

UNIVERSIDAD DE ORIENTE
NÚCLEO DE ANZOÁTEGUI
ESCUELA DE INGENIERIA Y CIENCIAS APLICADAS
DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS



DESARROLLO DE UN SOFTWARE QUE PERMITA LA GENERACIÓN AUTOMÁTICA DE CÓDIGO FUENTE DE INTERFACES GRÁFICAS DE USUARIO (GUI), COMO APOYO A LA ASIGNATURA “TALLER DE OBJETOS Y ABSTRACCIÓN DE DATOS”, IMPARTIDA EN EL DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS.

Realizado por:
Luis Fernando Farias Arteaga.

Trabajo de grado presentado como requisito parcial para optar al título de
INGENIERO EN COMPUTACIÓN

Barcelona, Octubre de 2009

UNIVERSIDAD DE ORIENTE
NÚCLEO DE ANZOÁTEGUI
ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS
DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS



DESARROLLO DE UN SOFTWARE QUE PERMITA LA GENERACIÓN
AUTOMÁTICA DE CÓDIGO FUENTE DE INTERFACES GRÁFICAS DE
USUARIO (GUI), COMO APOYO A LA ASIGNATURA “TALLER DE OBJETOS
Y ABSTRACCIÓN DE DATOS”, IMPARTIDA EN EL DEPARTAMENTO DE
COMPUTACIÓN Y SISTEMAS.

Asesora:

Ing. Zulirais García
Asesora Académica

Barcelona, Octubre de 2009.

UNIVERSIDAD DE ORIENTE
NÚCLEO DE ANZOÁTEGUI
ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS
DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS



DESARROLLO DE UN SOFTWARE QUE PERMITA LA GENERACIÓN AUTOMÁTICA DE CÓDIGO FUENTE DE INTERFACES GRÁFICAS DE USUARIO (GUI), COMO APOYO A LA ASIGNATURA “TALLER DE OBJETOS Y ABSTRACCIÓN DE DATOS”, IMPARTIDA EN EL DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS.



Jurado Calificador

Ing. Zulirais García
Asesora Académica

Ing. José Luis Bastardo
Jurado Principal

Ing. Claudio Cortínez
Jurado Principal

Barcelona, Octubre de 2009.

RESOLUCIÓN

De acuerdo con el artículo 44 del reglamento de trabajo de grado:

“Los trabajos de grado son de exclusiva propiedad de la Universidad de Oriente y sólo podrán ser utilizados a otros fines con el consentimiento del consejo de núcleo respectivo, quien lo participará al Consejo Universitario”.

DEDICATORIA

Dedico este proyecto y toda mi carrera universitaria a Dios por ser quien ha estado a mi lado en todo momento. A mi mamá Miryan Arteaga de Farias y a mi papá Luis “Guicho” Farias por darme todo ese cariño y calor humano necesario, por velar por mi salud, mis estudios, mi educación y mi felicidad. Es a ellos a quienes debo todo y gracias a ellos soy quien soy hoy en día.

También quiero recordar en esta dedicatoria a mi abuelo Nelson y a mi abuela Luisa quienes lamentablemente no están conmigo físicamente pero si en mis más bonitos recuerdos. Abuelos los amo, son ustedes mi mayor inspiración y mi mejor protección día a día.

AGRADECIMIENTOS

Principalmente a Dios por abrirme todas las puertas y por darme y permitirme conservar el privilegio de la vida.

A mi mamá Miryan, por estar a mi lado toda mi vida y velar por mí. Es a ti a quien más honro con este éxito porque eres tu quien desde que nací ha velado mas por él.

A mi papá “Guicho”, por estar a mi lado y por darme, a tu forma única, la mejor de las guías y modelo a seguir. Gracias por protegerme y por todo el esfuerzo q haces por hacer feliz a mí y a mi madre.

A mi abuela Myrian por todo el apoyo brindado a lo largo de toda mi carrera y por ser mi gran inspiración y modelo académico a seguir. Gracias por tanta sabiduría, tanta paz y tanto amor.

A mi madrina Luisa por ser simplemente mi santa, por ser mi guía espiritual y por quererme tanto.

A mi tía Madely por estar tan pendiente de mi y brindarme tanto apoyo cuando más lo he necesitado.

A mi tío Daniel Arteaga y mi tía Marianella Arteaga por ser siempre los mejores. Mis mejores amigos, cómplices y los más comprensivos en toda mi vida.

A mis primos Nacho, Francisco y Gerardo por ser más que mis hermanos toda mi vida y por darme tantos momentos gratos.

A mi tutora académica, Prof. Zulirais García, por extenderme sus conocimientos y hacer de mi una mejor persona y un mejor profesional.

A toda mi familia en general, a quienes no menciono porque la lista sería interminable. A todos los quiero mucho, gracias por estar ahí para mí.

A todos ustedes: Muchas Gracias!!

Luis Fernando Farias Arteaga

ÍNDICE

RESOLUCIÓN	IV
DEDICATORIA	V
AGRADECIMIENTOS	VI
ÍNDICE	VIII
INDICE DE ILUSTRACIONES	XIV
INDICE DE TABLAS	XIX
RESUMEN	XXI
CAPÍTULO I	22
INTRODUCCIÓN	22
1.1 PLANTEAMIENTO DEL PROBLEMA.....	22
1.2 OBJETIVOS.....	27
1.2.1 Objetivo General.....	27
1.2.2 Objetivos Específicos.....	27
CAPÍTULO II	29
MARCO TEÓRICO	29
2.1 ANTECEDENTES.....	29
2.1.1 “Diseño e implementación de un case orientado a la solución de problemas matemáticos utilizando las técnicas del organigrama o diagrama de flujo”. [1].....	29
2.1.2 “Desarrollo de un tutor para la enseñanza de los algoritmos usados en las estructuras de datos utilizando técnicas de animación”. [2].....	31
2.1.3 “Desarrollo de un software interactivo para la enseñanza de la asignatura Química I (10-1814) del área científica-tecnológica”. [3].....	32

2.2 CONOCIMIENTOS PREVIOS	33
2.2.1 Proceso unificado de desarrollo de software	33
2.2.1.1 Principios del RUP	33
2.2.1.2 Ciclo de vida	34
2.2.2 Lenguaje Unificado de Modelado (UML).....	36
2.2.2.1 Definición	37
2.2.2.2 Utilidad de UML	38
2.2.2.3 Bloques Básicos de UML	39
2.2.2.3.1 Elementos del Lenguaje.....	40
2.2.2.3.2 Relaciones entre Elementos	43
2.2.2.3.3 Diagramas de UML	45
2.2.3 Ingeniería de Software con Modelaje Orientada a Objetos	53
2.2.4 Programación Orientada a Objetos	54
2.2.4.1 Definición	54
2.2.4.2 Conceptos Fundamentales	54
2.2.4.3 Características de la POO	56
2.2.5 Interfaz de Usuario	58
2.2.5.1 Definición	58
2.2.5.2 Principales Funciones de las Interfaces de Usuario.....	58
2.2.5.3 Tipos de Interfaces de Usuario	58
2.2.6 Interfaces Gráficas de Usuario (GUI).....	59
2.2.7 Lenguaje de Programación Java	60
2.2.8 Aplicaciones	61
2.2.9 Sub aplicación Java para Web o Applet	61
CAPÍTULO III.....	63
FASE DE INICIO	63
3.1 CONTEXTO DEL SISTEMA	64
3.1.1 Modelo de Dominio.....	64

3.1.2	Glosario de términos para el modelo de dominio.....	66
3.2	REQUISITOS FUNCIONALES.....	67
3.3	IDENTIFICACION DE RIESGOS.....	68
3.4	MODELO DE CASOS DE USO.....	69
3.4.1	Identificación de Actores.....	71
3.4.2	Descripción de casos de uso del sistema	71
3.5	REQUISITOS ADICIONALES DEL SISTEMA	89
3.6	ANÁLISIS	90
3.6.1	Diagramas de Clases de analisis.....	90
3.6.1.1	Diagrama de clases de análisis para el caso de uso “Modelar Interfaz Grafica”.....	91
3.6.1.2	Diagrama de clases de análisis para el caso de uso “Generar Código”	92
3.6.1.3	Diagrama de clases de análisis para el caso de uso “Configurar Archivo”.....	93
3.6.2	Diagramas de Colaboración.....	93
3.6.2.1	Diagrama de colaboración para el caso de uso “Modelar Interfaz Grafica”	94
3.6.2.2	Diagrama de colaboración para el caso de uso “Generar código”	95
3.6.2.3	Diagrama de colaboración para el caso de uso “Configurar Archivo”.....	96
3.6.3	Diagrama de Paquetes de Análisis.....	98
3.7	CONCLUSIÓN DE LA FASE DE INICIO.....	100
CAPÍTULO IV	102
FASE DE ELABORACIÓN	102
4.1	REQUISITOS	102
4.1.1	Interfaz de inicio LFUI Creator	103

4.1.2 Interfaz principal (Pantalla de Inicio).....	104
4.1.3 Interfaz principal (Modo edición de UI).....	105
4.1.4 Interfaz principal (Modo generacion de codigo)	106
4.1.5 Dialogo de mensaje de Error	107
4.2 DISEÑO.....	107
4.2.1 Diagrama de Clase de Diseño.....	108
4.2.1.1 Diagrama de clases de Diseño para el caso de uso Generar Código	109
4.2.2. Diagrama de Secuencia	110
4.2.2.1 Diagrama de secuencia para el caso de uso Generar Código. .	110
4.2.2.2 Diagrama de secuencia para el caso de uso Modelar Interfaz Gráfica.....	111
4.2.3 Diagrama de Capas	112
4.3 IMPLEMENTACIÓN	113
4.3.1 Diagrama de Componentes:.....	114
4.3.2 Implementación de los componentes asociados al caso de uso Generar Código.	114
4.4 CONCLUSIÓN DE LA FASE DE ELABORACIÓN	117
CAPÍTULO V	119
FASE DE CONSTRUCCIÓN.....	119
5.1 INTRODUCCION	119
5.1.1 Escogencia del Lenguaje de Programación	119
5.2 IMPLEMENTACION	121
5.2.1 Análisis de formato para archivos de propiedades “PRV” y sintaxis “STX”	121
5.2.1.1 Formato PRV	121
5.2.1.2 Formato STX	122
5.2.2 Diagrama de Componentes Total	124

5.3 PRUEBAS	126
5.3.1 Pruebas por Unidad	126
5.3.2 Pruebas de Integración.....	129
5.3.2.1 Integración de la Configuración de Proyecto	131
5.3.2.1.1 Resultados.....	131
5.3.2.1.2 Procedimiento de prueba	132
5.3.2.2 Código de componentes en la fase de Configuración de Proyecto	134
5.3.2.2.1 Proyecto.java	134
5.3.2.2.2 GestorArchivoProyecto.java.....	135
5.3.2.2.3 NuevoProyectoDialog.java.....	138
5.3.2.2.4 NuevoProyectoDialogListener.java.....	141
5.4 CONCLUSIÓN DE LA FASE DE CONSTRUCCIÓN.....	142
CAPÍTULO VI	144
FASE DE TRANSICIÓN.....	144
6.1 INTRODUCCIÓN	144
6.2 MANUAL DE USUARIO	144
6.2.1 Introducción.....	147
6.2.2 Interfaz de Inicio LFUI Creator.....	147
6.2.3 Elementos de la Interfaz Principal.....	148
6.2.3.1 Opciones de Archivo de Proyecto	148
6.2.3.2 Opciones de Proyecto o gestión de UIs.	149
6.2.3.3 Opciones de Configuración de Sistema.....	150
6.2.3.4 Opciones de Ayuda del Sistema	150
6.2.3.5 Modo de Edición	151
6.2.3.6 Control de Vistas de Proyecto	152
6.2.3.7 Vista de Proyecto.....	152
6.2.3.8 Vista de Componentes	153

6.2.3.9 Paleta de Componentes.....	154
6.2.3.10 Control de Propiedades.....	155
6.2.4 Gestión de Proyectos	156
6.2.4.1 Crear proyecto	156
6.2.4.2 Abrir Proyecto	156
6.2.4.3 Guardar y Guardar como	157
6.2.4.4 Proyectos recientes	157
6.2.4.5 Cerrar proyecto	158
6.2.5 Modelado de una Interfaz Gráfica	158
6.2.5.1 Agregar una nueva UI.....	158
6.2.5.2 Eliminar una UI	159
6.2.5.7 Agregar una UI Existente.	159
6.2.5.8 Agregar componente a UI.....	159
6.2.5.8 Modificar componente en UI.....	160
6.2.6 Generación de Código	161
CONCLUSIONES	163
RECOMENDACIONES.....	165
BIBLIOGRAFÍA	166

ÍNDICE DE ILUSTRACIONES

FIGURA 2.1: ESFUERZO EN ACTIVIDADES SEGÚN FASE DEL PROYECTO.....	36
FIGURA 2.2: REPRESENTACIÓN DE UN ACTOR.	40
FIGURA 2.3: REPRESENTACIÓN DE UN CASO DE USO.	40
FIGURA 2.4: REPRESENTACIÓN DE UNA CLASE.	41
FIGURA 2.5. REPRESENTACIÓN DE UN OBJETO.	41
FIGURA 2.6: REPRESENTACIÓN DE UN MENSAJE.	42
FIGURA 2.7: REPRESENTACIÓN DE UN PAQUETE.....	42
FIGURA 2.8: REPRESENTACIÓN DE LA RELACIÓN DE DEPENDENCIA ENTRE DOS CLASES.	43
FIGURA 2.9: REPRESENTACIÓN DE LA RELACIÓN DE ASOCIACIÓN ENTRE DOS CLASES.	44
FIGURA 2.10: REPRESENTACIÓN DE UNA RELACIÓN DE AGREGACIÓN ENTRE DOS CLASES.	44
FIGURA 2.11: REPRESENTACIÓN DE UNA RELACIÓN DE GENERALIZACIÓN ENTRE DOS CLASES.....	45
FIGURA 2.12: DIAGRAMA DE CASOS DE USO.....	46
FIGURA 2.13: DIAGRAMA DE SECUENCIA.	47
FIGURA 2.14: DIAGRAMA DE COLABORACIÓN.	48
FIGURA 2.15: DIAGRAMA DE CLASES.....	49
FIGURA 2.16: DIAGRAMA DE DESPLIEGUE.....	49

FIGURA 2.17: DIAGRAMA DE COMPONENTES.	50
FIGURA 2.18: DIAGRAMA DE ESTRUCTURA COMPUESTA.	50
FIGURA 2.19: DIAGRAMA DE OBJETOS.....	51
FIGURA 2.20. DIAGRAMA DE ESTADOS.	52
FIGURA 2.21: DIAGRAMA DE ACTIVIDADES.....	53
FIGURA 3.1: MODELO DE DOMINIO.....	65
FIGURA 3.2: MODELO DE CASOS DE USO.....	70
FIGURA 3.3: DIAGRAMA DE CLASES DE ANÁLISIS PARA EL CASO DE USO “MODELAR INTERFAZ GRAFICA”	91
FIGURA 3.4: DIAGRAMA DE CLASES DE ANÁLISIS PARA EL CASO DE USO “GENERAR CÓDIGO”	92
FIGURA 3.5: DIAGRAMA DE CLASES DE ANÁLISIS PARA EL CASO DE USO “CONFIGURAR ARCHIVO”	93
FIGURA 3.6: DIAGRAMA DE COLABORACIÓN PARA EL CASO DE USO “MODELAR INTERFAZ GRAFICA”	94
FIGURA 3.7: DIAGRAMA DE COLABORACIÓN PARA EL CASO DE USO “GENERAR CÓDIGO”	95
FIGURA 3.8: DIAGRAMA DE COLABORACIÓN PARA EL CASO DE USO “CONFIGURAR ARCHIVO”	96
FIGURA 3.9: DIAGRAMA DE PAQUETES DE ANÁLISIS DEL SISTEMA.	98
FIGURA 3.10: PAQUETES DE ANÁLISIS DE MODELADO DE UI.	99
FIGURA 3.11: PAQUETES DE ANÁLISIS CONFIGURACIÓN DE ARCHIVO DE PROYECTO.....	99

FIGURA 3.11: PAQUETES DE ANÁLISIS CONFIGURACIÓN GENERACIÓN DE CÓDIGO.....	100
FIGURA 4.1: INTERFAZ DE INICIO DEL SISTEMA	103
FIGURA 4.2: INTERFAZ PRINCIPAL (PANTALLA DE INICIO).....	104
FIGURA 4.3: INTERFAZ PRINCIPAL (MODO EDICIÓN DE UI)	105
FIGURA 4.4: INTERFAZ PRINCIPAL (MODO GENERACION DE CODIGO).....	106
FIGURA 4.5: DIALOGO DE MENSAJE DE ERROR	107
FIGURA 4.6: DIAGRAMA DE CLASE DE DISEÑO LFUI CREATOR.....	108
FIGURA 4.7: DIAGRAMA DE CLASE DE DISEÑO PARA EL CASO DE USO GENERAR CÓDIGO	109
FIGURA 4.8: DIAGRAMA DE SECUENCIA PARA EL CASO DE USO GENERAR CÓDIGO	110
FIGURA 4.9: DIAGRAMA DE SECUENCIA PARA EL CASO DE USO MODELAR INTERFAZ GRÁFICA	111
FIGURA 4.10: DIAGRAMA DE CAPAS LFUI CREATOR.	113
FIGURA 4.11: DIAGRAMA DE COMPONENTES PARCIAL PARA EL CASO DE USO GENERAR CÓDIGO.	114
FIGURA 5.1: INTERFAZ DEL ENTORNO DE PROGRAMACIÓN JAVA (JCREATOR).....	120
FIGURA 5.2: DIAGRAMA DE COMPONENTES TOTAL	125
FIGURA 5.3: DIAGRAMA DE COMPONENTES TOTAL POR FASE DE INTEGRACIÓN	130

FIGURA 5.4: COMPONENTE NUEVO PROYECTO DIALOG RECIBIENDO LA DATA DE NUEVO PROYECTO	132
FIGURA 5.5: DIRECTORIO DE NUEVO PROYECTO GENERADO.....	133
FIGURA 5.6: CONTENIDO DEL DIRECTORIO DE NUEVO PROYECTO GENERADO.....	133
FIGURA 6.1: PANTALLA DE INICIO DEL SISTEMA	147
FIGURA 6.2: OPCIONES DE ARCHIVO DE PROYECTO EN LA BARRA DE MENÚ.....	148
FIGURA 6.3: OPCIONES DE ARCHIVO DE PROYECTO EN LA BARRA DE HERRAMIENTAS	149
FIGURA 6.4: OPCIONES DE PROYECTO O GESTIÓN DE UIS EN LA BARRA DE MENÚ.....	149
FIGURA 6.5: OPCIONES DE PROYECTO O GESTIÓN DE UIS EN LA BARRA DE HERRAMIENTAS	150
FIGURA 6.6: OPCIONES DE CONFIGURACIÓN DE SISTEMA.....	150
FIGURA 6.7: OPCIONES DE AYUDA DEL SISTEMA	151
FIGURA 6.8: MODO DE EDICIÓN	151
FIGURA 6.9: CONTROL DE VISTAS DEL PROYECTO	152
FIGURA 6.10: VISTA DE PROYECTO	152
FIGURA 6.11: VISTA DE COMPONENTES.....	153
FIGURA 6.12: PALETA DE COMPONENTES	154
FIGURA 6.13: CONTROL DE PROPIEDADES	155
FIGURA 6.14: DIALOGO DE NUEVO PROYECTO.....	156
FIGURA 6.15: DIALOGO PARA ABRIR PROYECTO.	157

FIGURA 6.16: DIALOGO DE NUEVA UI	158
FIGURA 6.17: DELIMITACIÓN DE NUEVO COMPONENTE	160
FIGURA 6.18: SELECCIÓN DE UN COMPONENTE.....	161
FIGURA 6.19: CÓDIGO GENERADO PARA UI EN EDICIÓN	162
FIGURA 6.20: CÓDIGO GENERADO PARA CLASE MANEJADORA DE EVENTOS DE LA UI EN EDICIÓN.....	162

ÍNDICE DE TABLAS

TABLA 3.1: GLOSARIO DE TÉRMINOS.....	66
TABLA 3.2: LISTA DE RIESGOS CRÍTICOS	69
TABLA 3.3: IDENTIFICACIÓN DE ACTORES.....	71
TABLA 3.4: MENSAJES DEL DIAGRAMA DE COLABORACIÓN PARA EL CASO DE USO “MODELAR INTERFAZ GRAFICA”. (1/2).....	94
TABLA 3.4: MENSAJES DEL DIAGRAMA DE COLABORACIÓN PARA EL CASO DE USO “MODELAR INTERFAZ GRAFICA”. (2/2).....	95
TABLA 3.5: MENSAJES DEL DIAGRAMA DE COLABORACIÓN PARA EL CASO DE USO “GENERAR CÓDIGO”.....	96
TABLA 3.6: MENSAJES DEL DIAGRAMA DE COLABORACIÓN PARA EL CASO DE USO “CONFIGURAR ARCHIVO”.....	97
TABLA 5.1: FORMATO DE ARCHIVO PRV	122
TABLA 5.2: PATRONES DE ARCHIVO STX.....	124
TABLA 5.3: CLASES DE EQUIVALENCIA DEL COMPONENTE ADDUIDIALOG (1/2).....	126
TABLA 5.3: CLASES DE EQUIVALENCIA DEL COMPONENTE ADDUIDIALOG (2/2).....	127
FUENTE: ELABORACIÓN PROPIA	127
TABLA 5.4: CLASES DE EQUIVALENCIA DEL COMPONENTE NUEVOPROYECTODIALOG	127
FUENTE: ELABORACIÓN PROPIA.....	127

TABLA 5.5: CASOS DE PRUEBA DE CAJA NEGRA PARA EL COMPONENTE ADDUIDIALOG	128
FUENTE: ELABORACIÓN PROPIA	128
TABLA 5.6: CASOS DE PRUEBA DE CAJA NEGRA PARA EL COMPONENTE NUEVOPROYECTODIALOG	128
FUENTE: ELABORACIÓN PROPIA	128
TABLA 5.7: LEYENDA DE COLORES PARA FASES DEL DIAGRAMA DE COMPONENTES TOTAL.....	131
TABLA 5.8: CASO DE PRUEBA PARA LA FASE DE CONFIGURACIÓN DE PROYECTO O FASE 3.....	131

RESUMEN

En el Dpto. de Computación y Sistemas del Núcleo de Anzoátegui, de la Universidad de Oriente, son dictadas las asignaturas Objetos y Abstracción de Datos, y Taller Objetos y Abstracción de Datos, siendo la segunda el apoyo práctico de la primera. Como herramienta de programación utilizada para la codificación de los proyectos en estas materias actualmente se usa Java el cual es un lenguaje de programación netamente orientado a aplicaciones gráficas y posee potentes herramientas para el desarrollo de GUI's. Debido a esto la enseñanza del uso de dichos objetos visuales requiere de un gran tiempo, y en consecuencia se ven reducidas las horas de clases dedicadas a los otros aspectos del contenido programático de las cátedras. En vista de esta situación, surge la necesidad de desarrollar un software con el que se pretende crear una herramienta de apoyo para estas materias. Esta herramienta debe permitir la generación automática de código fuente Java para GUI's diseñadas por el usuario, utilizando sencillos métodos gráficos. Dicho código seguirá las pautas de la Programación Orientada a Objetos y será presentado en forma elegante, esto con la finalidad de reforzar en los estudiantes los conocimientos necesarios para el desarrollo de una interfaz gráfica.

