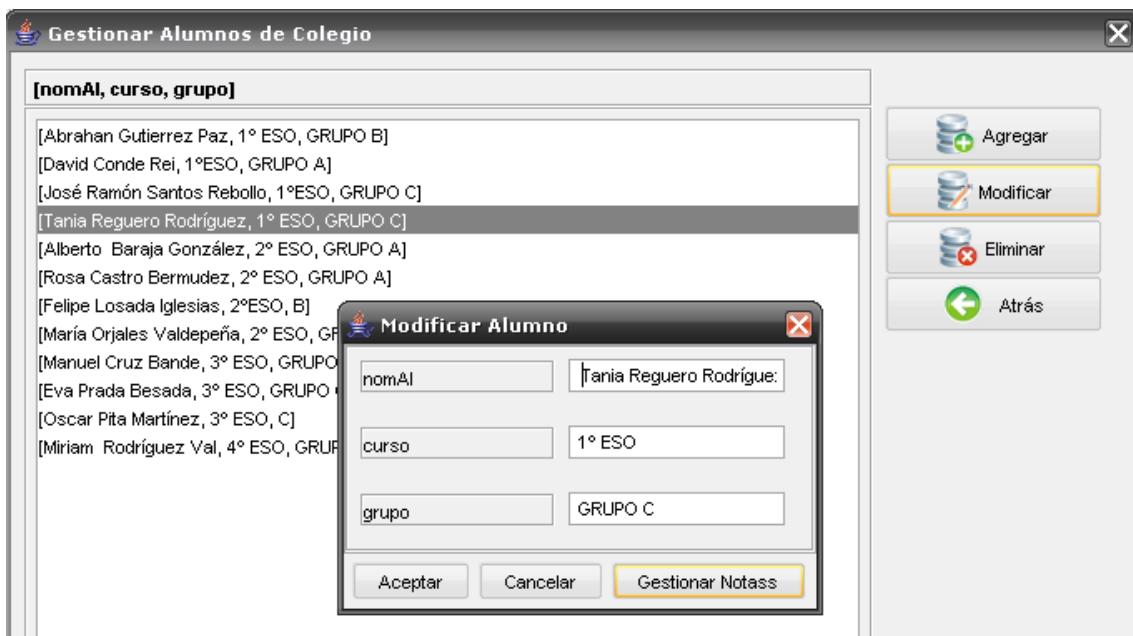
Para cada Curso creamos los grupos que tenga, con su referencia a un profesor que actúa como tutor que será introducida a mano:



Y por último gestionamos los alumnos del colegio



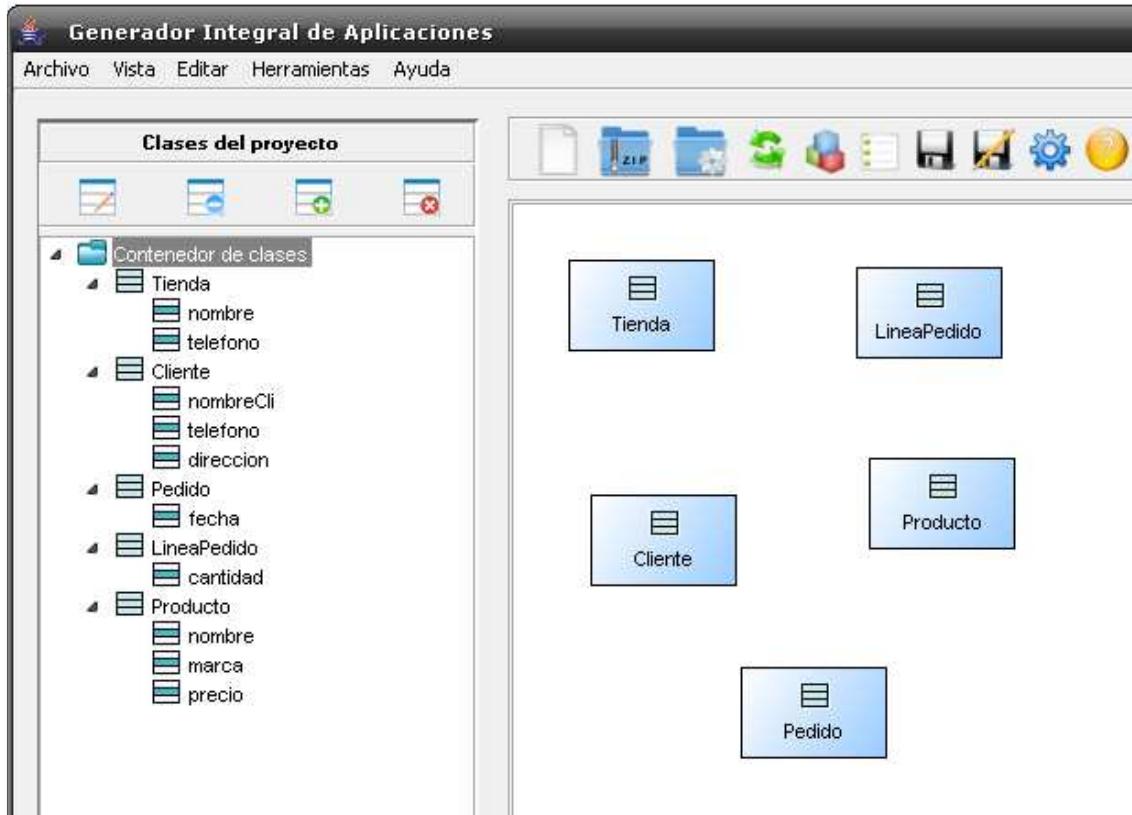
Y para cada alumno sus notas.



6.3.4 Ejemplo Tienda online

En este ejemplo crearemos el proyecto directamente en la herramienta, creándolo desde cero mediante la opción ‘Nuevo Proyecto’.

En primer lugar debemos de crear las clases y añadir sus atributos.



Para crear las relaciones entre clases debemos de añadir unos atributos determinados que el sistema entienda que son relaciones:

Para las relaciones 1 a 1 los nombres de los atributos deben de comenzar por \$r_ y ser del tipo de la clase que queremos referenciar. Por ejemplo si queremos relacionar una línea de un pedido con un producto, añadiremos el atributo de nombre \$r_Producto y tipo Producto a la clase LineaPedido.

Para las relaciones 1 a n los nombres de los atributos deben de comenzar por \$v_ ser del tipo Vector y además completar el tipo de los elementos del vector con el nombre de alguna de las clases del sistema. Por ejemplo si queremos reflejar que un pedido tiene n líneas de pedido añadiremos a la clase Pedido un atributo de tipo Vector de LineasPedido de nombre \$v_lineasPedido.

A continuación mostramos como quedaría el diagrama después de haber creado todas las relaciones entre las clases.

A posteriori seleccionamos como clase principal del sistema la clase Tienda, y pasamos a la vista de IDE de Java.

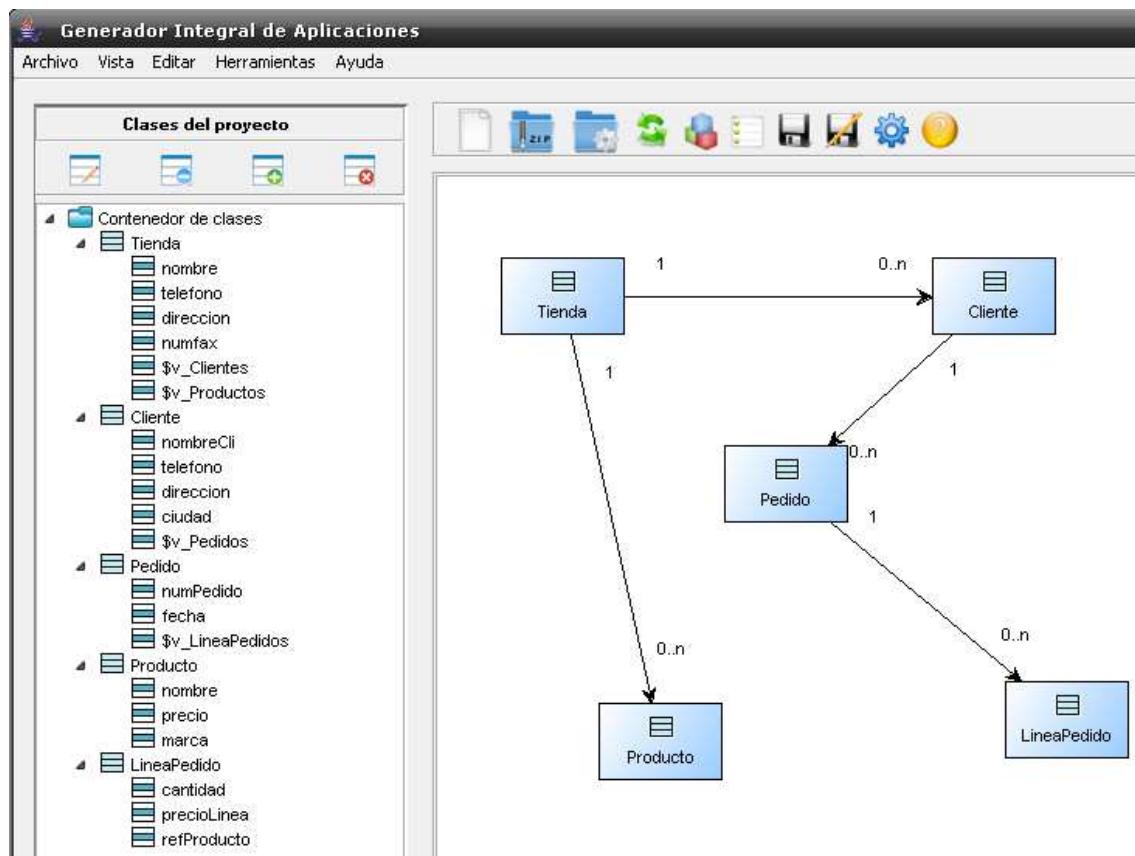
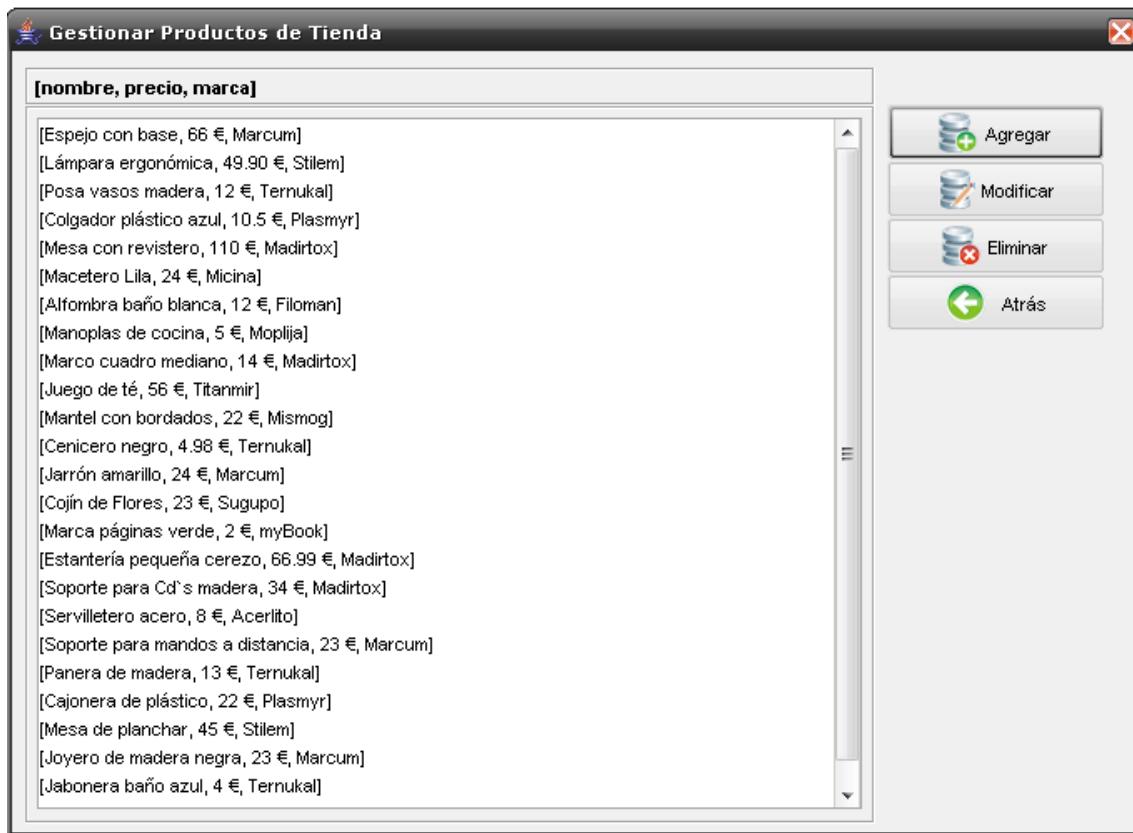