CAPÍTULO I

INTRODUCCIÓN

1.1 PLANTEAMIENTO DEL PROBLEMA

En el contexto del proceso de interacción persona - ordenador, la interfaz gráfica de usuario es el artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático. En otras palabras, la interfaz gráfica de usuario (en inglés Graphical User Interface, GUI) es el recurso que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en un software, habitualmente las acciones se realizan mediante manipulación directa para facilitar la interacción del usuario con la computadora.

En vista de la constante evolución tecnológica, la construcción de proyectos de software se ha enfocado en facilitar al usuario el desarrollo de tareas muy complejas y de alta dificultad de entendimiento a través de herramientas gráficas que permitan una mejor comunicación entre el hombre y la máquina, dándole a éste la posibilidad de adaptarse, teniendo más control y manejo de los programas. Es por esto, que la elaboración de GUI's, se ha convertido en una de las etapas más relevantes en el momento de llevar a cabo el diseño de un software.

La universidad como soporte principal en la formación de profesionales de las diferentes áreas del conocimiento, debe contar con distintos medios que sirvan como herramientas básicas para los estudiantes y profesores que están en búsqueda de

mayor eficiencia en el campo de su especialidad. En este contexto se encuentra la Universidad de Oriente, Núcleo Anzoátegui, que ofrece una diversidad de carreras

como la Ingeniería en Computación y la de Sistemas, que son especialidades centradas en nuevas tecnologías informáticas.

En el Departamento de Computación y Sistemas del Núcleo de Anzoátegui, de la Universidad de Oriente, son dictadas las asignaturas Objetos y Abstracción de Datos, (Código 072-2123) y Taller Objetos y Abstracción de Datos, (Código 072-2131), siendo la segunda el apoyo práctico de la primera. El contenido de estas cátedras es muy extenso y abarca aspectos de gran importancia en el área de programación, volviéndose una materia fundamental en las carreras tecnológicas antes mencionadas.

Como herramienta de programación utilizada para la codificación de los proyectos en estas materias actualmente se usa Java, el cual es un lenguaje de programación orientado a objetos de última generación y figura como uno de los lenguajes con un mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática. Java es netamente orientado a aplicaciones gráficas y posee potentes herramientas para el desarrollo de GUI's, por esto la enseñanza del uso básico de dichos objetos requiere de un gran tiempo tanto teórico como práctico, y debido a esto se ven reducidas las horas de clases dedicadas a los otros aspectos del contenido de la asignatura Objetos y Abstracción de Datos y el taller de la misma.

Dada la situación antes planteada, se formula la siguiente interrogante:

¿Por qué los estudiantes no hacen uso de una herramienta que optimice el proceso para desarrollar las GUI's de sus proyectos, mediante el uso de métodos más simples y cómodos? Esto con la finalidad de reducir las horas dedicadas a la enseñanza de componentes gráficos y destinar más tiempo en clase al aprendizaje de otros aspectos que son de más importancia en el área de la programación orientada a objetos.

Desde el punto de ingeniería de software, la interfaz de usuario juega un papel preponderante en el desarrollo y puesta en marcha de todo sistema. Es la carta de presentación del mismo y en ocasiones resulta determinante para la aceptación o rechazo de todo un proyecto.

En la actualidad la mayoría de las aplicaciones existentes que proporcionan herramientas para el diseño de GUI's no presentan el código fuente generado de una manera sencilla y ordenada que permita un fácil entendimiento para el usuario, detalles como estos hacen que un estudiante que apenas se inicie en el lenguaje de programación resulte confundido y no se interese en entender la manera en la que fue codificada la interfaz gráfica, sino que se limite a usar el producto que la herramienta les proporciona sin comprender el mismo. De esta forma sigue siendo necesario que se dediquen muchas horas de clases a solventar dudas en la codificación de GUI's.

En vista de esta situación, surge la necesidad de desarrollar un software con el que se pretende crear una herramienta de apoyo para la asignatura Taller de Objetos y Abstracción de Datos. Esta herramienta debe permitir la generación automática de código fuente, válido y fuertemente orientado a objetos, para GUI's. La aplicación también debe presentar el mencionado código en forma sencilla y elegante, esto con la finalidad de proveer a los estudiantes la posibilidad de finalizar sus proyectos y facilitar los conocimientos necesarios para el desarrollo de una interfaz gráfica, que corresponda a las exigencias del contenido programático vigente para así lograr el nivel de asimilación y comprensión que la asignatura necesita.

El entorno gráfico del sistema contará con una interfaz gráfica amigable y sencilla que permita a los usuarios construir las GUI's de manera fácil, utilizando recursos semejantes a los usados en un dibujo digital. En resumen, algunas de las funciones serán:

- El sistema permitirá crear proyectos de GUI's para Applets y Aplicaciones Java.
- El usuario será capaz de seleccionar con el ratón cada componente gráfico de un panel de botones relacionados a cada objeto y arrastrarlo a la ubicación que desee en la ventana o frame en edición.
- El sistema también permitirá aplicar esquemas de ordenamiento de componentes a cada ventana en edición.
- El usuario podrá editar las propiedades dimensionales de cada componente a través del ratón, observando resultados instantáneos.
- El sistema proporcionará opciones para editar las propiedades gráficas de cada componente a través de una tabla de propiedades.
- El usuario será capaz de acceder al código fuente generado cada vez que lo desee para ir verificando y aprendiendo la codificación de cada componente gráfico.
- El usuario también será capaz de asociar a cada objeto gráfico una gama de eventos los cuales serán codificados siguiendo la metodología de delegación de eventos.
- Por último el usuario podrá exportar el código fuente generado en archivos *.java direccionados a la ubicación deseada.

En lo que respecta al educador, sólo bastarán unas pocas horas de clases para presentar a los estudiantes el sistema y las nociones básicas del manejo del mismo, ya que la interfaz tendrá de manera explícita los pasos necesarios para llevar a cabo un Proyecto, que permitirá a los profesores de la materia delegar al sistema la enseñanza de los componentes gráficos del lenguaje de programación usado y continuar con el contenido programático de la materia.

Para el desarrollo del proyecto se empleará la metodología R.U.P., (“Rational Unified Process”), que no es un sistema con pasos firmemente establecidos, sino un conjunto de técnicas adaptables al contexto y junto al U.M.L., (“Lenguaje Unificado de Modelado”), constituyen la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

Para la implementación de la aplicación se utilizará como lenguaje de programación Java el cual proporciona las herramientas necesarias para el desarrollo de aplicaciones de toda índole, multiplataforma y con entornos gráficos complejos, lo cual es necesario para lograr que el sistema tenga un interfaz que ofrezca una realimentación significativa y sea consistente para hacer que el usuario se adapte más fácil.

1.2 OBJETIVOS

1.2.1 Objetivo General

Desarrollar un software que permita la generación automática de código fuente de interfaces gráficas de usuario (GUI), como apoyo a la asignatura “Taller de Objetos y Abstracción de datos”, impartida en el departamento de Computación y Sistemas.

1.2.2 Objetivos Específicos

1. Recolectar toda la información acerca de los componentes gráficos de Java que serán incluidos en el sistema.
2. Diseñar la arquitectura del software, definiendo todos los diagramas asociados.

3. Realizar el diseño de interfaz de la aplicación que servirá como medio de comunicación entre el programa y el usuario, así como también los módulos adicionales añadidos a la arquitectura.
4. Realizar la codificación de todos los componentes diseñados.
5. Efectuar pruebas, tanto de unidad como de integración, realizando las depuraciones respectivas.
6. Realizar los manuales de usuario y de mantenimiento de la aplicación.

CAPÍTULO II

MARCO TEÓRICO

2.1 ANTECEDENTES

En la Universidad de Oriente, Núcleo de Anzoátegui, los estudiantes de pre-grado del Departamento de Computación y Sistemas; han desarrollado diversos software para diferentes niveles de educación. Entre los cuales se encuentran:

2.1.1 “Diseño e implementación de un case orientado a la solución de problemas matemáticos utilizando las técnicas del organigrama o diagrama de flujo”. [1]

Resumen: Las herramientas CASE utilizan representaciones gráficas en la pantalla para ayudar a automatizar la planeación, el análisis, el diseño y la generación de software, es decir, los programas deben generarse en forma automática a partir de diseños, especificaciones o imágenes de alto nivel en una pantalla CASE. El código se puede generar a partir de tablas de decisión, reglas, diagramas de acciones, diagramas de eventos, diagramas de transición de estados, representaciones de los objetos, sus propiedades y relaciones, etc.

El diseño se basó en un I-CASE, el cual apoya todas las etapas del ciclo de vida clásico de la ingeniería del software, así como la generación de código con un solo depósito consistente y las técnicas de análisis y diseño orientado a objetos. En

este depósito, se almacenaron los conocimientos utilizados en el momento de creación.

Autor: Aquiles A. Barreto M. e Ydones B. López Z.

Institución: Universidad de Oriente, Departamento de Computación y Sistemas, Escuela de Ingeniería y Ciencias Aplicadas.

Nivel: Pregrado.

Fecha: 1997.

2.1.2 “Desarrollo de un tutor para la enseñanza de los algoritmos usados en las estructuras de datos utilizando técnicas de animación”. [2]

Resumen: Algunos estudiantes de Ingeniería de Computación en la Universidad de Oriente Núcleo de Anzoátegui, presentan deficiencias en el manejo de las estructuras de datos y los algoritmos relacionados con las mismas. La falta de dominio de estos temas, afecta sensiblemente el desarrollo del estudiante en el área de software. Para solventar estas deficiencias se desarrolló un sistema tutor que tiene las siguientes características: posee el conocimiento de los algoritmos usados en las estructuras de datos, cuenta con animación proporcionando un ambiente más ameno y dinámico, puede repetir una lección más de una vez, requiere de un entrenamiento mínimo, posee ayuda en línea basada en hipertextos. En el desarrollo de este proyecto, se empleó el análisis orientado a objetos, a través del cual se obtuvieron objetos reutilizables y para su codificación se usó el lenguaje de programación C++. Para la visualización del comportamiento de las diferentes estructuras de datos se usó la técnica de animación por arreglo gráfico (bitblt), que proporciona una actitud más activa al estudiante, y hace que el aprendizaje sea más efectivo. El sistema tutor no sustituirá la presencia del profesor, más bien servirá para reforzar el aprendizaje.

Autor: Claudio A. Cortínez N. y Jesús A. Martínez C.

Institución: Universidad de Oriente, Departamento de Computación y Sistemas, Escuela de Ingeniería y Ciencias Aplicadas.

Nivel: Pregrado.

Fecha: 1997.

2.1.3 “Desarrollo de un software interactivo para la enseñanza de la asignatura Química I (10-1814) del área científica-tecnológica”. [3]

Resumen: El Software Educativo para el aprendizaje de la Asignatura Química I, SEPAQ I, es un material educativo computarizado destinado a apoyar el proceso de enseñanza-aprendizaje de la Asignatura Química I, la cual pertenece a la Unidad de Estudios Básicos de la Universidad de Oriente, Núcleo Anzoátegui; esta aplicación será capaz de instruir, evaluar y corregir a los estudiantes que cursan dicha asignatura, de una manera amena y diferente, que los motivara a estudiar e investigar más sobre Química; busca fortalecer el proceso de aprendizaje dado en las aulas y disminuir así el número de alumnos reprobados en dicha asignatura por semestre. El SEPAQ I fue llevado a cabo implementando la metodología de software educativo orientado a objetos, está conformado por ocho micromundos (Estructura Atómica, Tabla Periódica, Enlace Químico, Orígenes de la Teoría Atómica, Estequiometría, Soluciones, Gases y Acerca del Software) correspondientes siete de ellos a los siete temas estudiados dentro de la asignatura Química I, y uno adicional de información para el usuario; cada micromundo consiste de un conjunto de páginas o unidades de información multimedia, contentivas de imágenes, gráficos, animaciones, texto, hipervínculos, botones de desplazamiento, etc., las cuales fueron implementadas usando como lenguajes de programación: JAVA y HTML.

Autor: Nairubia R. Rivas L.

Institución: Universidad de Oriente, Departamento de Computación y Sistemas, Escuela de Ingeniería y Ciencias Aplicadas.

Nivel: Pregrado.

Fecha: 2001.

2.2 CONOCIMIENTOS PREVIOS

2.2.1 Proceso unificado de desarrollo de software

También llamado Rational Unified Process en inglés, habitualmente resumido como RUP, es una infraestructura flexible de desarrollo de software que proporciona prácticas recomendadas probadas y una arquitectura configurable. Es un Proceso Práctico. [4]

2.2.1.1 Principios del RUP

Adaptar el proceso

El proceso deberá adaptarse a las características propias del proyecto u organización. El tamaño del mismo, así como su tipo o las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto.

Balancear prioridades

Los requerimientos de los diversos participantes pueden ser diferentes, contradictorios o disputarse recursos limitados. Debe encontrarse un balance que satisfaga los deseos de todos. Debido a este balanceo se podrán corregir desacuerdos que surjan en el futuro.

Demostrar valor iterativamente

Los proyectos se entregan, aunque sea de un modo interno, en etapas iteradas. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados.

Elevar el nivel de abstracción

Este principio dominante motiva el uso de conceptos reutilizables tales como patrón del software, lenguajes 4GL o marcos de referencia (frameworks) por nombrar algunos. Esto evita que los ingenieros de software vayan directamente de los requisitos a la codificación de software a la medida del cliente, sin saber con certeza qué codificar para satisfacer de la mejor manera los requerimientos y sin comenzar desde un principio pensando en la reutilización del código. Un alto nivel de abstracción también permite discusiones sobre diversos niveles y soluciones arquitectónicas. Éstas se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con el lenguaje UML.

Enfocarse en la calidad

El control de calidad no debe realizarse al final de cada iteración, sino en todos los aspectos de la producción. El aseguramiento de la calidad forma parte del proceso de desarrollo y no de un grupo independiente.

2.2.1.2 Ciclo de vida

El ciclo de vida RUP es una implementación del Desarrollo en espiral. Fue creado ensamblando los elementos en secuencias semi-ordenadas. El ciclo de vida organiza las tareas en fases e iteraciones.

RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades. En la Figura muestra cómo varía el esfuerzo asociado a las disciplinas según la fase en la que se encuentre el proyecto RUP.

Las primeras iteraciones (en las fases de Inicio y Elaboración) se enfocan hacia la comprensión del problema y la tecnología, la delimitación del ámbito del proyecto, la eliminación de los riesgos críticos, y al establecimiento de una baseline, (Línea base), de la arquitectura.

Durante la fase de inicio las iteraciones hacen mayor énfasis en actividades de modelado del negocio y de requerimientos.

En la fase de elaboración, las iteraciones se orientan al desarrollo de la baseline de la arquitectura, abarcan más los flujos de trabajo de requerimientos, modelo de negocios (refinamiento), análisis, diseño y una parte de implementación orientado a la baseline de la arquitectura.

En la fase de construcción, se lleva a cabo la construcción del producto por medio de una serie de iteraciones.

Para cada iteración se selecciona algunos Casos de Uso, se refina su análisis y diseño y se procede a su implementación y pruebas. Se realiza una pequeña cascada para cada ciclo. Se realizan tantas iteraciones hasta que se termine la implementación de la nueva versión del producto.

En la fase de transición se pretende garantizar que se tiene un producto preparado para su entrega a la comunidad de usuarios.

Como se puede observar en cada fase participan todas las disciplinas, pero que dependiendo de la fase el esfuerzo dedicado a una disciplina varía. [5]

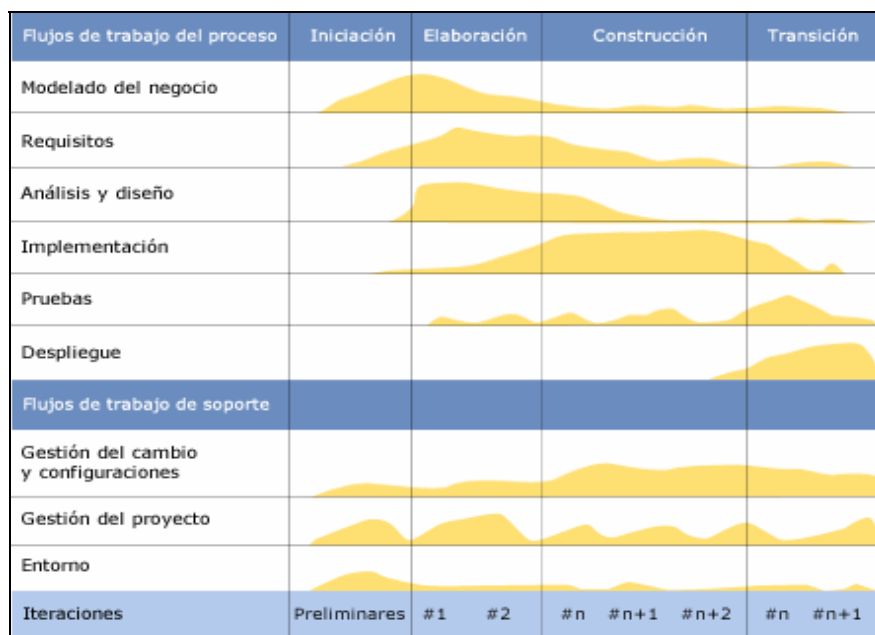


Figura 2.1: Esfuerzo en actividades según fase del proyecto

Fuente: Grupo Soluciones Innova. (2007).

2.2.2 Lenguaje Unificado de Modelado (UML)

El proceso unificado utiliza el UML para expresar gráficamente todos los esquemas de un software.

El Lenguaje Unificado de Modelado emergió en los '90, luego de la búsqueda de un lenguaje de modelado que unificara, que siguió a la “guerra de métodos” de los '70 y '80. A pesar que UML evolucionó primeramente de varios métodos orientados al objeto de segunda generación UML no es simplemente un lenguaje para el

modelado orientado a objetos de tercera generación. Su alcance extiende su uso más allá de sus predecesores. Y es la experiencia, experimentación y una gradual adopción del estándar lo que revelará su verdadero potencial y posibilitará a las organizaciones darse cuenta de sus beneficios.