Ilustración 94: Herramienta GiA Java - Ejemplo Tienda Online

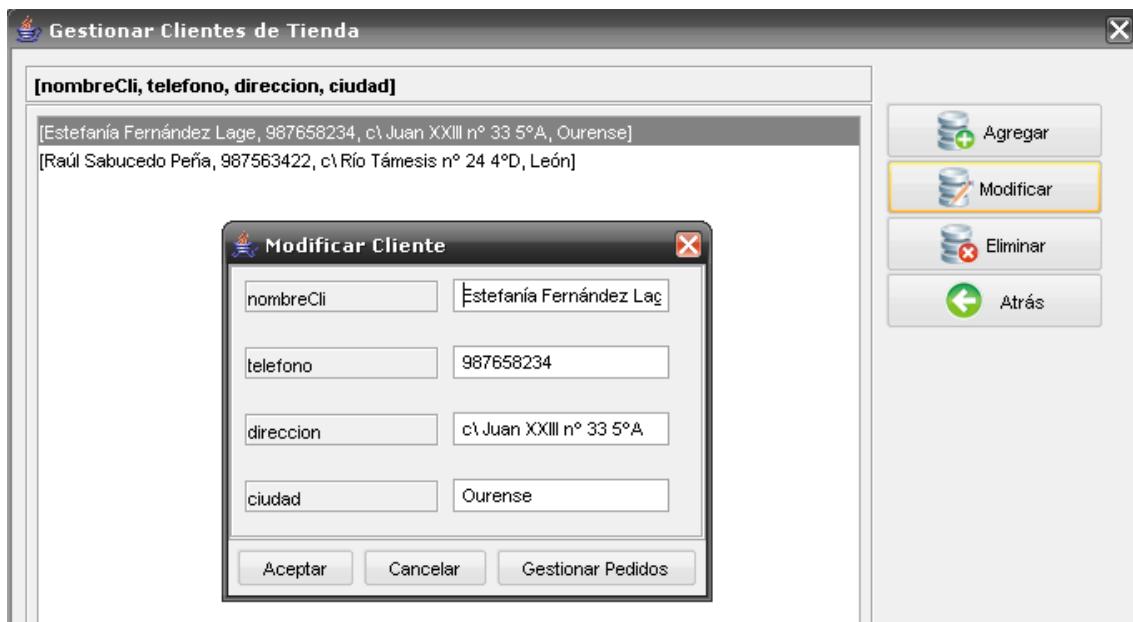
Compilamos y ejecutamos la aplicación:



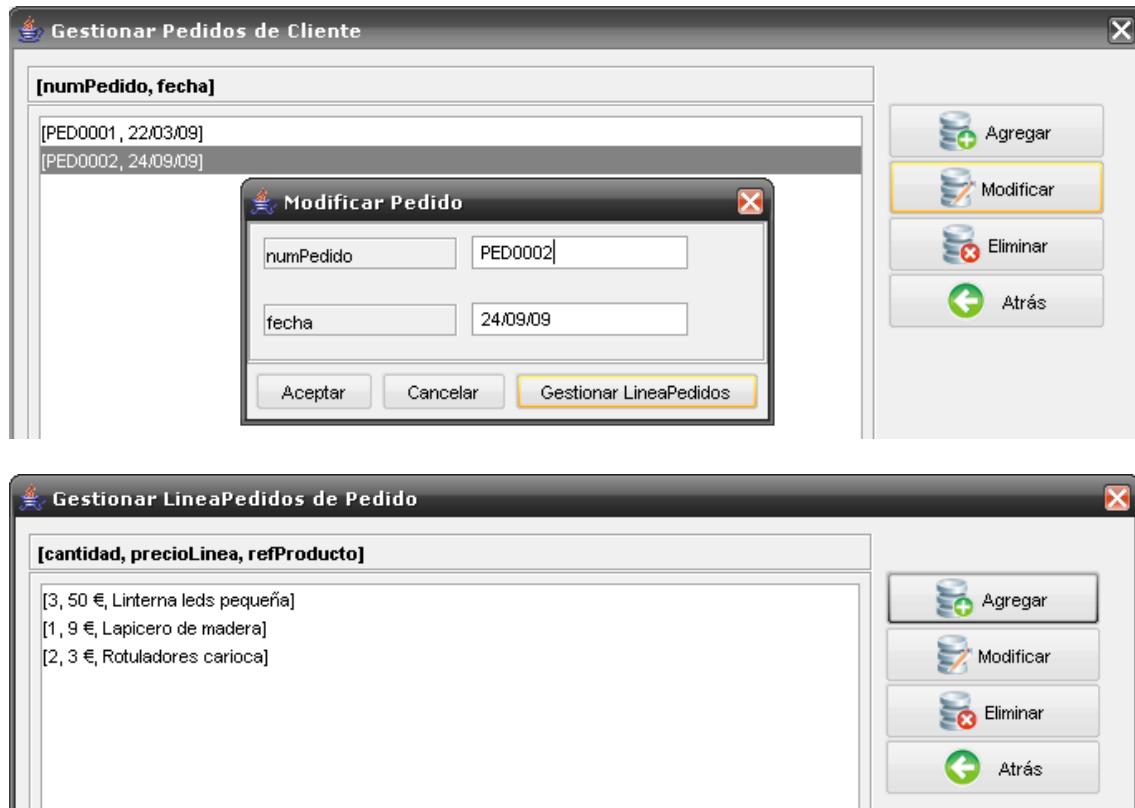
Editamos los datos de la tienda y gestionamos los productos que se venden en la tienda.