2.2.2.1 Definición

UML es un lenguaje de modelado unificado basado en una notación grafica que permite: especificar, construir, visualizar y documentar los objetos de un sistema.

Se usa para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir. UML capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Un sistema se modela como una colección de objetos discretos que interactúan para realizar un trabajo que finalmente beneficia a un usuario externo.

UML es un lenguaje de propósito general para el modelado orientado a objetos; es también un lenguaje de modelamiento visual que permite una abstracción del sistema y sus componentes.

Un modelo captura una vista de un sistema del mundo real. Es una abstracción de dicho sistema, considerando un cierto propósito. Así, el modelo describe completamente aquellos aspectos del sistema que son relevantes al propósito del modelo, y a un apropiado nivel de detalle.

Un proceso de desarrollo de software debe ofrecer un conjunto de modelos que permitan expresar el producto desde cada una de las perspectivas de interés. El código fuente del sistema es el modelo más detallado del sistema (y además es

ejecutable). Sin embargo, se requieren otros modelos. Cada modelo es completo desde su punto de vista del sistema, sin embargo, existen relaciones de trazabilidad entre los diferentes modelos.

A través de un diagrama se puede realizar la representación gráfica de una colección de elementos de modelado, a menudo dibujada como un grafo conexo de arcos (relaciones) y vértices (otros elementos del modelo). Un diagrama no es un elemento semántico, un diagrama muestra representaciones de elementos semánticos del modelo, pero su significado no se ve afectado por la forma en que son representados.

La mayoría de los diagramas de UML y algunos símbolos complejos son grafos que contienen formas conectadas por rutas. La información está sobre todo en la topología, no en el tamaño o la colocación de los símbolos. Hay tres clases importantes de relaciones visuales: conexión, contención y adhesión. Así como también existen cuatro clases de construcciones gráficas que se usan en la notación de UML: íconos, símbolos bidimensionales, rutas y cadenas.

2.2.2.2 Utilidad de UML

- UML es un lenguaje para modelamiento de propósito general evolutivo, ampliamente aplicable, debe de ser soportado por herramientas e industrialmente estandarizado. Se aplica a una multitud de diferentes tipos de sistemas, dominios, y métodos o procesos.
- Como lenguaje de propósito general, se enfoca en el corazón de un conjunto de conceptos para la adquisición, compartición, y utilización de conocimientos emparejados con mecanismos de extensión.

- Como un lenguaje de modelamiento ampliamente aplicable, puede ser aplicado a diferentes tipos de sistemas (software y no-software), dominios (negocios versus software) y métodos o procesos.
- Como un lenguaje para modelamiento soportable por herramientas, las herramientas ya están disponibles para soportar la aplicación del lenguaje para especificar, visualizar, construir y documentar sistemas.
- Como un lenguaje para modelamiento industrialmente estandarizado, no es un lenguaje cerrado, propiedad de alguien, sino más bien, un lenguaje abierto y totalmente extensible reconocido por la industria.

UML posibilita captura, comunicación y nivelación de conocimiento estratégico, táctico y operacional para facilitar el incremento de valor, aumentando la calidad, reduciendo costos y reduciendo el tiempo de presentación al mercado; manejando riesgos y siendo proactivo para el posible aumento de complejidad o cambio.

2.2.2.3 Bloques Básicos de UML

- Elementos: Estructurales, comportamiento, agrupación, anotación.
- Relaciones: Dependencia, asociación, generalización, realización.
- Diagramas: Clases, objetos, casos de uso, secuencia, colaboración, estados, actividades, componentes, despliegue, capas.

2.2.2.3.1 Elementos del Lenguaje

- **Elementos estructurales:** Son los nombres de los modelos de UML. En su mayoría son las partes estáticas de un modelo, y representan conceptos o cosas materiales. Colectivamente, los elementos estructurales se denominan clasificadores.

Entre los elementos estructurales tenemos:

- a) **Actores:** Un actor es "algo" o "alguien" que puede interaccionar con el sistema que se está desarrollando (ver Figura 2.2).



Figura 2.2: Representación de un Actor.

Fuente: Benevento, M. y Sánchez, C. (2008)

- b) **Casos de uso:** Un caso de uso es una descripción de un conjunto de secuencias de acciones que un sistema ejecuta y que produce un resultado observable de interés para un actor particular (ver Figura 2.3).

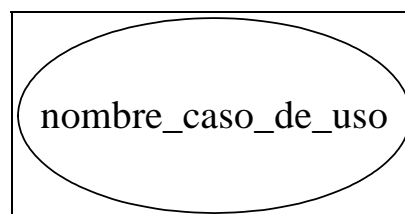


Figura 2.3: Representación de un Caso de Uso.

Fuente: Benevento, M. y Sánchez, C. (2008)

- c) **Clases:** Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica (ver Figura 2.4).

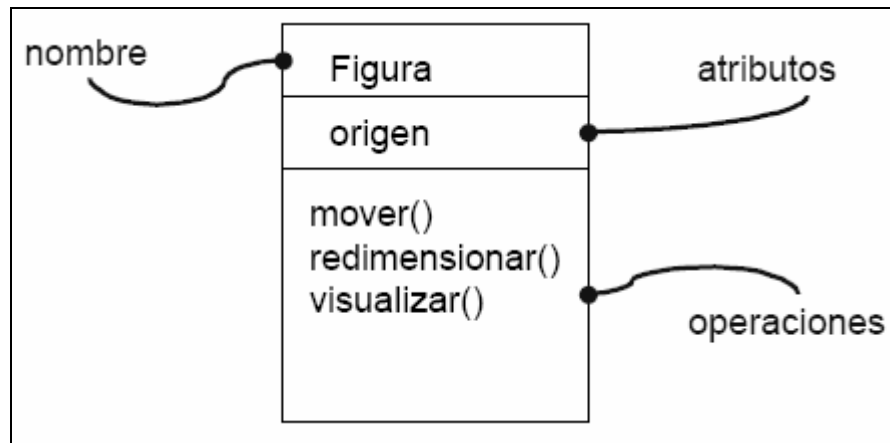


Figura 2.4: Representación de una Clase.

Fuente: Benevento, M. y Sánchez, C. (2008)

- d) **Objetos:** Un objeto es una instancia de alguna clase (ver Figura 2.5).

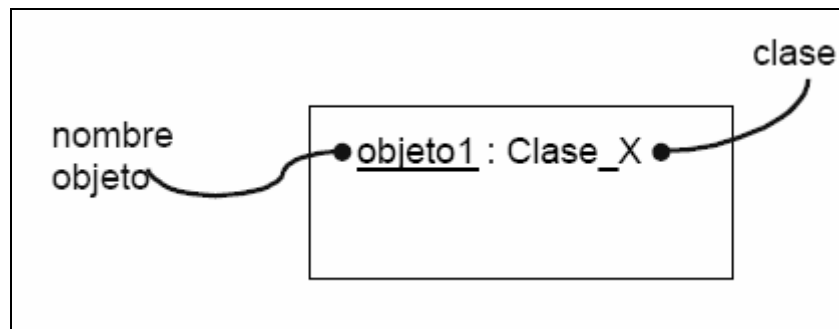


Figura 2.5. Representación de un Objeto.

Fuente: Benevento, M. y Sánchez, C. (2008)

- **Elementos de Comportamiento:** Son las partes dinámicas de los modelos UML. Éstos son los verbos de un modelo, y representan comportamiento en el tiempo y el espacio.
 - a) **Mensaje:** Los mensajes se usan para especificar una comunicación entre objetos (ver Figura 2.6). Se utilizan en los diagramas de secuencia.

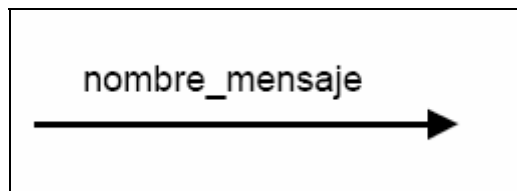


Figura 2.6: Representación de un Mensaje.

Fuente: Benevento, M. y Sánchez, C. (2008)

- **Elementos de Agrupación:** Son las partes organizativas de los modelos UML. Éstos son las cajas en las que puede descomponerse un modelo. Hay un tipo principal de elementos de agrupación: los paquetes.
 - a) **Paquete:** Sirve para organizar elementos en grupos. Un paquete es puramente conceptual (sólo existe en tiempo de desarrollo), como se puede observar en la Figura 2.7.

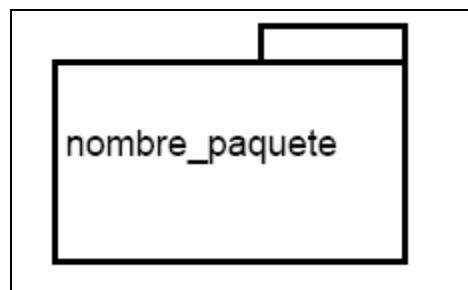


Figura 2.7: Representación de un Paquete.

Fuente: Benevento, M. y Sánchez, C. (2008)

2.2.2.3.2 Relaciones entre Elementos

a) Dependencia:

Es una relación semántica entre dos elementos (o dos conjuntos de elementos), en la cual un cambio en un elemento puede afectar a la semántica de otro elemento (ver Figura 2.8).



Figura 2.8: Representación de la Relación de Dependencia entre dos Clases.

Fuente: Benevento, M. y Sánchez, C. (2008)

Existen varios tipos de dependencia predefinidas que se indican mediante estereotipos, por ejemplo: «extend», e «include» para casos de uso.

b) Asociación:

Es una relación estructural entre dos elementos, que describe las conexiones entre ellos (suele ser bidireccional). Es la única relación permitida entre los actores y los casos de uso (refleja la comunicación existente entre un actor y un caso de uso).

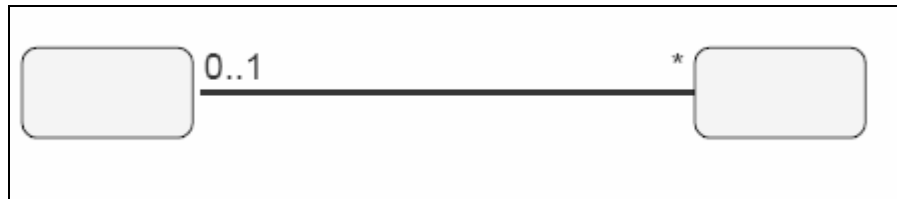


Figura 2.9: Representación de la Relación de Asociación entre Dos Clases.

Fuente: Benevento, M. y Sánchez, C. (2008)

c) Agregación:

Es una relación estructural entre un todo y sus partes. Se denota por una línea terminada en un "diamante" en el extremo de la clase que representa el todo.

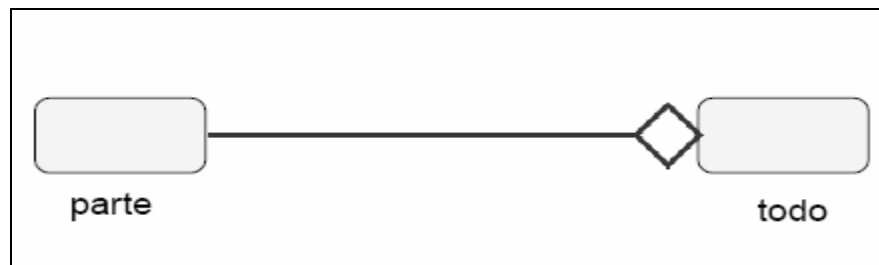


Figura 2.10: Representación de una Relación de Agregación entre Dos Clases.

Fuente: Benevento, M. y Sánchez, C. (2008)

d) Generalización:

Es una relación taxonómica entre un elemento más general (el padre) y un elemento más específico (el hijo). Se usa tanto en diagramas de clases como en diagramas de casos de uso.

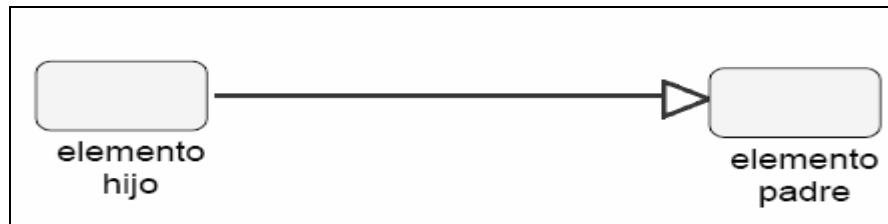


Figura 2.11: Representación de una Relación de Generalización entre Dos Clases.

Fuente: Benevento, M. y Sánchez, C. (2008)

2.2.2.3.3 Diagramas de UML

Los elementos de UML se muestran mediante diagramas que presentan múltiples vistas del sistema, ese conjunto de vistas son conocidos como modelos. UML presenta varios diagramas donde cada uno representa un aspecto del sistema, los cuales están conformados por Diagramas de estructura, Diagramas de comportamiento y Diagramas de interacción.

Diagramas de estructura, enfatizan en los elementos que deben existir en el sistema modelado:

- Diagrama de clases
- Diagrama de componentes
- Diagrama de objetos
- Diagrama de estructura compuesta
- Diagrama de despliegue
- Diagrama de paquetes

Diagramas de comportamiento, enfatizan en lo que debe suceder en el sistema modelado:

- Diagrama de actividades
- Diagrama de caso de usos
- Diagrama de estados

Diagramas de Interacción, un subtipo de diagramas de comportamiento, que enfatiza sobre el flujo de control y de datos entre los elementos del sistema modelado:

- Diagrama de secuencia
- Diagrama de colaboración
- Diagrama de tiempos
- Diagrama de vista de interacción

1. **Diagrama de Casos de Uso:** El diagrama de casos de uso se emplea para capturar información de cómo un sistema o negocio trabaja, o de cómo se desea que trabaje (ver Figura 2.12). Los casos de uso no son artefactos orientados a objetos, es una técnica para captura de requisitos. Captura la funcionalidad del sistema vista por los usuarios.

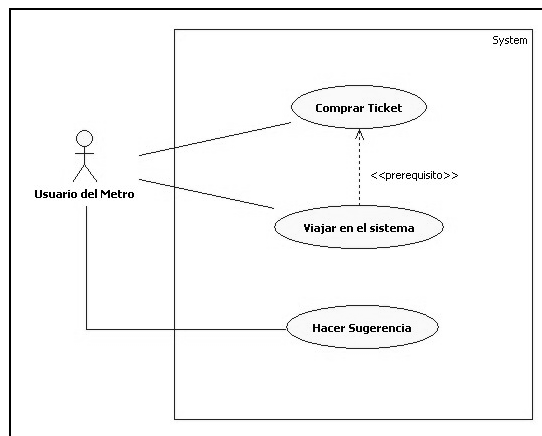


Figura 2.12: Diagrama de Casos de Uso.

Fuente: Benevento, M. y Sánchez, C. (2008)

2. **Diagrama de Secuencia:** El diagrama de secuencia muestra las interacciones entre los objetos organizadas en una secuencia temporal. En particular muestra los objetos participantes en la interacción y la secuencia de mensajes intercambiados (ver Figura 2.13). Representa una interacción, un conjunto de comunicaciones entre objetos organizadas visualmente por orden temporal. A diferencia de los diagramas de colaboración, los diagramas de secuencia incluyen secuencias temporales pero no incluyen las relaciones entre objetos. Pueden existir de forma de descriptor (describiendo todos los posibles escenarios) y en forma de instancia (describiendo un escenario real).

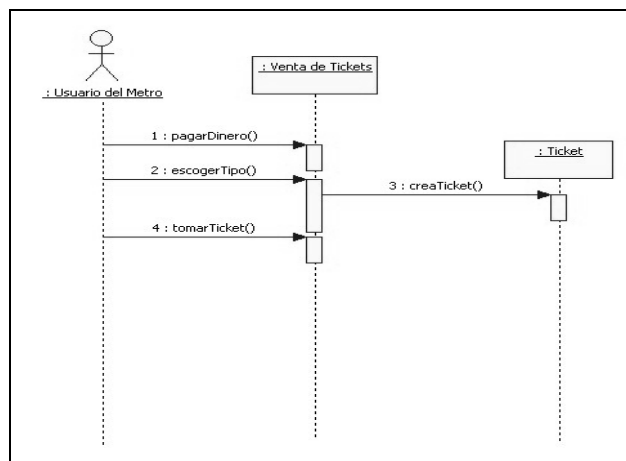


Figura 2.13: Diagrama de Secuencia.

Fuente: Benevento, M. y Sánchez, C. (2008)

3. **Diagrama de Colaboración:** Un diagrama de colaboración muestra la implementación de la operación (ver Figura 2.14). La colaboración muestra los parámetros y las variables locales de la operación, así como operaciones más permanentes; modela los objetos y los enlaces significativos dentro de una interacción. Un rol describe un objeto, y un rol en la asociación describe un

enlace dentro de una colaboración. Un diagrama de colaboración muestra los roles en la interacción en una disposición geométrica.

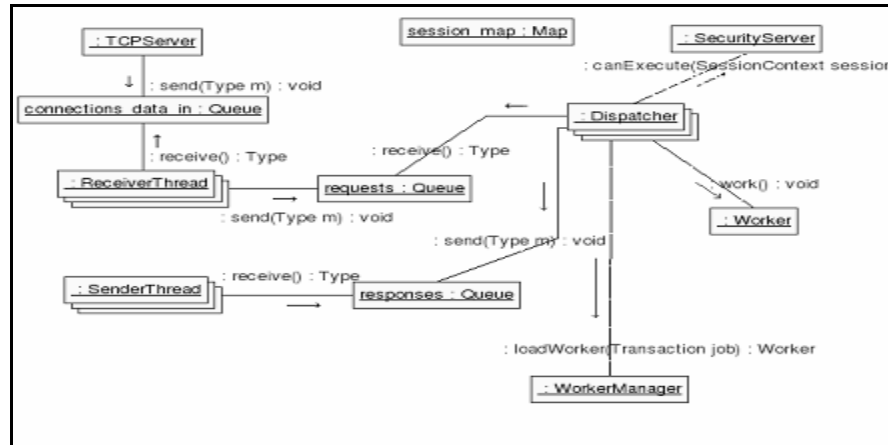


Figura 2.14: Diagrama de Colaboración.

Fuente: Benevento, M. y Sánchez, C. (2008)

4. **Diagrama de Clases:** Es un diagrama que muestra un conjunto de clases y sus colaboraciones y relaciones. Estos diagramas sirven para visualizar las relaciones existentes entre las distintas clases y las formas en que colaboran unas con otras (ver Figura 2.15). Las relaciones entre las distintas clases son las relaciones comunes existentes en UML aunque con matices: Una asociación se traduce como que desde los objetos de una clase se puede acceder a los objetos de otra. Una dependencia se puede visualizar como que la clase utilizada es un parámetro de un método de la clase que la utiliza. Una generalización se traduce como una herencia entre clases. Además de las clases en el análisis, se encuentran tres estereotipos fundamentales a saber: de interfaz, de control y de entidad.

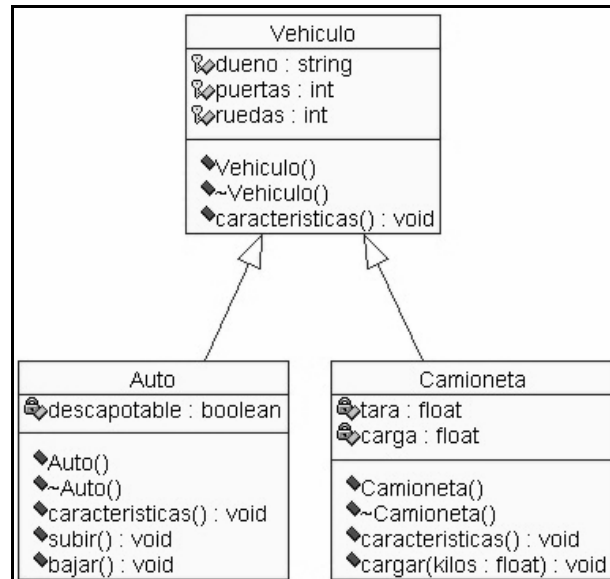


Figura 2.15: Diagrama de Clases.

Fuente: Benevento, M. y Sánchez, C. (2008)

5. **Diagrama de Despliegue:** Los diagramas de despliegue muestran la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos (ver Figura 2.16). La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación. Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria.

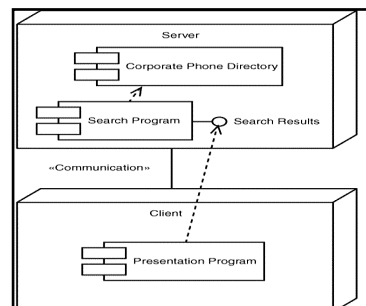


Figura 2.16: Diagrama de Despliegue.

Fuente: Benevento, M. y Sánchez, C. (2008)

6. **Diagrama de Componentes:** Es un diagrama que muestra un conjunto de componentes y sus relaciones (ver Figura 2.17). Los diagramas de componentes muestran los componentes de un sistema desde un punto de vista estático.

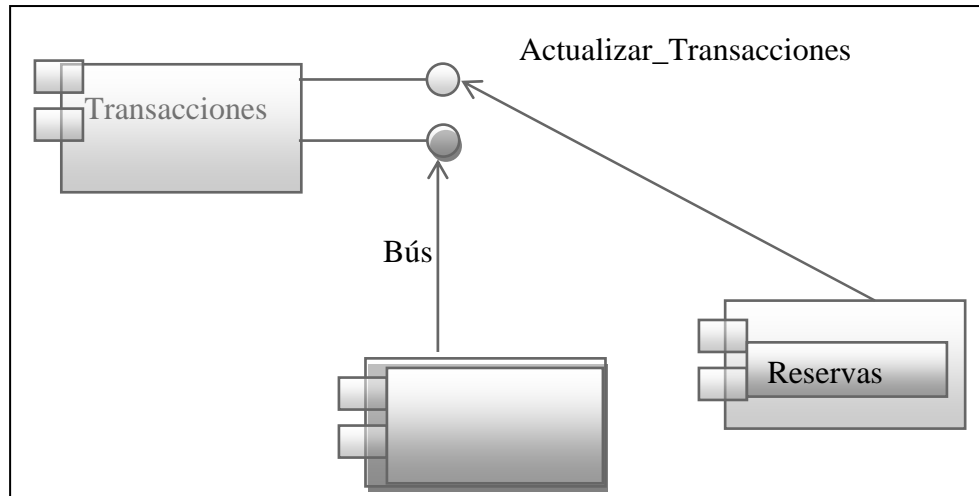


Figura 2.17: Diagrama de Componentes.

7. **Diagrama de Estructura Compuesta:** Un diagrama de estructura compuesta muestra las partes internas, los conectores y los puertos que implementan un componente (ver Figura 2.18). Cuando se instancia el componente, también se instancias las copias de sus partes internas.

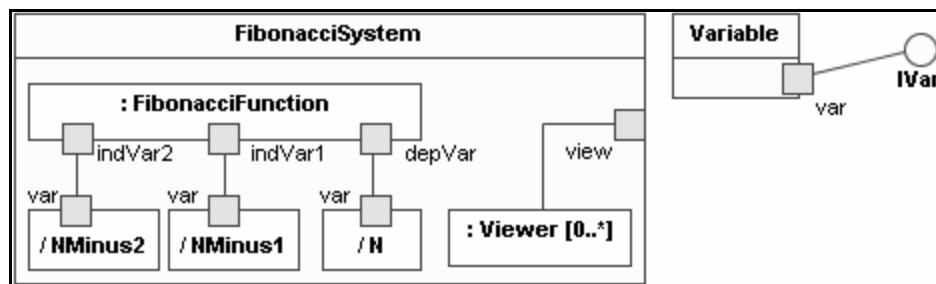


Figura 2.18: Diagrama de Estructura Compuesta.

Fuente: Benevento, M. y Sánchez, C. (2008)

8. **Diagrama de Objetos:** un diagrama de objetos representa un conjunto de objetos y sus relaciones (ver Figura 2.19). Se utilizan para describir estructuras de datos, instantáneas estáticas de las instancias de los elementos existentes en los diagramas de clases. Los diagramas de objetos abarcan la vista de diseño estática o la vista de procesos estática de un sistema al igual que los diagramas de clases, pero desde la perspectiva de casos reales o prototípicos.

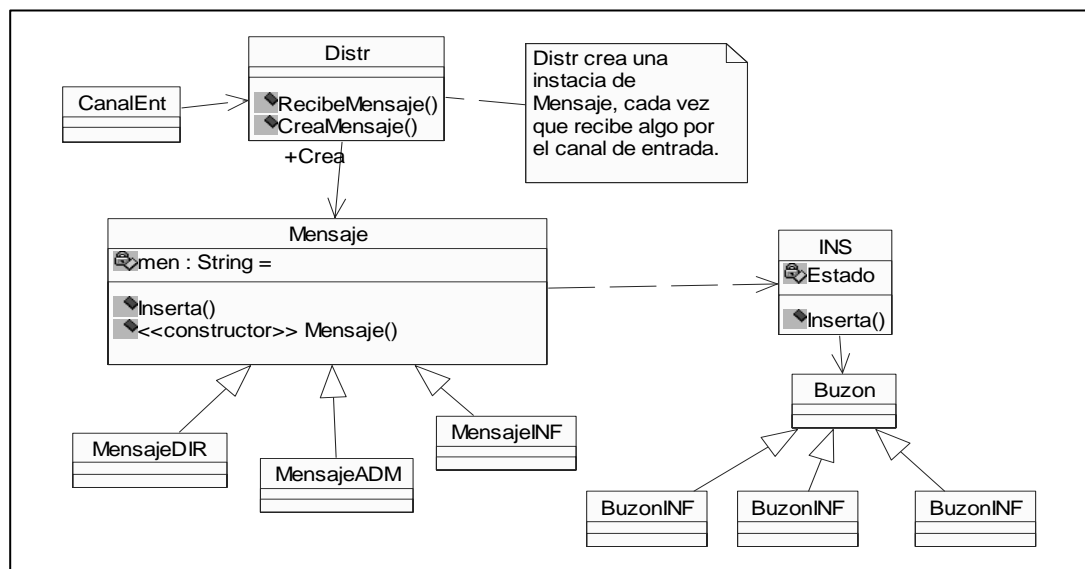


Figura 2.19: Diagrama de Objetos.

Fuente: Benevento, M. y Sánchez, C. (2008)

9. **Diagrama de Estados:** Un diagrama de estados representa una máquina de estados, constituida por estados, transiciones, eventos y actividades (ver Figura 2.20). Los diagramas de estados se utilizan para describir la vista dinámica de un sistema. Son especialmente importantes para modelar el comportamiento de una interfaz, una clase o una colaboración. Los diagramas de estados resaltan el

comportamiento dirigidos por eventos de un objeto, lo que es especialmente útil al modelar sistemas reactivos.

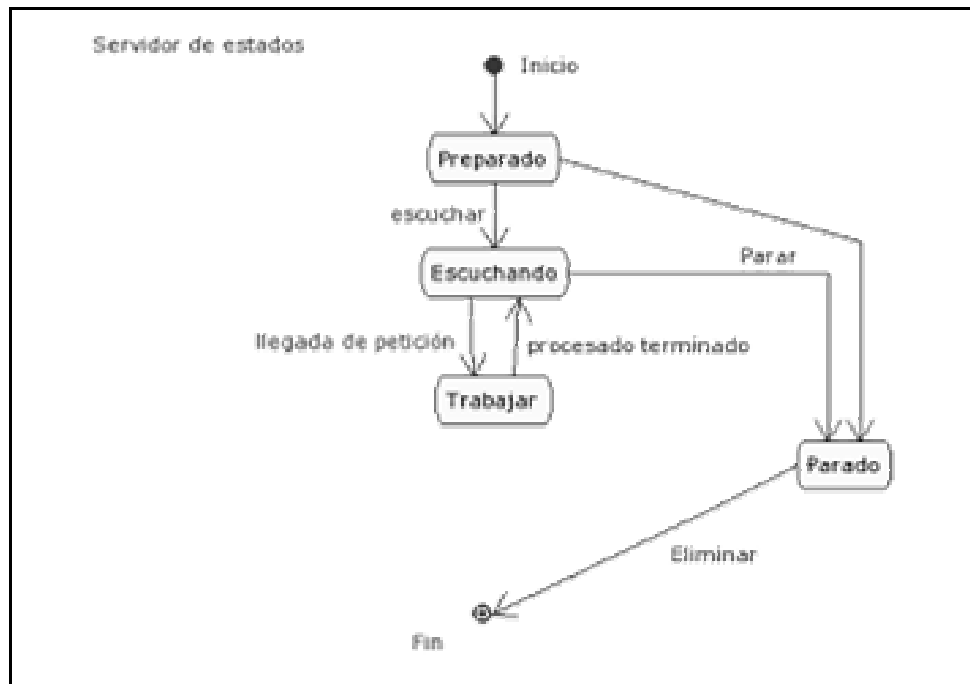


Figura 2.20. Diagrama de Estados.

Fuente: Benevento, M. y Sánchez, C. (2008)

10. Diagrama de Actividades: un diagrama de actividades muestra el flujo paso a paso en una computación (ver Figura 2.21). Una actividad muestra un conjunto de acciones, el flujo secuencial o ramificado de acción en acción, y los valores que son producidos o consumidos por las acciones. Los diagramas de actividades se utilizan para ilustrar la vista dinámica de un sistema.

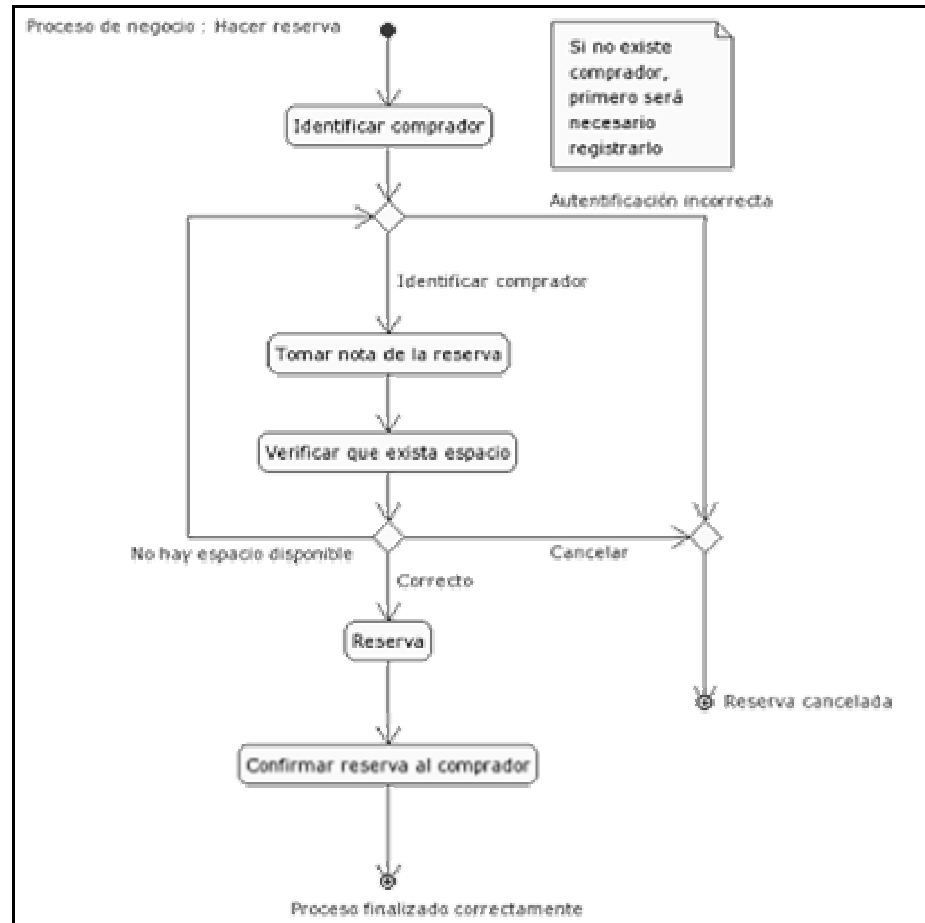


Figura 2.21: Diagrama de Actividades.

Fuente: Benevento, M. y Sánchez, C. (2008)

2.2.3 Ingeniería de Software con Modelaje Orientada a Objetos

El enfoque de orientación por objetos (O.O.) es un paradigma que también cubre el ciclo de vida del software y que permite tener un mayor acercamiento al mundo que se modela y cómo funciona este mundo. Con O.O. se puede hacer representación del mundo que se desea modelar en términos de los objetos que posee. Cada uno de ellos tiene sus propias características que lo identifican y un comportamiento específico,

así como también, se pueden usar los mecanismos de herencia y polimorfismo, para aprovechar las características y comportamiento de algunos objetos básicos. [6]

2.2.4 Programación Orientada a Objetos

2.2.4.1 Definición

La Programación Orientada a Objetos (POO u OOP según sus siglas en inglés) es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo, y encapsulamiento. Su uso se popularizó a principios de la década de 1990. Actualmente son muchos los lenguajes de programación que soportan la orientación a objetos.

2.2.4.2 Conceptos Fundamentales

La programación orientada a objetos es una nueva forma de programar que trata de encontrar una solución a estos problemas. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Entre ellos destacan los siguientes:

- **Clase:** definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ellas, (de c a d), Es la facilidad mediante la cual la clase D ha definido en ella cada uno de los atributos y operaciones de C, como si esos atributos y operaciones hubiesen sido definidos por la misma D.

- **Objeto:** entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos). Se corresponde con los objetos reales del mundo que nos rodea, o a objetos internos del sistema (del programa). Es una instancia a una clase.
- **Método:** algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.
- **Evento:** un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento, a la reacción que puede desencadenar un objeto, es decir la acción que genera.
- **Mensaje:** una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.
- **Propiedad o atributo:** contenedor de un tipo de datos asociados a un objeto (o a una clase de objetos), que hace los datos visibles desde fuera del objeto y esto se define como sus características predeterminadas, y cuyo valor puede ser alterado por la ejecución de algún método.
- **Estado interno:** es una variable que se declara privada, que puede ser únicamente accedida y alterada por un método del objeto, y que se utiliza para indicar distintas situaciones posibles para el objeto (o clase de objetos). No es visible al programador que maneja una instancia de la clase.

- **Componentes de un objeto:** atributos, identidad, relaciones y métodos.
- **Representación de un objeto:** un objeto se representa por medio de una tabla o entidad que esté compuesta por sus atributos y funciones correspondientes.

2.2.4.3 Características de la POO

Hay un cierto desacuerdo sobre exactamente qué características de un método de programación o lenguaje le definen como "orientado a objetos", pero hay un consenso general en que las características siguientes son las más importantes:

- **Abstracción:** Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar cómo se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos y cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción.
- **Encapsulamiento:** Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.
- **Principio de ocultación:** Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interfaz a otros objetos que especifica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no

tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no pueden cambiar el estado interno de un objeto de maneras inesperadas, eliminando efectos secundarios e interacciones inesperadas. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción. La aplicación entera se reduce a un agregado o rompecabezas de objetos.

- **Polimorfismo:** comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado. Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama asignación tardía o asignación dinámica. Algunos lenguajes proporcionan medios más estáticos (en "tiempo de compilación") de polimorfismo, tales como las plantillas y la sobrecarga de operadores de C++.
- **Herencia:** las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementarlo. Esto suele hacerse habitualmente agrupando los objetos en *clases* y estas en árboles o enrejados que reflejan un comportamiento común. Cuando un objeto hereda de más de una clase se dice que hay herencia múltiple. [6]

2.2.5 Interfaz de Usuario

2.2.5.1 Definición

La interfaz de usuario es el medio con que el usuario puede comunicarse con una máquina, un equipo o una computadora, y comprende todos los puntos de contacto entre el usuario y el equipo.

2.2.5.2 Principales Funciones de las Interfaces de Usuario

Sus principales funciones son:

- Puesta en marcha y apagado
- Control de las funciones manipulables del equipo
- Manipulación de archivos y directorios
- Herramientas de desarrollo de aplicaciones
- Comunicación con otros sistemas
- Información de estado
- Configuración de la propia interfaz y entorno
- Intercambio de datos entre aplicaciones
- Control de acceso
- Sistema de ayuda interactivo.

2.2.5.3 Tipos de Interfaces de Usuario

Según la forma de interactuar del usuario

Atendiendo a como el usuario puede interactuar con una interfaz, nos encontramos con varios tipos de interfaces de Usuario:

- Interfaces alfanuméricas (intérpretes de mandatos) que solo presentan texto.
- Interfaces gráficas de usuario (GUI, Graphics User Interfaces), las que permiten comunicarse con el ordenador de una forma muy rápida e intuitiva representando gráficamente los elementos de control y medida.
- Interfaces táctiles, que representan gráficamente un "panel de control" en una pantalla sensible que permite interaccionar con el dedo de forma similar a si se accionara un control físico.

Según su construcción

Pueden ser de hardware o de software:

Interfaces hardware: Se trata de un conjunto de controles o dispositivos que permiten la interacción hombre-máquina, de modo que permiten introducir o leer datos del equipo, mediante pulsadores, reguladores e instrumentos.

Interfaces software: Son programas o parte de ellos, que permiten expresar nuestros deseos al ordenador o visualizar su respuesta. [6]

2.2.6 Interfaces Gráficas de Usuario (GUI)

Todo aquello que hace posible la interacción con un sistema ejecutándose en una computadora.

Su calidad determina, entre otras cosas:

- Si el usuario acepta o no el sistema
- Si los diseñadores del sistema son elogiados o reprobados

- Si un sistema tiene éxito o fracasa en el mercado o la empresa

Al diseñar la interfaz de una aplicación, se debe tener en cuenta el deseo del usuario de enfrentarse a algo fácil, pero a la vez poderoso. [7]

2.2.7 Lenguaje de Programación Java

Es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Las aplicaciones Java están típicamente compiladas en un bytecode, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el bytecode es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del bytecode por un procesador Java también es posible.

La implementación original y de referencia del compilador, la máquina virtual y las bibliotecas de clases de Java fueron desarrolladas por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java Community Process, si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de Sun, algunas incluso bajo licencias de software libre. [8]

La versatilidad y eficiencia de la tecnología Java, la portabilidad de su plataforma y la seguridad que aporta, la han convertido en la tecnología ideal para su

aplicación a redes. De portátiles a centros de datos, de consolas de juegos a equipos científicos de alto rango, de teléfonos móviles a Internet, Java está en todas partes.

Más de 4.500 millones de dispositivos utilizan la tecnología Java, como:

- Más de 800 millones de equipos
- 2.100 millones de teléfonos móviles y otros dispositivos de mano (Fuente: Ovum)
- 3.500 millones de tarjetas inteligentes
- Sintonizadores, impresoras, cámaras web, juegos, sistemas de navegación para automóviles, terminales de lotería, dispositivos médicos, cajeros de pago en aparcamientos, etc. [9]

2.2.8 Aplicaciones

En informática, una aplicación es un tipo de programa diseñado para facilitar al usuario la realización de un determinado tipo de trabajo. Esto lo diferencia principalmente de otros tipos de programas como los sistemas operativos (que hace funcionar al ordenador), las utilidades (que realiza tareas de mantenimiento o de uso general), y los lenguajes de programación (con el cual se crean los programas informáticos), que realizan tareas más avanzadas y no pertinentes al usuario común.

2.2.9 Sub aplicación Java para Web o Applet

Un applet es un componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo un navegador web. El applet debe ejecutarse en un contenedor, que lo proporciona un programa anfitrión, mediante un plugin, o en

aplicaciones como teléfonos móviles que soportan el modelo de programación por applets.

A diferencia de un programa, un applet no puede ejecutarse de manera independiente, ofrece información gráfica y a veces interactúa con el usuario, típicamente carece de sesión y tiene privilegios de seguridad restringidos. Un applet normalmente lleva a cabo una función muy específica que carece de uso independiente.

CAPÍTULO III

FASE DE INICIO

En esta fase se establecerá una visión del proyecto y su alcance, donde se obtendrá el concepto inicial del sistema, mediante la identificación de sus funcionalidades y requisitos. Se tendrá como propósito la determinación del ámbito del sistema y de sus principales funciones, identificar los riesgos asociados al proyecto, realizar el modelado del dominio del negocio, como abstracción o esquema del modelo de casos de uso, el cual captura los tipos más importantes de objetos en el contexto del sistema, y proponer una visión general de la arquitectura candidata del sistema, asociado al sistema a desarrollar. En esta fase se detallarán todos los artefactos a generar por el proyecto. Se evaluará la viabilidad del proyecto.

Para llevar a cabo la fase de inicio se abarcaran los flujos de trabajo, requisitos y análisis, comenzando con la captura de requisitos mediante el conocimiento del contexto del sistema, dicho contexto se logra a través del establecimiento del diagrama de dominio del sistema, además se identificarán los riesgos que serán mitigados con la realización de las fases en el desarrollo del proyecto. A continuación se realizará la captura de requisitos y se identificarán los actores que interactúan con el sistema por medio del modelamiento de los diagramas de casos de uso. El siguiente flujo de trabajo es el de análisis, que permite modelar mediante los diagramas de clase de análisis para la comprensión de la información obtenida del flujo anterior. Por último, se realizarán los diagramas de colaboración para representar las interacciones entre los objetos, el diagrama de paquete de análisis para encapsular los casos de usos que fueron definidos al realizar el análisis del sistema.

3.1 CONTEXTO DEL SISTEMA

Cuando se habla de contexto se hace referencia a las circunstancias y condiciones que rodean un evento determinado, haciendo uso de este concepto se determinó el contexto del presente sistema, haciendo el análisis de los aspectos fundamentales que influyen en la construcción de una interfaz gráfica de usuario al nivel requerido por la asignatura objetos y abstracción de datos.