Creamos dos clientes y modificamos el primero, para añadir un pedido a su nombre.



Pulsamos el botón gestionar Pedidos y se abre una nueva ventana similar a la anterior en la que añadimos un nuevo pedido, y pasamos a añadir sus líneas de pedido.



6.3.5 Ejemplo Alquiler de Vehículos

En este último ejemplo crearemos el diagrama de clases desde la herramienta ArgoUML. Se trata de una empresa de alquiler de vehículos. El diagrama de clases es el siguiente:

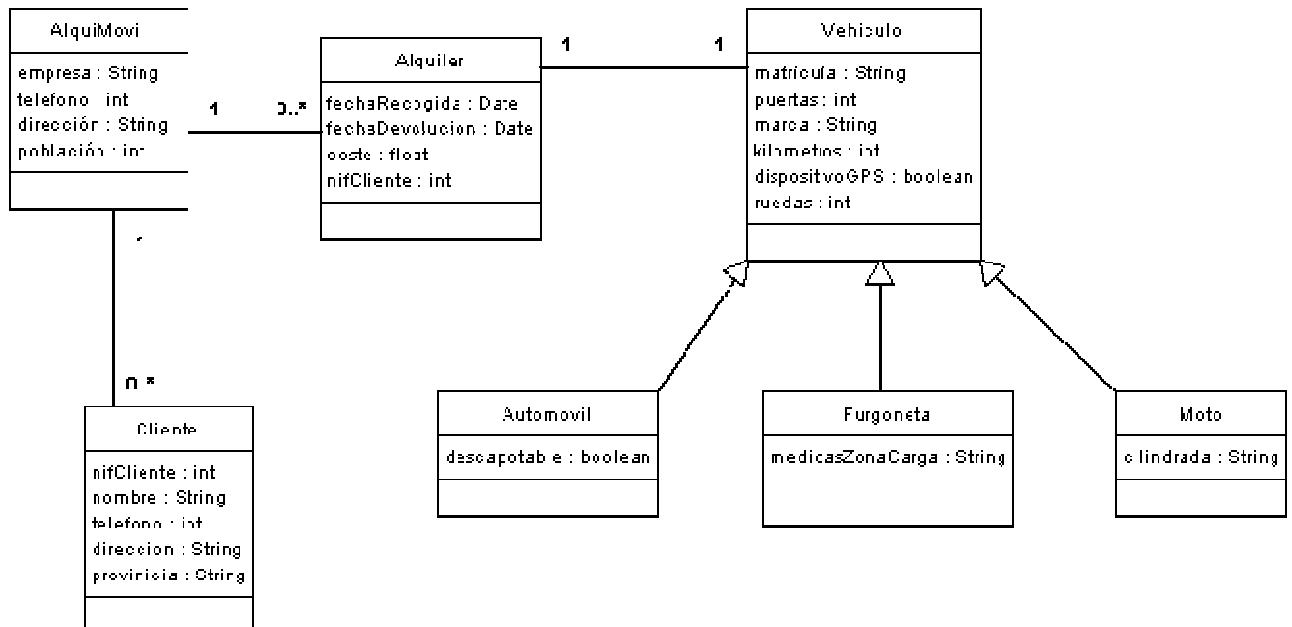
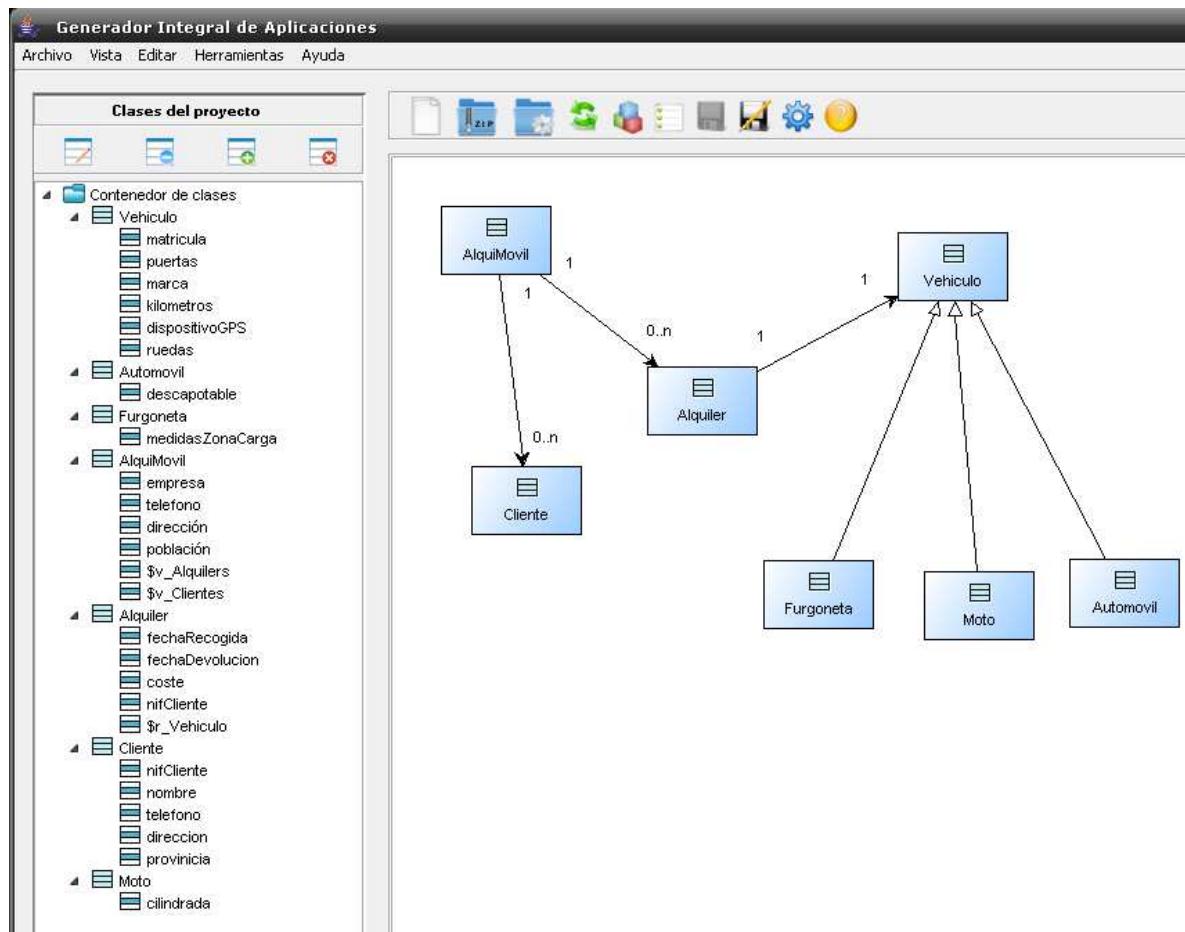
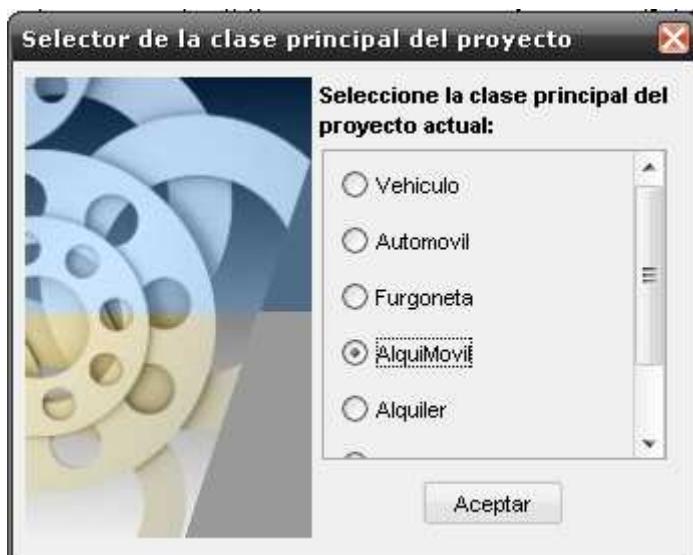