Este análisis se logró realizando visitas a los talleres de la materia y haciendo entrevistas a los estudiantes que cursan actualmente la asignatura (Periodo II-2009), esto con la finalidad de conocer directamente el proceso que llevan a cabo e identificar las debilidades de éste, para obtener un flujo de requisitos para el sistema. Dichos requisitos servirán de base como lineamiento a seguir durante el desarrollo de la aplicación propuesta.

3.1.1 Modelo de Dominio

El Modelo de Dominio es un modelo conceptual, empleado para comprender de una forma intuitiva el contexto del sistema y representar conceptos concernientes al dominio de una forma clara. Este modelo captura los tipos más importantes de objetos en el contexto del sistema. Muchos de los objetos del dominio pueden obtenerse de una especificación de requisitos o mediante entrevistas con los futuros usuarios. Los objetos o clases de negocio se relacionan mediante asociaciones, agregaciones y composiciones, lo cual es la manera de modelar el desarrollo de las actividades.

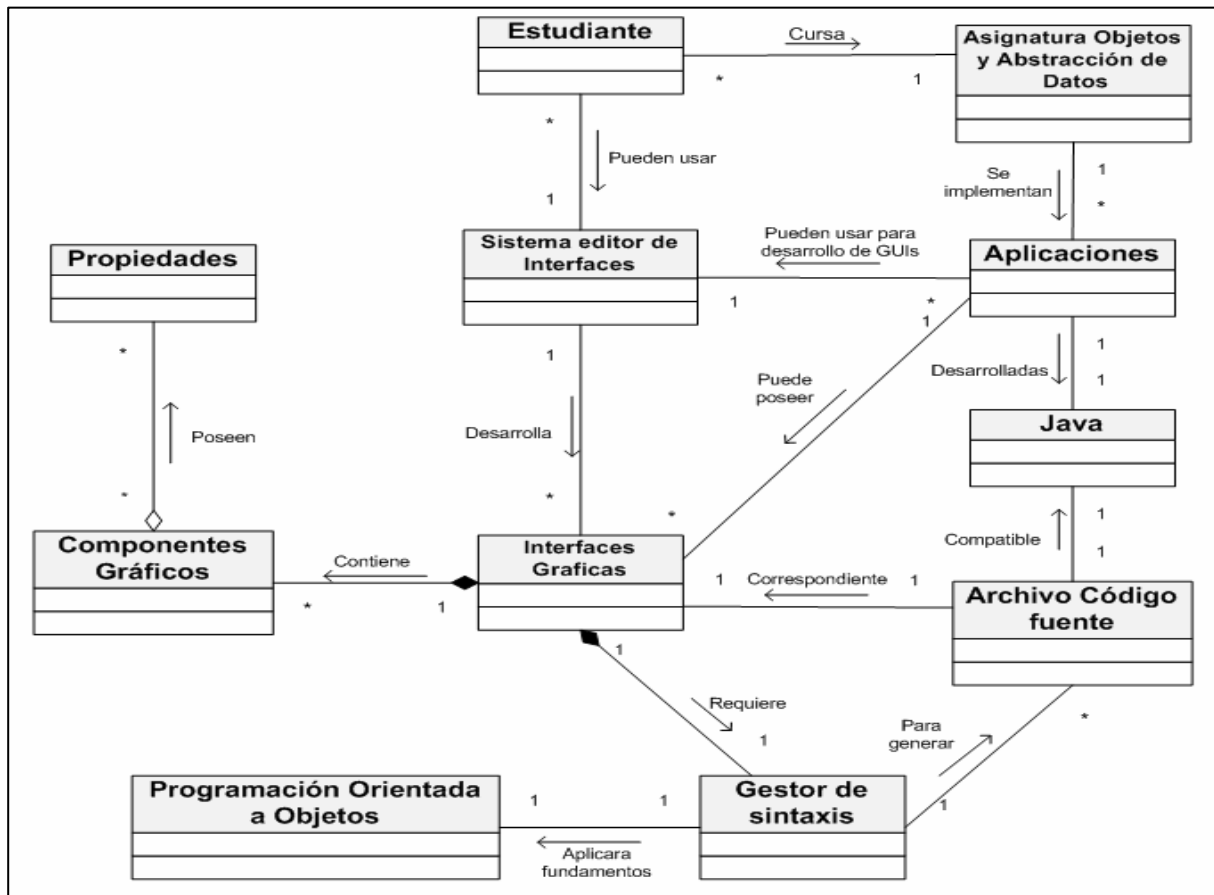


Figura 3.1: Modelo de Dominio.

Fuente: Elaboración propia

Para el modelo de dominio, tenemos las clases conceptuales del sistema por medio de las cuales tenemos una visión del proceso actualmente; así vemos como los Estudiantes que cursan la Asignatura Objeto y Abstracción de Datos podrán usar el Sistema editor de Interfaces para desarrollar las Interfaces Graficas de usuario de las Aplicaciones que implementen en dicha asignatura. Estas aplicaciones son desarrolladas en el lenguaje de programación Java y pueden poseer numerosas Interfaces graficas. También vemos que para cada interfaz, compuesta de

Componentes que poseen Propiedades, existe un Código fuente correspondiente. El código es compatible con Java y es generado por el Gestor de sintaxis del sistema el cual aplica fundamentos de la Programación Orientada a Objetos para generar la generación.

3.1.2 Glosario de términos para el modelo de dominio

Es necesario contar con un lenguaje común para facilitar el intercambio de ideas. El glosario o diccionario (ver Tabla 3.1) modelo incluye y define todos los términos que son de vital importancia para la comprensión del análisis que se realizará en las fases de desarrollo del presente estudio, los cuales se mencionan a continuación:

Tabla 3.1: Glosario de Términos.

TERMINO	DESCRIPCIÓN
Estudiante	Representa a la persona que hará uso del sistema.
Asignatura Objetos y Abstracción de Datos	Asignatura a la cual se desea prestar apoyo con la aplicación.
Aplicaciones	Programa diseñado para facilitar la realización de un determinado tipo de trabajo
Sistema editor de interfaces	Aplicación que genera código fuente de interfaces graficas.
Interfaces Graficas	Representa aquello que hace posible la interacción con un sistema ejecutándose en una computadora.
Java	Representa el lenguaje de programación usado en las asignaturas mencionadas.
Archivo código fuente	Archivo que contiene las instrucciones Java correspondientes a una Interfaz grafica de alguna aplicación.
Componentes Gráficos	Objetos y controles gráficos que componen una interfaz.
Propiedades	Atributos editables de un componente grafico.
Gestor de sintaxis	Modulo de generación de código y corrección de sintaxis.
Programación Orientada a Objetos	Paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora

Fuente: Elaboración propia

3.2 REQUISITOS FUNCIONALES

Los requerimientos establecen un contrato entre el sistema y su exterior, definen lo que se espera que realice el sistema, sin definir su funcionamiento interno. El objetivo final de cualquier diseño de la estructura de un sistema, es la de satisfacer los requerimientos de los usuarios para el sistema.

Se propone desarrollar un software con el que se pretende crear una herramienta de apoyo para la asignatura Taller de Objetos y Abstracción de Datos. Esta herramienta debe permitir la generación automática de código fuente, válido y fuertemente orientado a objetos, para GUI's. La aplicación también debe presentar el mencionado código en forma sencilla y elegante, esto con la finalidad de proveer a los estudiantes la posibilidad de finalizar sus proyectos y facilitar los conocimientos necesarios para el desarrollo de una interfaz gráfica, que corresponda a las exigencias del contenido programático vigente para así lograr el nivel de asimilación y comprensión que la asignatura necesita.

El entorno gráfico del sistema debe contar con una interfaz gráfica amigable y sencilla que permita a los usuarios construir las GUI's de manera fácil, utilizando recursos semejantes a los usados en un dibujo digital. En resumen, algunas de las funciones serian:

- El sistema permitirá crear proyectos de GUI's para Applets y Aplicaciones Java.
- El usuario será capaz de seleccionar con el ratón cada componente gráfico de un panel de botones relacionados a cada objeto y arrastrarlo a la ubicación que desee en la ventana o frame en edición.

- El sistema también permitirá aplicar esquemas de ordenamiento de componentes a cada ventana en edición.
- El usuario podrá editar las propiedades dimensionales de cada componente a través del ratón, observando resultados instantáneos.
- El sistema proporcionará opciones para editar las propiedades gráficas de cada componente a través de una tabla de propiedades.
- El usuario será capaz de acceder al código fuente generado cada vez que lo desee para ir verificando y aprendiendo la codificación de cada componente gráfico.
- El usuario también será capaz de asociar a cada objeto gráfico una gama de eventos los cuales serán codificados siguiendo la metodología de delegación de eventos.
- Por último el usuario podrá exportar el código fuente generado en archivos *.java direccionados a la ubicación deseada.

3.3 IDENTIFICACION DE RIESGOS

Los riesgos tienen como principal objetivo dirigir la viabilidad del proyecto a futuro, ya que estos pueden ocasionar pérdidas o daños, los cuales pongan en peligro el buen funcionamiento del sistema o en el peor de los casos que puedan afectar la culminación de dicho proyecto. En tal sentido, una vez identificados los posibles riesgos del sistema se priorizan y se define un plan de manejo para mitigarlos.

Una vez identificados pueden ser tratados de diversas maneras. Se cuenta fundamentalmente con cuatro elecciones: evitarlos, limitarlos, eliminarlos o controlarlos. Algunos riesgos pueden evitarse mediante una re-planificación del proyecto o un cambio en los requisitos; otros deberían restringirse de modo que solo afecten una parte del proyecto; otros atenuarse ejercitándolos y observándolos;

aunque existen riesgos que no pueden atenuarse, solamente pueden controlarse y observar si aparecen.

Tabla 3.2: Lista de riesgos críticos

RIESGO	CONTINGENCIA
Datos errados en el código generado.	Revisar la secuencia de generación de código ubicada en los archivos de sintaxis.
Incapacidad de uso del sistema por parte de usuarios finales.	Verificar el correcto diseño de interfaces y manual de usuario.
Escogencia de herramientas inadecuadas para el desarrollo del Sistema.	Investigar el alcance de estas herramientas y sus posibles sustituciones.

Fuente: Elaboracion propia

3.4 MODELO DE CASOS DE USO

El Modelo de casos de Uso se compone por varios objetos que intervienen en los procesos que un sistema es capaz de ejecutar, los cuales se describen mediante la identificación de casos de uso, identificación de actores y descripción de los casos de uso.

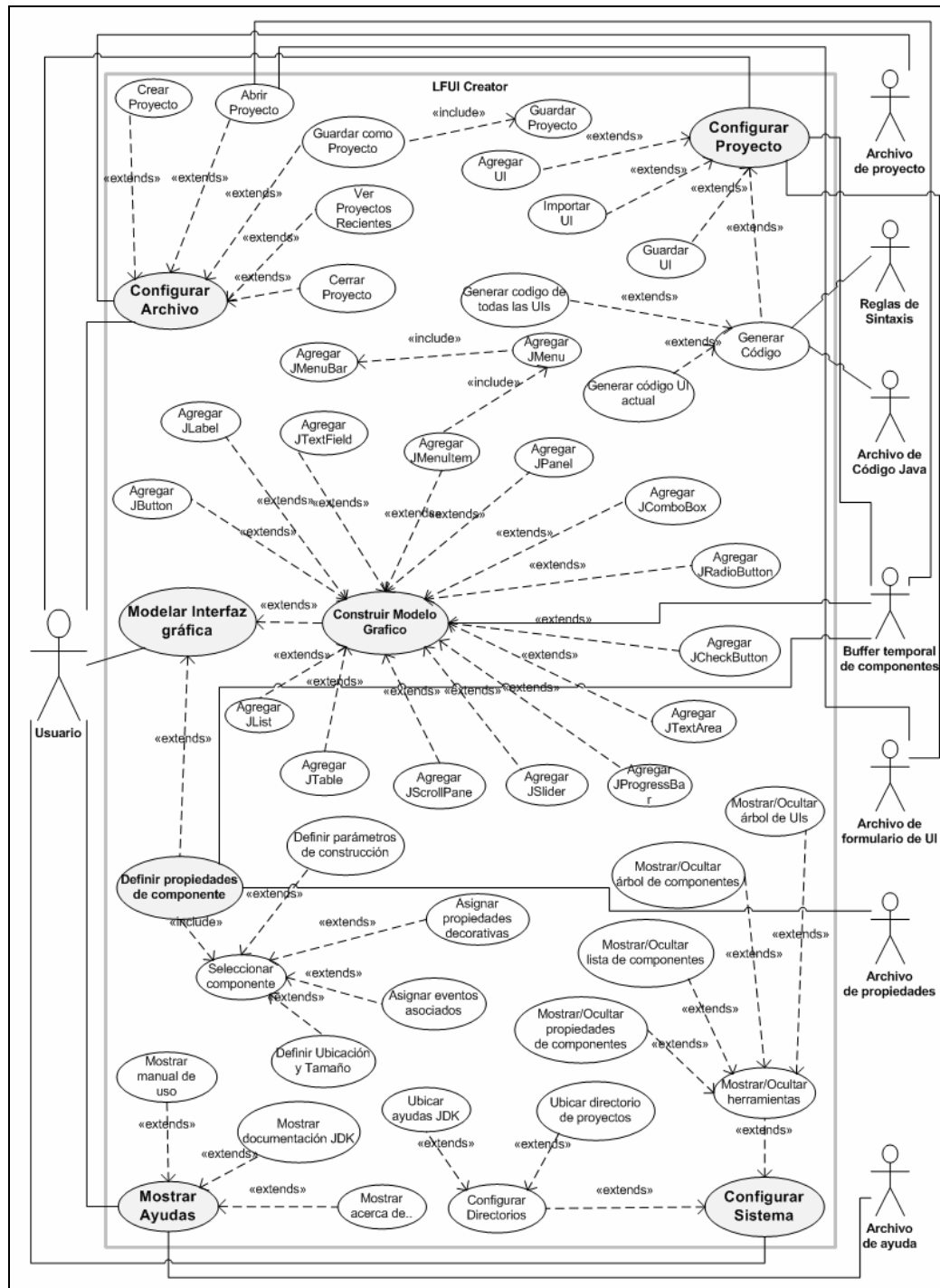


Figura 3.2: Modelo de Casos de uso.

Fuente: Elaboracion propia

3.4.1 Identificación de Actores.

Tabla 3.3: Identificación de actores.

ACTOR	DESCRIPCIÓN
Usuario	Usuario del sistema, quienes requieran construir una interfaz gráfica. Representado por estudiantes y profesores del departamento de Computación y Sistemas de la Universidad de Oriente.
Código Generado.	Es el fichero *.java generado por la aplicación como producto de la interfaz grafica construida por el usuario.
Reglas de sintaxis.	Es el fichero que contiene todas las reglas de sintaxis y escritura asociadas a cada instrucción necesitada en la generación de código.
Buffer temporal de componentes.	Estructura lógica que almacena temporalmente los componentes y propiedades seleccionadas por el usuario del sistema.
Archivo de propiedades.	Es el fichero que contiene todas las propiedades que serán posibles de editar a un componente grafico determinado.
Archivo de formulario.	Es el fichero que almacena el buffer temporal de componentes para su posterior recuperación en otra ocasión que el proyecto actual sea editado.
Archivo de proyecto.	Es el fichero que contiene la información general del proyecto, como lo es su ubicación, las interfaces graficas asociadas y la fecha de edición.
Archivo de ayuda.	Es el fichero que contiene una guía práctica de uso de la aplicación.

Fuente: Elaboracion propia

3.4.2 Descripción de casos de uso del sistema

Caso de Uso: *Configurar Archivo.*

Actores: Usuario, Archivo de proyecto.

Descripción: Este caso de uso le permite al usuario del sistema activar la opción de su escogencia dentro de las disponibles, para procesar datos de configuración del archivo de proyecto.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.

Flujo Principal:

- El usuario invoca el caso de uso Configurar archivo.
- El usuario selecciona la opción en el menú de Configurar archivo.
- El sistema despliega la interfaz asociada a la opción seleccionada.
- Fin del caso de uso.

Flujo Alternativo:

- El usuario puede salir del sistema.

Caso de Uso: Crear Proyecto.

Actores: Usuario, Archivo de proyecto.

Descripción: Este caso de uso le permite al usuario del sistema crear un proyecto e ingresar la data asociada al nuevo proyecto.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.
- El usuario debe activar la configuración de archivo.

Flujo Principal:

- El usuario invoca el caso de uso Configurar archivo.
- El usuario selecciona Crear Proyecto en el menú de Configurar archivo.
- El sistema despliega la interfaz asociada a Crear proyecto nuevo.
- El usuario carga la data del nuevo proyecto.
- El archivo de proyecto es generado.
- Fin del caso de uso.

Flujo Alternativo:

- El usuario puede salir del sistema.

Caso de Uso: *Abrir Proyecto.*

Actores: Usuario, Archivo de proyecto, Buffer de componentes, Archivo de Formulario UI.

Descripción: Este caso de uso le permite al usuario del sistema abrir un proyecto existente.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.
- El usuario debe activar la configuración de archivo.

Flujo Principal:

- El usuario invoca el caso de uso Configurar archivo.
- El usuario selecciona Abrir proyecto en el menú de Configurar archivo.
- El sistema despliega la interfaz asociada a Abrir proyecto.
- El usuario carga la ruta del proyecto.
- El archivo de proyecto es cargado y decodificado.

- Los archivos de formulario asociados al proyecto son cargados y decodificados.
- El buffer temporal de componentes relacionado a cada archivo de formulario es generado.
- Las UIs asociadas a cada archivo de formulario son desplegadas en el editor de interfaces.
- Fin del caso de uso.

Flujo Alternativo:

- El usuario puede salir del sistema.

Caso de Uso: Guardar Proyecto.

Actores: Usuario, Archivo de proyecto.

Descripción: Este caso de uso le permite al usuario del sistema guardar el proyecto en gestión.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.
- El usuario debe activar la configuración de archivo.
- El usuario debe activar el caso de uso Guardar como proyecto.

Flujo Principal:

- El usuario invoca el caso de uso Configurar archivo.
- El usuario selecciona Guardar proyecto en el menú de Configurar archivo.
- El archivo de proyecto es actualizado.
- Fin del caso de uso.

Flujo Alternativo:

- El usuario puede salir del sistema.

Caso de Uso: Guardar como Proyecto.

Actores: Usuario, Archivo de proyecto.

Descripción: Este caso de uso le permite al usuario del sistema guardar un proyecto existente especificando nueva data y ubicación.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.
- El usuario debe activar la configuración de archivo.

Flujo Principal:

- El usuario invoca el caso de uso Configurar archivo.
- El usuario selecciona Guardar como proyecto en el menú de Configurar archivo.
- El sistema despliega la interfaz asociada a Guardar como proyecto.
- El usuario carga la nueva ruta y nombre del proyecto.
- El archivo nuevo archivo de proyecto es generado.
- Fin del caso de uso.

Flujo Alternativo:

- El usuario puede salir del sistema.

Caso de Uso: Configurar Proyecto.

Actores: Usuario, Archivo de formulario de UI, Buffer temporal de componentes.

Descripción: Este caso de uso le permite al usuario del sistema activar la opción de su escogencia dentro de las disponibles, para gestionar las UIs y códigos generados asociados al proyecto.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.

Flujo Principal:

- El usuario invoca el caso de uso Configurar Proyecto.
- El usuario selecciona la opción en el menú de Configurar Proyecto.
- El sistema despliega la interfaz asociada a la opción seleccionada.
- Fin del caso de uso.

Flujo Alternativo:

- El usuario puede salir del sistema.

Caso de Uso: Agregar UI.

Actores: Usuario.

Descripción: Este caso de uso le permite al usuario del sistema agregar una nueva UI al proyecto en gestión, especificando el nombre y el tipo.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.
- El usuario debe activar la configuración de proyecto.

Flujo Principal:

- El usuario invoca el caso de uso Configurar Proyecto.
- El usuario selecciona Agregar UI en el menú de Configurar Proyecto.
- El sistema despliega la interfaz asociada a nueva UI.
- El usuario carga la data de la nueva interfaz.
- El sistema despliega una nueva UI para edición.
- Fin del caso de uso.

Flujo Alternativo:

- El usuario puede salir del sistema, para eso tendrá que confirmar si desea o no guardar los cambios en el proyecto y así guardar la nueva UI.

Caso de Uso: *Importar UI.*

Actores: Usuario.

Descripción: Este caso de uso le permite al usuario del sistema agregar una UI existente al proyecto en gestión, especificando la ubicación, esta UI será copiada al proyecto.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.
- El usuario debe activar la configuración de proyecto.

Flujo Principal:

- El usuario invoca el caso de uso Configurar Proyecto.
- El usuario selecciona Importar UI en el menú de Configurar Proyecto.
- El sistema despliega la interfaz asociada a importar UI.
- El usuario carga la ruta de la interfaz que desea agregar.

- El sistema despliega la UI para edición.
- Fin del caso de uso.

Flujo Alternativo:

- El usuario puede salir del sistema, para eso tendrá que confirmar si desea o no guardar los cambios en el proyecto y así guardar la UI agregada.

Caso de Uso: Guardar UI.

Actores: Usuario, Archivo de formulario de UI.

Descripción: Este caso de uso le permite al usuario del sistema guardar una UI existente y así generar el archivo de formulario el cual estará ubicado en la ruta del proyecto.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.
- El usuario debe activar la configuración de proyecto.

Flujo Principal:

- El usuario invoca el caso de uso Configurar Proyecto.
- El usuario selecciona guardar UI en el menú de Configurar Proyecto.
- Fin del caso de uso.

Flujo Alternativo:

- El usuario puede proceder a agregar componentes sin guardar la UI.
- El usuario puede salir del sistema.

Caso de Uso: Generar Código.

Actores: Usuario, Buffer temporal de componentes, Reglas de sintaxis, Archivo de código fuente.

Descripción: Este caso de uso le permite al usuario del sistema generar el código fuente correspondiente a una UI existente y así generar el archivo de código fuente el cual estará ubicado en la ruta del proyecto.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.
- El usuario debe activar la configuración de proyecto.
- El usuario debe agregar o importa una UI.

Flujo Principal:

- El usuario invoca el caso de uso Configurar Proyecto.
- El usuario selecciona Generar Código de UI en el menú de Configurar Proyecto.
- Fin del caso de uso.

Flujo Alternativo:

- El usuario puede proceder a agregar componentes a la UI.
- El usuario puede agregar otras UIs.
- El usuario puede editar las propiedades de los componentes en la UI en edición.
- El usuario puede salir del sistema.

Caso de Uso: Modelar Interfaz Gráfica.

Actores: Usuario.

Descripción: Este caso de uso le permite al usuario del sistema editar una UI existente, agregando componentes y editando sus propiedades.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.
- El usuario debe activar la configuración de proyecto.
- usuario debe agregar o importa una UI.

Flujo Principal:

- El usuario es libre de agregar componentes o editar las propiedades de la ventana en edición o cualquiera de los componentes agregados.

Flujo Alternativo:

- El usuario puede proceder generar el código de la UI.
- El usuario puede agregar otras UIs.
- El usuario puede salir del sistema.

Caso de Uso: *Construir Modelo Grafico.*

Actores: Usuario, Buffer temporal de componentes.

Descripción: Este caso de uso le permite al usuario del sistema editar una UI existente, agregando componentes de la paleta de componentes del sistema y editando directamente su tamaño y ubicación.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.
- El usuario debe activar la configuración de proyecto.

- usuario debe agregar o importa una UI.

Flujo Principal:

- El usuario selecciona el tipo de componente que desea agregar en la paleta de componentes.
- Por medio de la acción de arrastre del ratón define la ubicación y las dimensiones del componente que se está agregando.
- Puede modificar también usando el ratón los valores de ubicación y dimensión de cualquier componente agregado.

Flujo Alternativo:

- El usuario puede proceder generar el código de la UI.
- El usuario puede agregar otras UIs.
- El usuario puede salir del sistema.

Caso de Uso: *Definir propiedades de componente.*

Actores: Usuario, Buffer temporal de componentes, Archivo de propiedades.

Descripción: Este caso de uso le permite al usuario del sistema editar una UI existente, modificando las propiedades de cada componentes usando la ventana de propiedades.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.
- El usuario debe activar la configuración de proyecto.
- usuario debe agregar o importa una UI.
- El usuario debe agregar un componente.
- El usuario debe seleccionar un componente.

Flujo Principal:

- El usuario selecciona el componente que desea editar.
- El usuario llena los campos que desea modificar en la ventana de propiedades.
- EL usuario presiona el botón actualizar.

Flujo Alternativo:

- El usuario puede proceder generar el código de la UI.
- El usuario puede agregar otras UIs.
- El usuario puede agregar otros componentes.
- El usuario puede salir del sistema.

Caso de Uso: *Seleccionar componente.*

Actores: Usuario.

Descripción: Este caso de uso le permite al usuario del sistema seleccionar un componente agregado.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.
- El usuario debe activar la configuración de proyecto.
- usuario debe agregar o importa una UI.
- El usuario debe agregar un componente.

Flujo Principal:

- El usuario hace clic sobre el componente que desea seleccionar.

Flujo Alternativo:

- El usuario puede proceder generar el código de la UI.
- El usuario puede agregar otras UIs.

- El usuario puede agregar otros componentes.
- El usuario puede salir del sistema.

Caso de Uso: Definir parámetros de construcción.

Actores: Usuario, Archivo de propiedades.

Descripción: Este caso de uso le permite al usuario del sistema editar una UI existente, modificando las propiedades de construcción de cada componente como lo son; Nombre, Título, Esquema, etc.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.
- El usuario debe activar la configuración de proyecto.
- usuario debe agregar o importa una UI.
- El usuario debe agregar un componente.
- El usuario debe seleccionar un componente.

Flujo Principal:

- El usuario selecciona el componente que desea editar.
- El usuario llena los campos que desea modificar en la ventana de propiedades.
- EL usuario presiona el botón actualizar.

Flujo Alternativo:

- El usuario puede proceder generar el código de la UI.
- El usuario puede agregar otras UIs.
- El usuario puede agregar otros componentes.
- El usuario puede salir del sistema.

Caso de Uso: Agregar propiedades decorativas.

Actores: Usuario, Archivo de propiedades.

Descripción: Este caso de uso le permite al usuario del sistema editar una UI existente, modificando las propiedades de construcción de cada componente como lo son; Color de Fondo, Color de fuente, Icono, etc.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.
- El usuario debe activar la configuración de proyecto.
- usuario debe agregar o importa una UI.
- El usuario debe agregar un componente.
- El usuario debe seleccionar un componente.

Flujo Principal:

- El usuario selecciona el componente que desea editar.
- El usuario activa las interfaces de selección de cada valor.
- El usuario selecciona los valores deseados.
- EL usuario presiona el botón actualizar.

Flujo Alternativo:

- El usuario puede proceder generar el código de la UI.
- El usuario puede agregar otras UIs.
- El usuario puede agregar otros componentes.
- El usuario puede salir del sistema.

Caso de Uso: Agregar eventos asociados.

Actores: Usuario, Archivo de propiedades.

Descripción: Este caso de uso le permite al usuario del sistema agregar métodos para escuchar los eventos relacionados a un tipo de componente, haciendo uso de la ventana de propiedades.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.
- El usuario debe activar la configuración de proyecto.
- usuario debe agregar o importa una UI.
- El usuario debe agregar un componente.
- El usuario debe seleccionar un componente.

Flujo Principal:

- El usuario selecciona el componente que desea editar.
- El usuario activa las casillas relacionadas a cada tipo de evento.
- EL usuario presiona el botón actualizar.

Flujo Alternativo:

- El usuario puede proceder generar el código de la UI.
- El usuario puede agregar otras UIs.
- El usuario puede agregar otros componentes.
- El usuario puede salir del sistema.

Caso de Uso: Definir ubicación y tamaño.

Actores: Usuario, Archivo de propiedades.

Descripción: Este caso de uso le permite al usuario del sistema editar la ubicación y el tamaño de un componente haciendo uso también de la ventana de propiedades del sistema.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.
- El usuario debe activar la configuración de proyecto.
- usuario debe agregar o importa una UI.
- El usuario debe agregar un componente.
- El usuario debe seleccionar un componente.

Flujo Principal:

- El usuario selecciona el componente que desea editar.
- El usuario llena los campos que desea modificar en la ventana de propiedades.
- EL usuario presiona el botón actualizar.

Flujo Alterno:

- El usuario puede proceder generar el código de la UI.
- El usuario puede agregar otras UIs.
- El usuario puede agregar otros componentes.
- El usuario puede salir del sistema.

Caso de Uso: *Mostrar Ayudas.*

Actores: Usuario, Archivo de ayudas.

Descripción: Este caso de uso le permite al usuario del sistema activar la opción de su escogencia para mostrar el manual de usuario del sistema, acerca del sistema y la documentación del Java Development Kit.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.

Flujo Principal:

- El usuario invoca el caso de uso Mostrar ayudas.
- El usuario selecciona la opción en el menú de Mostrar ayudas.
- El sistema despliega la interfaz asociada a la opción seleccionada.
- Fin del caso de uso.

Flujo Alternativo:

- El usuario puede salir del sistema.

Caso de Uso: Configurar Sistema.

Actores: Usuario.

Descripción: Este caso de uso le permite al usuario del sistema activar la opción de su escogencia y mostrar el menú de vistas del sistema y configuración de directorios.

Pre-Condición:

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.

Flujo Principal:

- El usuario invoca el caso de uso Configurar Sistema.
- El usuario selecciona la opción en el menú de Configurar Sistema.
- El sistema despliega la interfaz asociada a la opción seleccionada.
- Fin del caso de uso.

Flujo Alternativo:

- El usuario puede salir del sistema.

Caso de Uso: Configurar Directorios.**Actores:** Usuario.**Descripción:** Este caso de uso le permite al usuario del sistema seleccionar la ruta que tendrán los proyectos por defecto y la ubicación de la documentación del JDK.**Pre-Condición:**

- El usuario debe activar el sistema.
- El sistema debe iniciarse y activar el menú principal.

Flujo Principal:

- El usuario invoca el caso de uso Configurar Sistema.
- El usuario selecciona la opción Configurar Directorios en el menú de Configurar Sistema.
- El sistema despliega la interfaz asociada a Directorios.
- El usuario carga las ubicaciones de los directorios.
- Fin del caso de uso.

Flujo Alternativo:

- El usuario puede salir del sistema.

Caso de Uso: Mostrar/Ocultar herramientas.**Actores:** Usuario.**Descripción:** Este caso de uso le permite al usuario del sistema activar o desactivar las ventanas de herramientas del sistema (Paleta de componentes, Ventana de propiedades, Vista de proyecto y Vista de UI).**Pre-Condición:**

- El usuario debe activar el sistema.

- El sistema debe iniciarse y activar el menú principal.

Flujo Principal:

- El usuario invoca el caso de uso Configurar Sistema.
- El usuario selecciona la opción Mostrar/Ocultar Herramientas en el menú Configurar Sistema.
- El usuario activa o desactiva la herramienta q desee.
- Fin del caso de uso.

Flujo Alternativo:

- El usuario puede salir del sistema.

3.5 REQUISITOS ADICIONALES DEL SISTEMA

Ambiente de Desarrollo: el software debe desarrollarse usando como herramienta principal un lenguaje de programación y un IDE especializado en el lenguaje seleccionado. Dicho lenguaje será orientado a objetos.

Plataforma Hardware: el software debe instalarse en equipos que estén a disposición de todos los estudiantes del Departamento de Computación y Sistemas que requieran de una herramienta para desarrollar Interfaces Graficas de usuario para aplicaciones o proyectos de software.

Ya finalizado el primer flujo de trabajo (captura de requisitos) en esta fase de inicio, con el uso de la identificación del contexto del sistema y los requisitos funcionales esenciales, se procede a trabajar en el segundo flujo de trabajo de esta fase, el de análisis, en el cual se profundizará en lo referente a las funciones del sistema antes mencionadas.

3.6 ANÁLISIS

En esta etapa se analizan, refinan y estructuran los requisitos encontrados en el flujo de trabajo precedente, es decir, la captura de requisitos. La idea es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que nos ayude a estructurar el sistema entero incluyendo su arquitectura. El propósito fundamental del análisis es resolver los casos de uso analizando los requisitos con mayor profundidad.

3.6.1 Diagramas de Clases de analisis

Las clases de análisis, también llamados objetos de análisis, son clases estereotipadas que representan un modelo conceptual para los elementos del sistema que tienen responsabilidad y comportamiento. Hay tres tipos de clases de análisis, las cuales son usadas en todo el modelo de análisis:

- Clases de interfaz
- Clases de entidad
- Clases de control

Una clase de interfaz es una clase estereotipada que modela la interacción entre uno o más actores (usuarios y sistemas externos) y el sistema. Esta interacción a menudo implica recibir (y presentar) información y peticiones de (y hacia) los usuarios y los sistemas externos. Las clases de interfaz modelan las partes del sistema que dependen de sus actores, lo cual implica que clarifican y reúnen los requisitos en los límites del sistema.

Las clases de entidad, por su parte, se utilizan para modelar información que posee una vida larga y que es a menudo persistente. Modela la información y el

comportamiento asociado de algún fenómeno o concepto, como una persona, un objeto del mundo real o un suceso del mundo real.

Las clases de control representan coordinación, secuencia, transacciones y control de otros objetos, y se usan con frecuencia para encapsular el control de un caso de uso en concreto.

Los diagramas de clases de análisis siempre están asociados a una clase, a una operación o a un caso de uso en particular, en este caso están asociados a los casos de uso explicados con anterioridad.

3.6.1.1 Diagrama de clases de análisis para el caso de uso “Modelar Interfaz Grafica”

Nota: (.....) La línea punteada indica que para que para que este caso de uso pueda darse, el usuario tiene que estar asociado a un proyecto. La configuración del archivo de proyecto se explica en el diagrama de colaboración relacionado a dicho caso de uso.

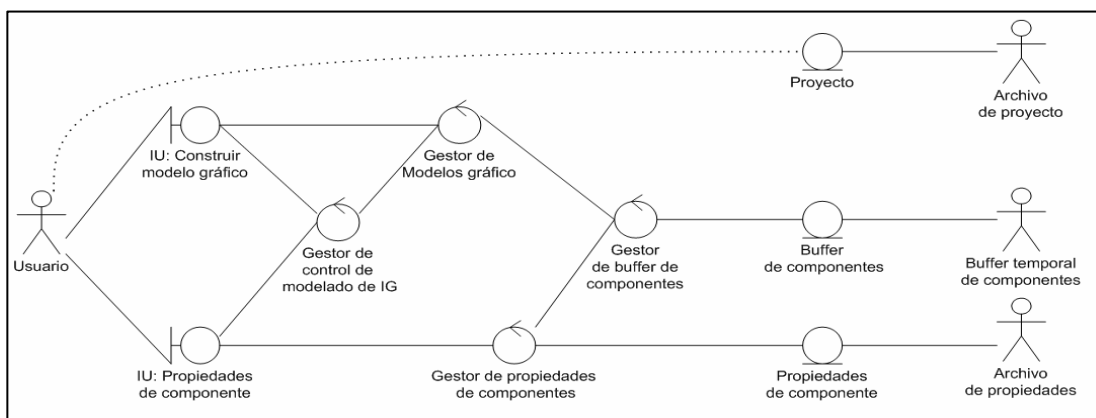


Figura 3.3: Diagrama de clases de análisis para el caso de uso “Modelar Interfaz Grafica”

Fuente: Elaboracion propia

3.6.1.2 Diagrama de clases de análisis para el caso de uso “Generar Código”

Nota: (.....) La línea punteada indica que para que para que este caso de uso pueda darse, el usuario tiene que estar asociado a un proyecto. La configuración del archivo de proyecto se explica en el diagrama de clases de analisis relacionado a dicho caso de uso.

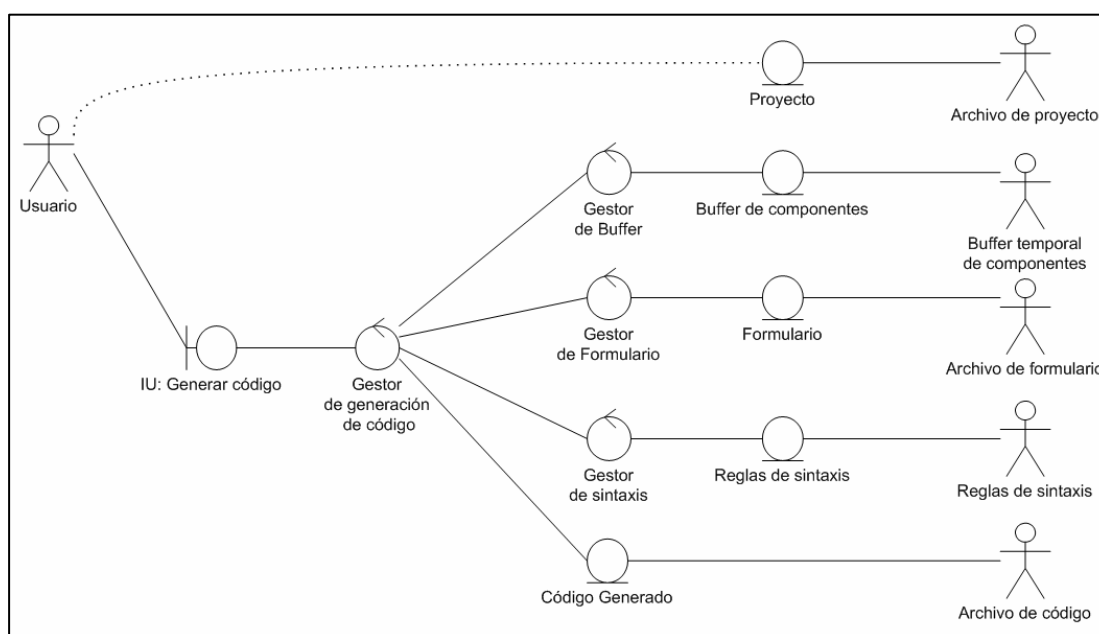


Figura 3.4: Diagrama de clases de análisis para el caso de uso “Generar Código”

Fuente: Elaboracion propia

3.6.1.3 Diagrama de clases de análisis para el caso de uso “Configurar Archivo”

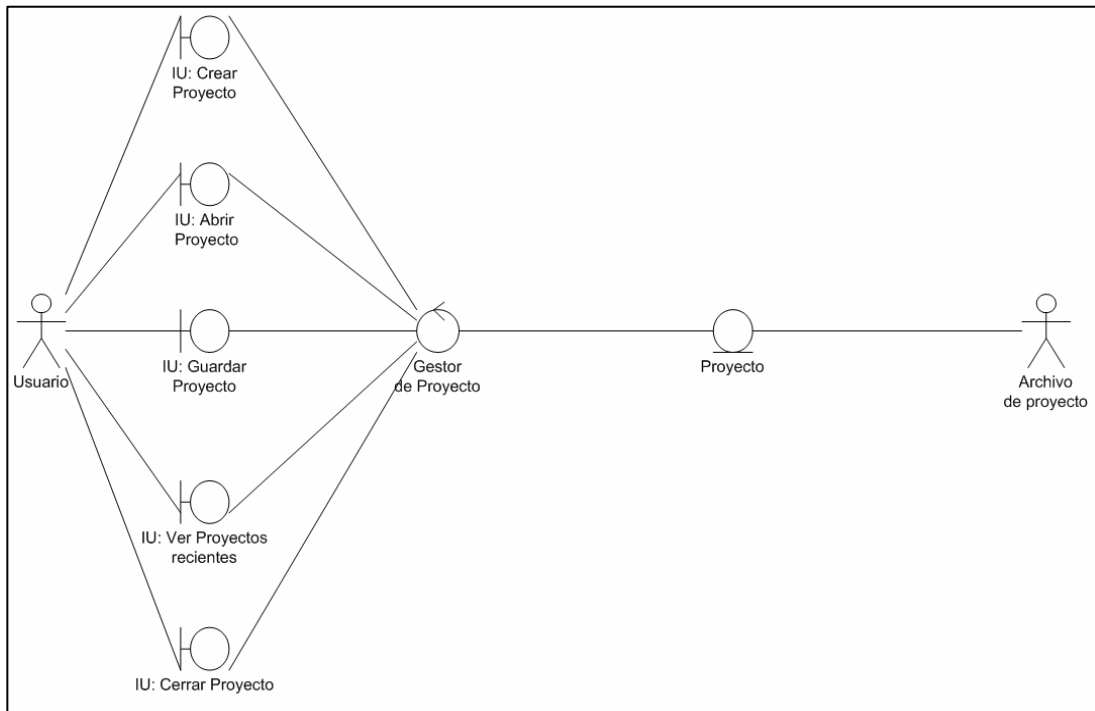


Figura 3.5: Diagrama de clases de análisis para el caso de uso “Configurar Archivo”

Fuente: Elaboración propia

3.6.2 Diagramas de Colaboración

En los diagramas de colaboración, se muestran las interacciones entre objetos creando enlaces entre ellos y añadiendo mensajes a esos enlaces. El nombre de un mensaje debe denotar el propósito del objeto invocante en la interacción con el objeto invocado.

3.6.2.1 Diagrama de colaboración para el caso de uso “Modelar Interfaz Grafica”

Nota: (.....) La línea punteada indica que para que este caso de uso pueda darse, el usuario tiene que estar asociado a un proyecto. La configuración del archivo de proyecto se explica en el diagrama de colaboración relacionado a dicho caso de uso.