Ilustración 95: Diagrama de clase Ejemplo Alquiler de vehículos - Modelado con ArgoUML

Exportamos desde ArgoUML el modelo al formato XMI y lo importamos con la herramienta GiA java. El resultado de la importación es el siguiente:



Como podemos ver en este ejemplo existen relaciones de herencia. Automóvil, Furgoneta y Moto descienden de la clase Vehículo, esto se verá reflejado en el código ya que todas harán su extends de la clase vehículo, pero no se ve reflejado en la interfaz ya que al generar el código marcaremos como clase principal la clase AlquiMovil.

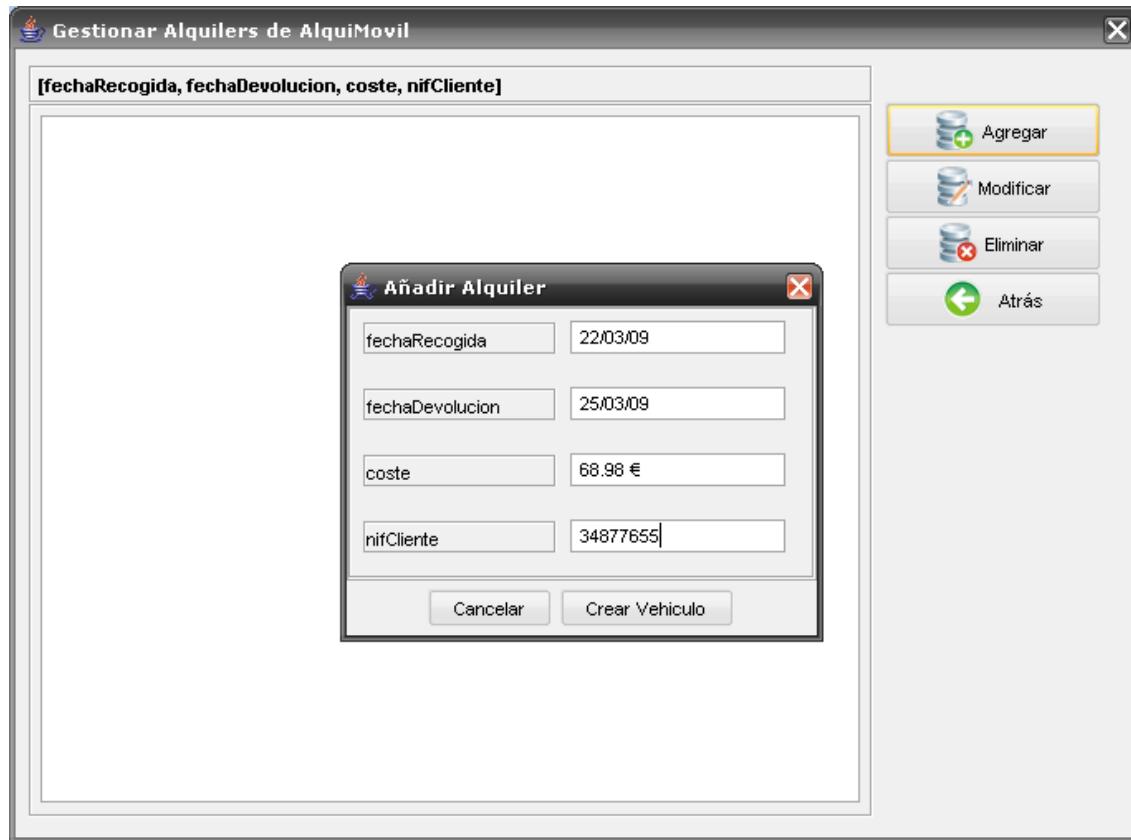


Generamos el código, lo compilamos y ejecutamos, el resultado es el siguiente:



Desde el menú editar podremos acceder a las secciones 'Gestionar Clientes' y 'Gestionar Alquileres':





Cada ficha de alquiler está relacionada con un automóvil del que introduciremos los datos al pulsar el botón crear vehículo.



Al pulsar se añadirá a la ventana de gestión de alquileres el alquiler añadido para ese cliente de ese vehículo.



Si queremos volver a la pantalla principal para gestionar los clientes habituales de la empresa deberemos de pulsar el botón Atrás. Y se gestionarán de la misma manera que se han hecho los alquileres.

1. BIBLIOGRAFÍA

A continuación se detalla la bibliografía utilizada:

- **El lenguaje unificado de modelado: manual de referencia**
James Rumbaugh, Ivar Jacobson, Grady Booch
Addison-Wesley, 2000
- **UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado**
Craig Larman
Prentice Hall, 2002
- **Mastering XMI : Java programming with XMI, XML, and UML**
Timothy J. Grose, Gary C. Doney, Stephen A. Brodsky
Publicación New York : John Wiley & Sons, cop. 2002
- **XML : la guía total del programador**
Francisco Minera
Publicación Buenos Aires : Mp Ediciones, 2006
- **Programación en Java 5.0**
James Cohoon, Jack W. Davidson ; traducción Jesús Sánchez Allende...[et al.]
McGraw-Hill, 2006

2. DIRECCIONES DE INTERNET

Página oficial OMG. *Especificación XMI*. Consulta: 13 Enero 2009

<http://www.omg.org/technology/documents/formal/xmi.htm>

Página de ExpertCode: conjunto de herramientas para escribir generadores de código.
Tutorial: *Manipulación de modelos UML*. Consulta: 22 Febrero 2009.

<http://expertcoder.sourceforge.net/tutorial/es/uml.html>

Artículo de IBM. *Trabajando con XML: UML ,XMI y generación de código*. Consulta: 23 Febrero 2009.