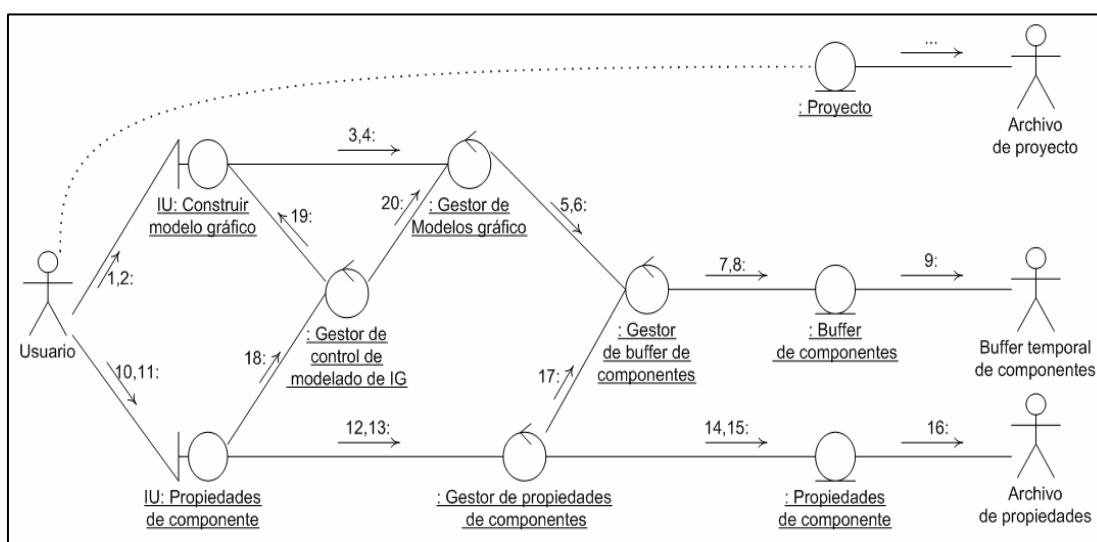


Figura 3.6: Diagrama de colaboración para el caso de uso “Modelar Interfaz Grafica”

Fuente: Elaboración propia

Leyenda

Tabla 3.4: Mensajes del Diagrama de colaboración para el caso de uso “Modelar Interfaz Grafica”. (1/2)

1.- Solicitud para insertar nuevo componente.
2.- Tamaño y ubicación de componente insertado.
3.- Solicitud de actualización de data de modelo grafico.
4.- Envío de data de nuevo componente insertado.
5.- Solicitud de almacenamiento de data nuevo componente.

Tabla 3.4: Mensajes del Diagrama de colaboración para el caso de uso “Modelar Interfaz Grafica”. (2/2)

6.- Envío de data de nuevo componente insertado.
7.- Solicitud de actualización de datos de buffer temporal.
8.- Nueva data de buffer temporal.
9.- Data actualizada de buffer temporal.
10.- Solicitud edición de componente seleccionado.
11.- Data para actualizar propiedades de componente.
12.- Envío de data de tipo de componente seleccionado.
13.- Solicitud de propiedades disponibles para componente.
14.- Solicitud de lectura propiedades disponibles para componente.
15.- Envío de data de tipo de componente seleccionado.
16.- Envío de data de ubicación de fichero de propiedades.
17.- Solicitud actualización de propiedades componente en edición.
18.- Solicitud de actualización de propiedades de componente grafico.
19.- Solicitud de actualización de vista de componente.
20.- Envío de data actualización componente grafico.

Fuente: Elaboracion propia

3.6.2.2 Diagrama de colaboración para el caso de uso “Generar código”

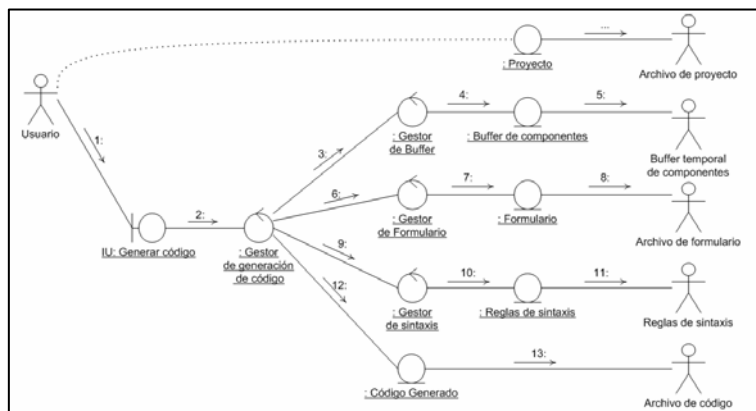


Figura 3.7: Diagrama de colaboración para el caso de uso “Generar Código”

Fuente: Elaboracion propia

Leyenda

Tabla 3.5: Mensajes del Diagrama de colaboración para el caso de uso “Generar Código”.

1.- Solicitud de generación de código.
2.- Envío de data de IU para generación de código.
3.- Solicitud de data de componentes en IU.
4.- Solicitud de data de componentes en IU.
5.-Solicitud de data de componentes en IU.
6.- Solicitud de almacenar data componentes.
7.- Envío de data de componentes.
8.- Solicitud de almacenar data de componentes.
9.- Solicitud de reglas de sintaxis.
10.- Solicitud de reglas de sintaxis.
11.- Consulta de reglas de sintaxis.
12.- Envío de código generado.
13.- Solicitud de almacenamiento de código generado.

Fuente: Elaboracion propia

3.6.2.3 Diagrama de colaboración para el caso de uso “Configurar Archivo”

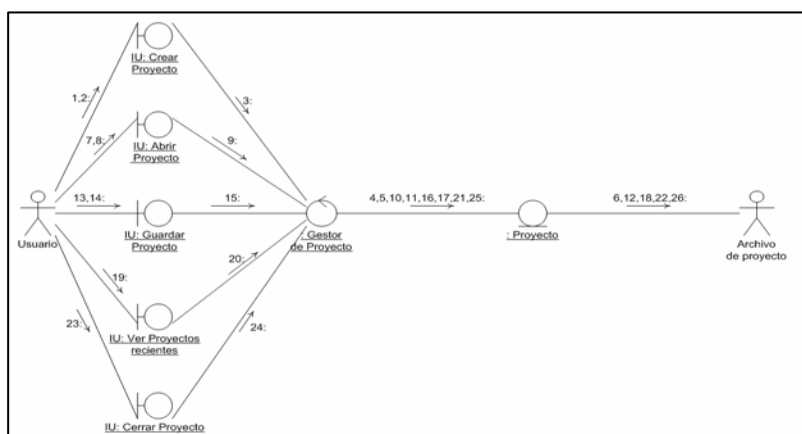


Figura 3.8: Diagrama de colaboración para el caso de uso “Configurar Archivo”

Fuente: Elaboracion propia

Leyenda

Tabla 3.6: Mensajes del Diagrama de colaboración para el caso de uso “Configurar Archivo”.

1.- Solicitud de creación de proyecto.
2.- Data suministrada para proyecto nuevo.
3.- Envío de data para nuevo proyecto.
4.- Solicitud de verificación proyecto existente.
5.- Envío de data para almacenar proyecto
6.- Solicitud para almacenar data proyecto.
7.- Solicitud para abrir proyecto.
8.- Ubicación y nombre de proyecto.
9.- Envío de data para abrir proyecto.
10.- Solicitud de verificación proyecto existente.
11.- Envío de data para abrir proyecto.
12.- Solicitud para abrir proyecto.
13.- Solicitud de guardar proyecto.
14.- Data suministrada para guardar proyecto.
15.- Envío de data para almacenar en proyecto.
16.- Verificación de ruta existente o nueva ruta.
17.- Envío de data para almacenar en proyecto.
18.- Solicitud para guardar cambios en proyecto.
19.- Solicitud para abrir proyecto reciente.
20.- Envío de nombre y ubicación de proyecto reciente.
21.- Envío de data para abrir proyecto.
22.- Solicitud para abrir proyecto.
23.- Solicitud para cerrar proyecto.
24.- Envío de nombre de proyecto a cerrar.
25.- Solicitud para cerrar proyecto.
26.- Solicitud para cerrar proyecto.

Fuente: Elaboracion propia

3.6.3 Diagrama de Paquetes de Análisis

Un diagrama de paquetes muestra como un sistema esta dividido en agrupaciones lógicas, mostrando las dependencias entre estas agrupaciones. Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema.

Los paquetes están normalmente organizados para maximizar la coherencia interna dentro de cada paquete y minimizar el acoplamiento externo entre los paquetes; dicho esto se puede deducir que los paquetes son buenos elementos de gestión. Cada paquete puede asignarse a un individuo o a un equipo, y las dependencias entre ellos pueden indicar el orden del desarrollo requerido.

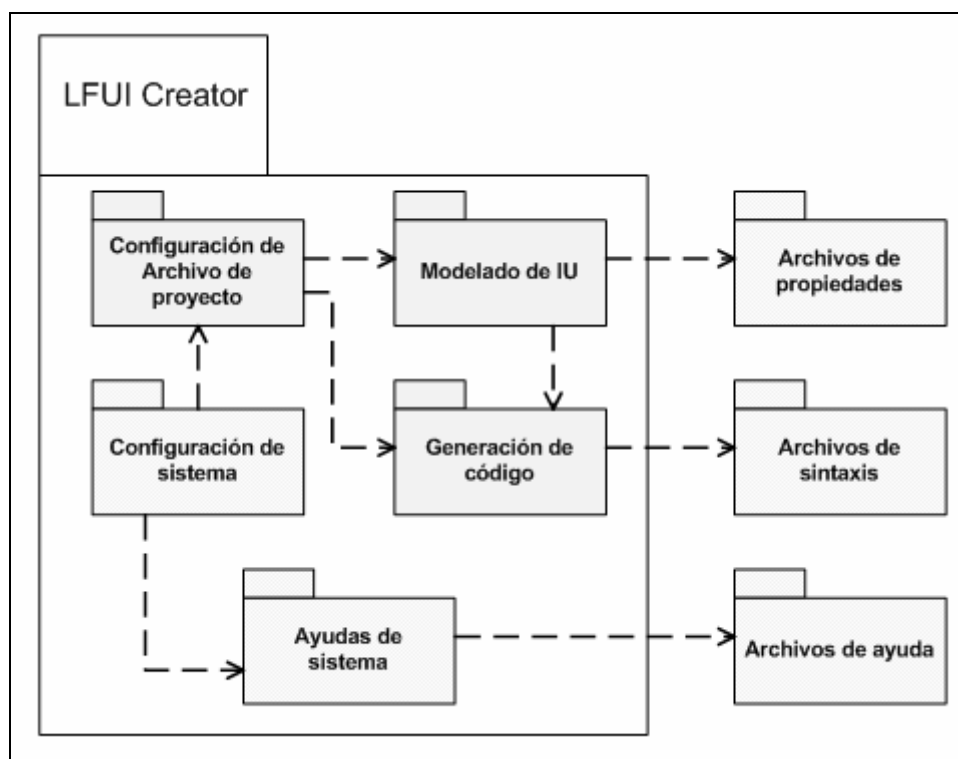


Figura 3.9: Diagrama de paquetes de análisis del sistema.

Fuente: Elaboración propia

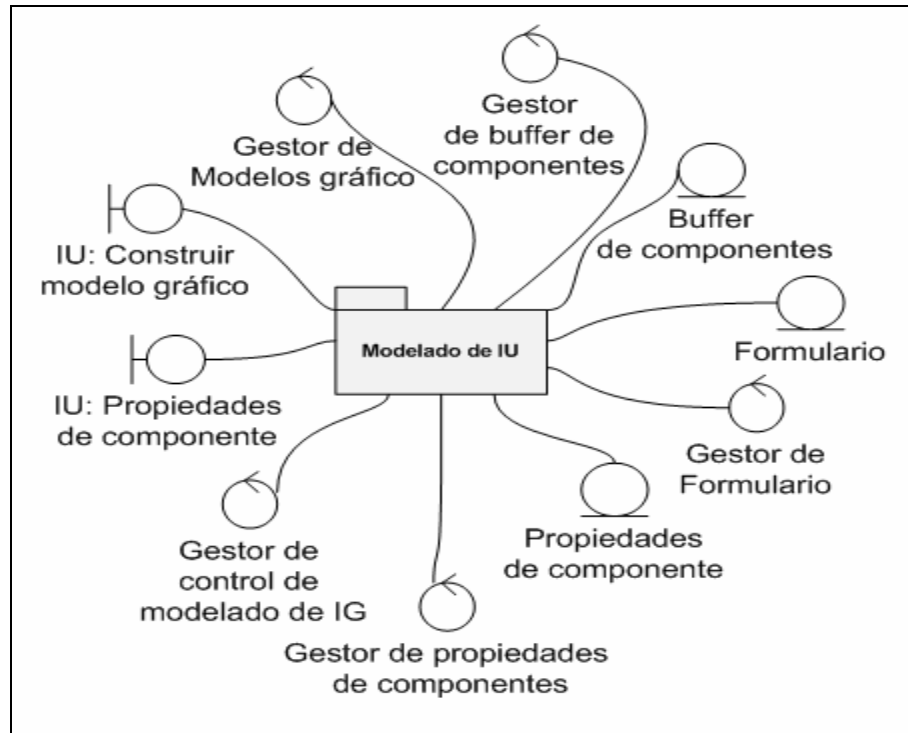


Figura 3.10: Paquetes de análisis de Modelado de UI.

Fuente: Elaboracion propia

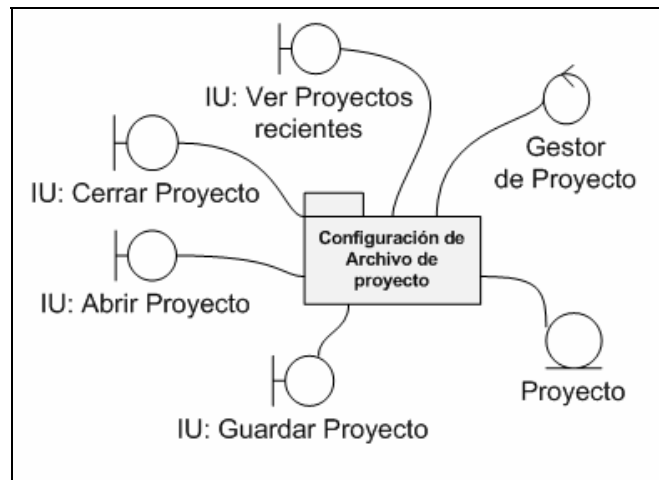


Figura 3.11: Paquetes de análisis Configuración de archivo de proyecto.

Fuente: Elaboracion propia

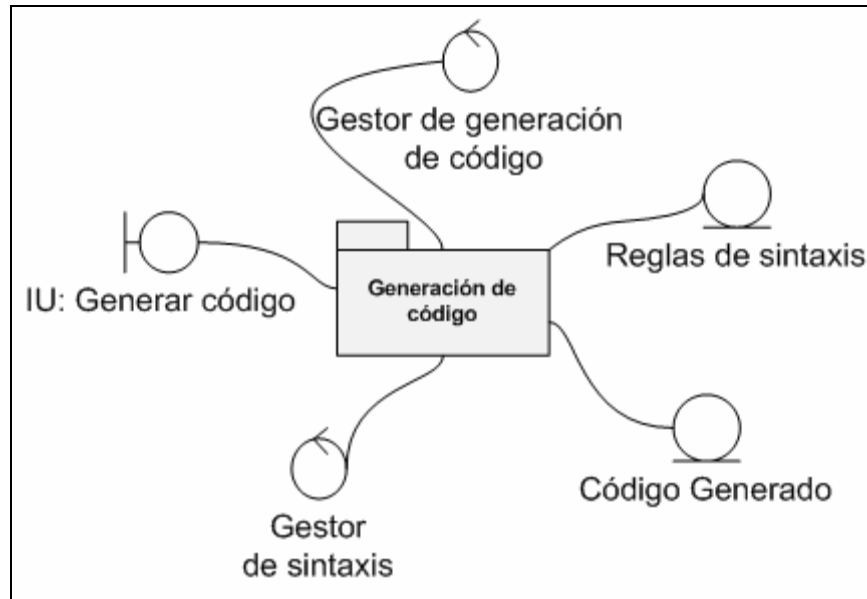


Figura 3.11: Paquetes de análisis Configuración Generación de código.

Fuente: Elaboracion propia

3.7 CONCLUSIÓN DE LA FASE DE INICIO

En esta fase se analizó el proceso actual para capturar los objetos más importantes dentro del contexto del sistema, realizando un análisis de las necesidades de los usuarios, una lista de requisitos funcionales indispensables y un modelo de dominio. En el glosario de términos se definieron las palabras o expresiones relacionadas al sistema, se estableció una lista de riesgos críticos del proyecto, los cuales se mitigaron a lo largo de esta fase y se continuaron disminuyendo en la fase de elaboración.

Además se construyó el modelo de casos de uso para la captura de los requisitos funcionales, con la identificación de actores, los casos de uso y las relaciones que existen entre estos.

Después de evaluar los objetivos de la fase de inicio, ámbito del sistema, riesgos críticos y arquitectura candidata, se logró determinar la viabilidad del proyecto y por lo tanto se continúa con el desarrollo del software.

CAPÍTULO IV

FASE DE ELABORACIÓN

La fase de elaboración tiene por objeto construir el núcleo central de la arquitectura, resolver los elementos de alto riesgo, definir los requerimientos y estimar los recursos necesarios. Dicha fase indica la culminación de una etapa en la cual se concretó la fase de inicio, obteniendo una gran parte del análisis del diseño y teniendo así una base importante del proyecto, logrando ahora profundizar en los procesos que dan mayor complejidad al sistema.

En esta fase se van a transformar y refinar los modelos de la fase de inicio en otra serie de modelos que irán perfilando una solución más cercana al mundo real, se construye la línea base de la arquitectura incluyendo técnicas de diseño, más bien globales que de detalle. Se identifican los procesos, capas de software, paquetes, subsistemas, estableciendo sus responsabilidades y se efectúa la clarificación de las interfaces internas (entre componentes) y externas (con los actores), refinándolas e incluyendo parámetros y valores de retorno.

4.1 REQUISITOS

En este flujo de trabajo no se identificaron nuevos actores, casos de uso, ni requisitos que se pudiesen adicionar a los antes establecidos, ya que hubo un total entendimiento del contexto en la fase de inicio, que permitió establecer directamente los lineamientos que se van a utilizar durante el desarrollo del proyectos.

En vista de lo anterior expuesto, se hace conveniente hacer una referencia a la interfaz de usuario mostrando las pantallas más importantes que formarán parte del producto final.

4.1.1 Interfaz de inicio LFUI Creator

Esta es la interfaz que verán los usuarios luego de ejecutar la aplicación. Proporciona un tiempo adicional de carga para establecer la configuración principal del sistema y garantizar el correcto despliegue de componentes de la Interfaz principal.



Figura 4.1: Interfaz de inicio del sistema

Fuente: Elaboracion propia

4.1.2 Interfaz principal (Pantalla de Inicio)

La pantalla de inicio del sistema conforma el estado que toma la interfaz principal si no existe ningún proyecto activo. Esta muestra un mensaje de bienvenida y refleja los componentes de edición deshabilitados por defecto.



Figura 4.2: Interfaz principal (Pantalla de Inicio)

Fuente: Elaboración propia

4.1.3 Interfaz principal (Modo edición de UI)

El modo de edición es el estado que adapta la interfaz principal del sistema al estar un proyecto activo en plena edición de una GUI. En la Figura 4.3 se puede apreciar todos los componentes habilitados y el Formulario de GUI que se está creando.