<http://www.ibm.com/developerworks/xml/library/x-wxxm23/>

ZVON.org. *XMI Reference*. Consulta: 14 Febrero 2009

<http://www.zvon.org/xxl/XMI/Output/index.html>

Wiki de CodeFun. *Tutorial de XMI*. Consulta: 22 de Febrero 2009

<http://www.codefun.org/wiki/doku.php/tutorials:xmi>

Página oficial de ArgoUML. *Documentación* . Consulta: 27 de Junio de 2009

<http://argouml-stats.tigris.org/>

Grupo de investigación kybele. Departamento de lenguajes y sistemas informáticos.
Presentación de AndroMDA. Consulta: 23 Junio 2009.

<http://www.kybele.etsii.urjc.es/docencia/HC4GL/2007-2008/Material/Exposiciones/Andro.pps>

Sun Microsystems. *Technical Articles and Tips. XML y Java*. Consulta: 12 Enero 2009.

<http://java.sun.com/developer/TechTips/2000/tt0627.html>

Guía de Usuario de Enterprise Architect . Consulta: 15 Junio 2009

<http://www.sparxsystems.com.ar/download/Ayuda%20HTML/index.html?importexport.htm>

Página oficial de ArgoUML. *ArgoUML Features*. Consulta: 22 Abril 2009

<http://argouml.tigris.org/features.html#XMI>

Chuidiang, wiki de tutoriales. *Patrón modelo vista controlador*. Consulta: 22 enero 2009.

http://www.chuidiang.com/ood/patrones/modelo_vista_controlador.php

Monografías.com, Febe Ángel Ciudad Ricardo, *Utilización del Patrón Modelo – Vista – Controlador (MVC) en el diseño de software educativos*. Consulta: 22 enero 2009

<http://www.monografias.com/trabajos43/patron-modelo-vista/patron-modelo-vista.shtml>

Tecnología orientada a procesos de negocio. Ramiro Lago Bagüés, *Patrón "Modelo-Vista-Controlador"*. Consulta: 3 Febrero 2009.

<http://www.proactiva-calidad.com/java/patrones/mvc.html>

Wikipedia. *Comparación de herramientas generadoras de código*. Consulta: 24 Mayo 2009

http://en.wikipedia.org/wiki/Comparison_of_code_generation_tools

Página oficial de BoUML. Consulta: 13 Junio 2009.

<http://bouml.free.fr/>

CICA: Centro informático científico de Andalucía. *Tipos de eventos Java*. Consulta: 22 Junio 2009.

<http://www.cica.es/formacion/JavaTut/Cap4/tipoev.html>

Página oficial CodeGen. *MDA tool*. Consulta: 14 de Enero de 2009

<http://www.solnatec.com/xead/codegen/whatis.html>

JavaWorld. Artículo: *Use JGraph to create a Wikipedia browser*. 23 Junio 2009.

<http://www.javaworld.com/javaworld/jw-07-2007/jw-07-jgraph.html?page=3>

Blog de programación. Jhon Alvarez Borja. *Personalización de iconos en un jtree*. Consulta: 12 de Marzo 2009.

<http://my.opera.com/jalvarezborja/blog/personalizacion-de-iconos-en-un-jtree>

Blog de programación : Le funes. Artículo: *Incluir una imagen de fondo en un JPanel*. Consulta: 10 Enero 2009.

<http://lefunes.wordpress.com/2008/11/22/incluyendo-una-imagen-de-fondo-en-un-jpanel/>

Página oficial de java Sun. *Tutoriales: Printing*. Consulta: 3 Abril 2009.

<http://java.sun.com/docs/books/tutorial/2d/printing/index.html>

Universidad de Chile. Departamento de ciencias de la computación. *Introducción a Java printing*. Consulta: 3 Abril 2009

<http://www.dcc.uchile.cl/~lmateu/CC60H/Trabajos/furibe/>

Grupo de investigación kybele. Departamento de lenguajes y sistemas informáticos. Document Object Model. Consulta: 22 de Marzo 2009.

http://kybele.escet.urjc.es/documentos/DE/DE2006-T4F_XML_DOM.pdf

Cafeconleche.org: Blog de noticias y recursos XML. *The Document Object Model*. Consulta: 12 de Enero de 2009.

<http://www.cafeconleche.org/books/xmljava/chapters/ch09.html>

Web Programación en castellano. *Tutorial de swing*. Consulta: 22 Julio 2009

<http://www.programacion.net/tutorial/swing/>

Chuidiang, wiki de tutoriales. *Jtree*. Consulta: 4 Enero 2009.

<http://www.chuidiang.com/chuwiki/index.php?title=JTree>

Web de trials y tutoriales: Magusoft. *Tutorial de Jtree*. 5 Enero 2009.

<http://www.magusoft.net/trials/tree.html>