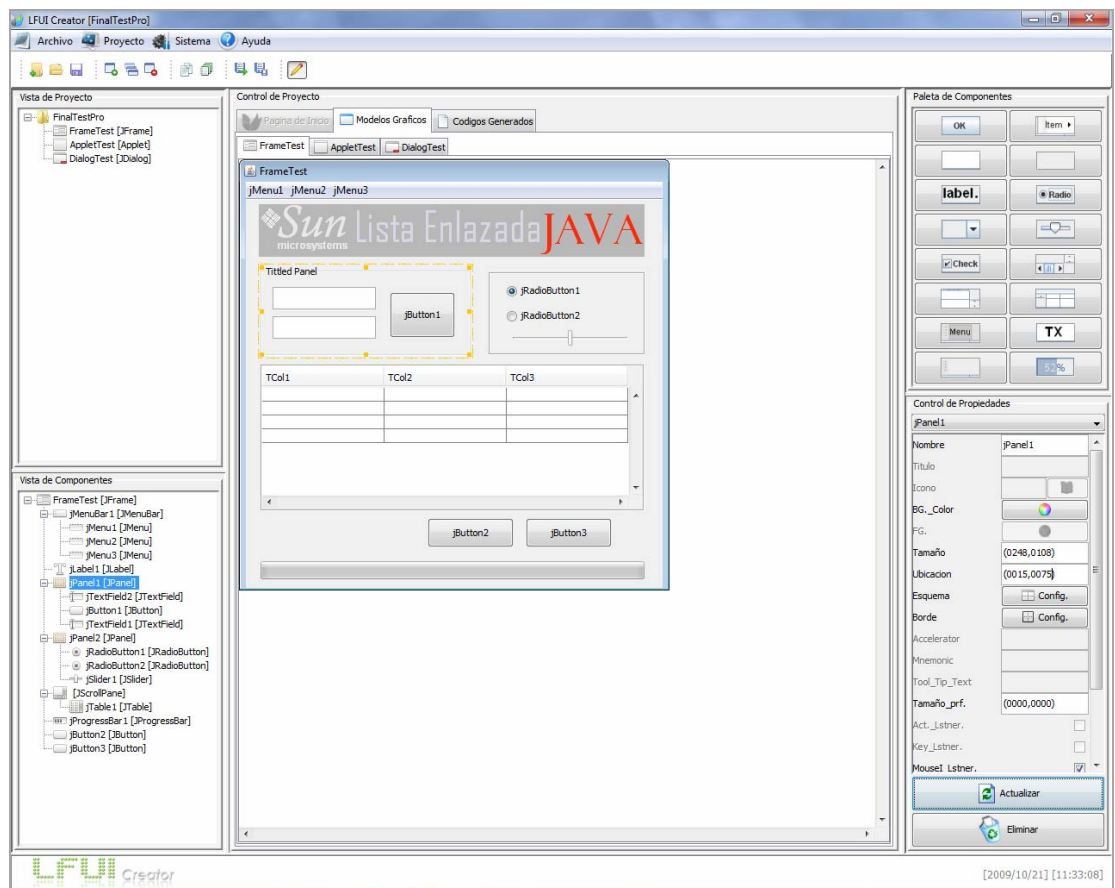


Figura 4.3: Interfaz principal (Modo edición de UI)

Fuente: Elaboración propia

4.1.4 Interfaz principal (Modo generacion de codigo)

La figura 4.4 muestra la interfaz principal desplegando el codigo fuente generado a partir de la UI editada por el usuario. El codigo es mostrado de forma sencilla y la interfaz proporciona algunas herramientas para trasladar o importar el texto.

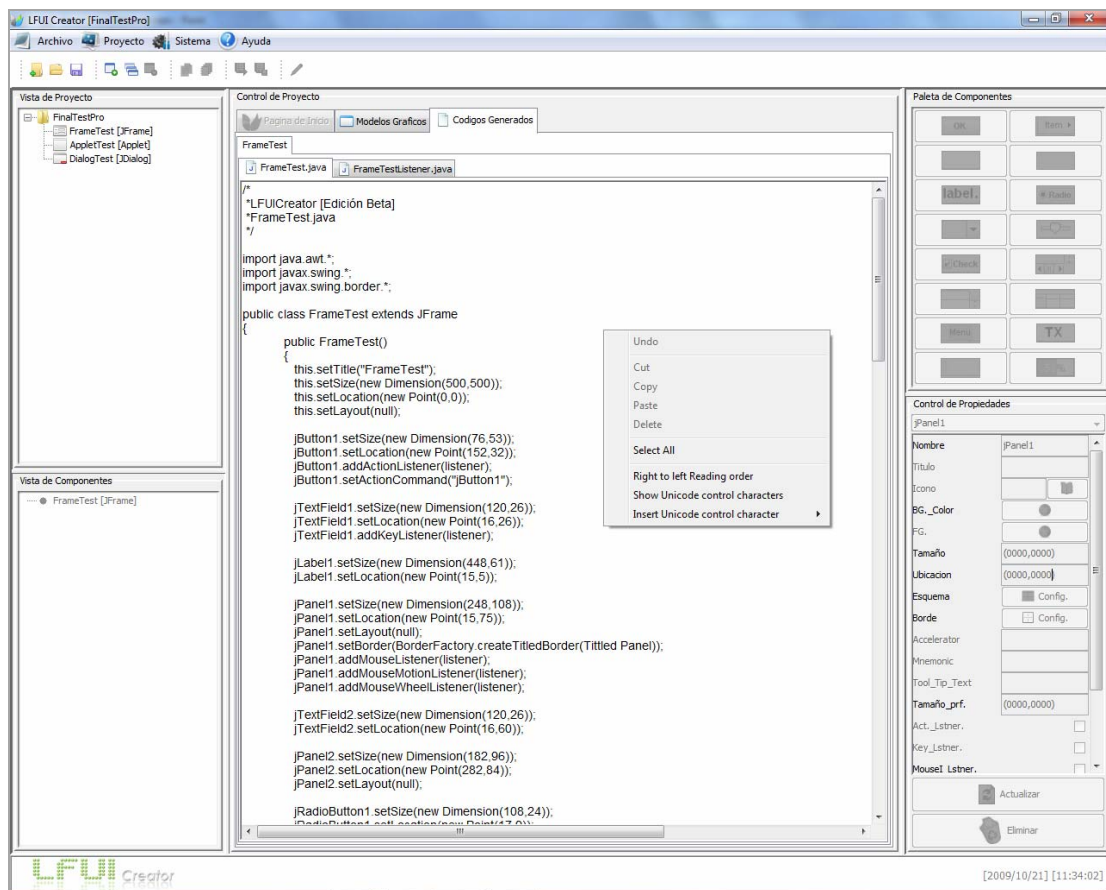


Figura 4.4: Interfaz principal (Modo generacion de codigo)

Fuente: Elaboracion propia

4.1.5 Dialogo de mensaje de Error

En la Figura 4.5 podemos observar el mensaje que el sistema mostrara cuando se detecte un valor incorrecto en algun campo de validacion importante. En este caso el campo mas reelevante es el texto insertado en la modificacion de nombre de algun objeto en edicion o de alguna UI agregada al proyecto. Este mensaje se desplegara para cualquier valor de texto que pueda generar un error de sintaxis en el codigo generado, lo que traeria como consecuencia un error de compilacion.

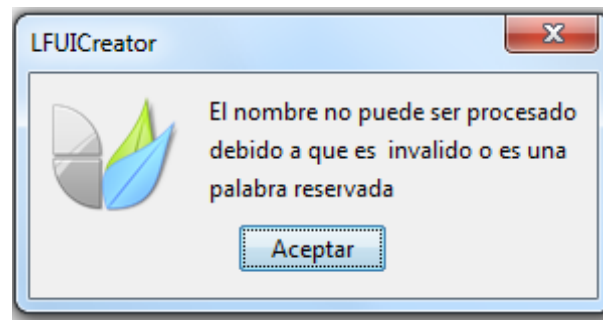


Figura 4.5: Dialogo de mensaje de Error

Fuente: Elaboracion propia

4.2 DISEÑO

En la fase de elaboraci3n el dise1o tiene como objetivo la obtenci3n de la vista de la arquitectura del modelo de dise1o y la realizaci3n f1sica de los casos de uso. Se modelara el sistema para que soporte todos los requisitos, incluyendo los requisitos no funcionales y otras restricciones, que se le suponen.

4.2.1.1 Diagrama de clases de Diseño para el caso de uso Generar Código

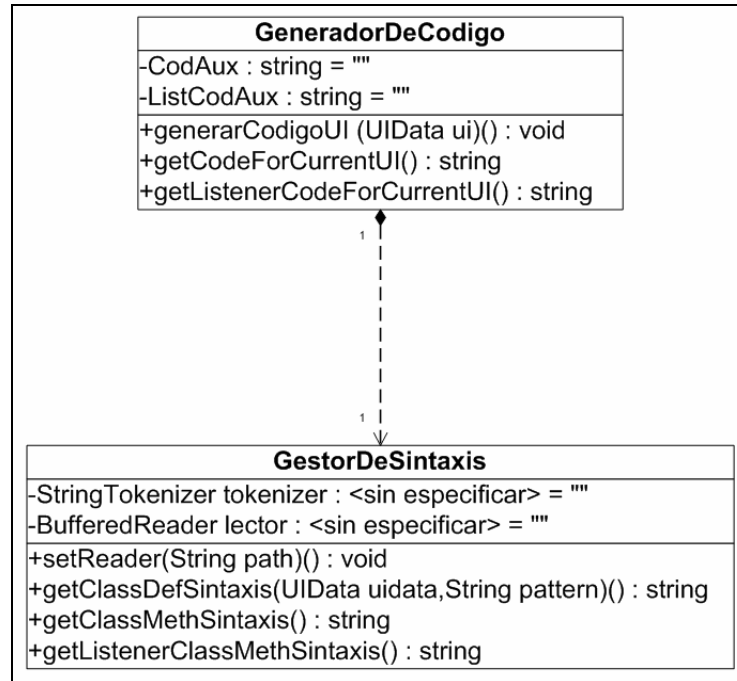


Figura 4.7: Diagrama de Clase de Diseño para el caso de uso Generar Código

Fuente: Elaboracion propia

La figura 4.7 muestra las clases involucradas en el caso de uso generar código. Se puede apreciar la clase `GeneradordeCodigo` la cual se encarga de ensamblar las cadenas de código obtenidas del `GestorDeSintaxis` que es la clase encargada de hacer la lectura de los archivos de sintaxis.

4.2.2. Diagrama de Secuencia

Los Diagramas de Secuencia son conocidos como Diagramas de Interacción dado que muestran la interacción que se establece en un conjunto de objetos y sus relaciones, además de los mensajes que estos se envían. Es decir, tratan la vista dinámica de un sistema. Estos diagramas enfatizan como se realiza un caso de uso en términos de las clases del diseño, identifica las instancias de los actores, las interacciones entre los objetos del diseño y los mensajes que se envían y reciben estos objetos. Aquí lo importante es encontrar secuencias de interacciones ordenadas en el tiempo. A continuación se muestran los Diagramas de Secuencia de los Casos de Uso más importantes del sistema LFUI Creator.

4.2.2.1 Diagrama de secuencia para el caso de uso Generar Código.

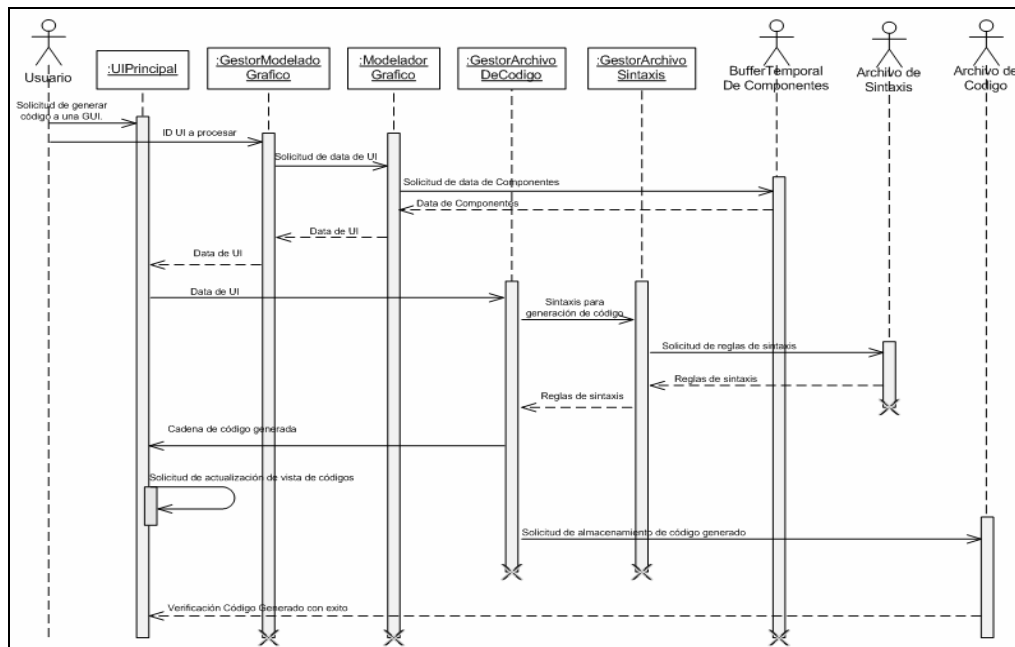


Figura 4.8: Diagrama de Secuencia para el caso de uso Generar Código

Fuente: Elaboración propia

4.2.3 Diagrama de Capas

El uso de la arquitectura de capas es aplicable a muchos tipos de sistemas. Este patrón define como organizar el modelo de diseño en capas, lo cual quiere decir que los componentes de una capa solo pueden hacer referencia a componentes en capas inmediatamente inferiores. Este patrón es importante porque simplifica la comprensión y la organización del desarrollo de sistemas complejos, reduciendo las dependencias de forma que las capas mas bajas no son conscientes de ningún detalle o interfaz de las superiores. Un sistema con una arquitectura en capas pone a los subsistemas de aplicación individuales en lo más alto. Estos se construyen a partir de subsistemas en las capas mas bajas, como son lo marcos de trabajo y las bibliotecas de clases.

La capa general de aplicación contiene los subsistemas que no son específicos de una sola aplicación, sino que pueden ser reutilizados por muchas aplicaciones diferentes dentro del mismo dominio o negocio. La arquitectura de las dos capas inferiores puede establecerse sin considerar los casos de uso de que no son dependientes del negocio. La arquitectura de las dos capas superiores se crea a partir de los casos de uso significativos para la arquitectura (estas capas son dependientes del negocio).

La Figura 4.10 nos muestra la arquitectura del sistema LFUI Creator utilizando un diagrama de capas.

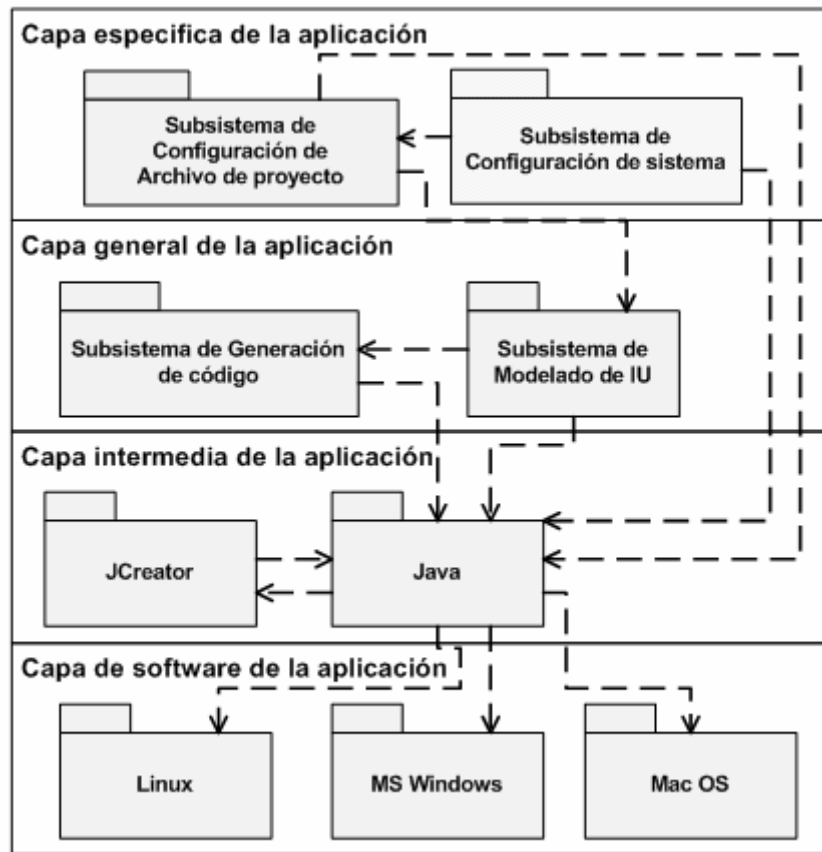


Figura 4.10: Diagrama de Capas LFUI Creator.

Fuente: Elaboración propia

4.3 IMPLEMENTACIÓN

En la implementación se comenzará con el uso de los resultados del diseño y se implementará el sistema en términos de componentes. El propósito principal de este flujo de trabajo es desarrollar la arquitectura y el sistema como un todo. Aquí se realizará el diagrama de componentes parcial de la aplicación.

4.3.1 Diagrama de Componentes:

Para la fase de elaboración se ha construido un adelanto de los componentes que conforman el caso de uso Generar Código. En la figura 4.11 observamos los componentes relacionados.

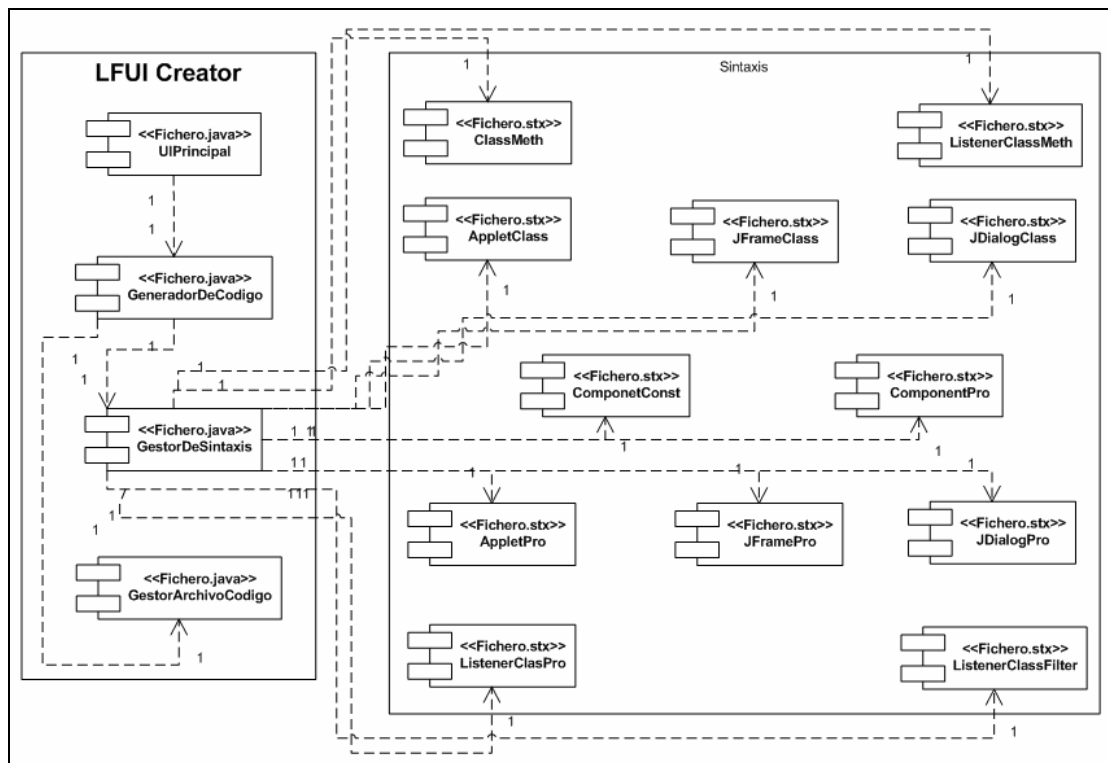


Figura 4.11: Diagrama de componentes parcial para el caso de uso Generar Código.

Fuente: Elaboracion propia

4.3.2 Implementación de los componentes asociados al caso de uso Generar Código.

```
package GeneracionCodigo;

import java.io.*;
import java.util.StringTokenizer;
import java.util.ArrayList;
```

```

import ModeladoUI.UIData;
import ModeladoUI.Componente;
import ModeladoUI.PropConstants;

public class GeneradorDeCodigo
{
    public GeneradorDeCodigo()
    {
        sintaxis = new GestorDeSintaxis();
    }

    public void generarCodigoUI(UIData ui)
    {
        String code = "";
        String code2 = "";

        String classDef = sintaxis.getClassDefSintaxis(ui, "#NNNNNNNNNN");
        String[] classPro =
sintaxis.getClassProSintaxis(ui, "XXXXXXXXXX1", "XXXXXXXXXX2", "XXXXXXXXXX3");
        String[] compInits =
sintaxis.getCompInitSintaxis(ui, "XXXXXXXXXX1", "XXXXXXXXXX2", "NNNNNNNNNN");
        ArrayList[] compPros =
sintaxis.getCompProSintaxis(ui, "XXXXXXXXXX1", "XXXXXXXXXX2", "XXXXXXXXXX3", "NNNNNN
NNNNN");
        Object[] compAdds = sintaxis.getCompAddSintaxis(ui);

        String classpro = "";
        String compinits = "";
        String comppros = "";
        String compadds = "";

        boolean[] pos = ui.getPOSIBLE();
        for (int i = 0; i<classPro.length; i++)
        {
            if(pos[i])
            {
                classpro += classPro[i]+"\\n          ";
            }
        }
        code = classDef;

        for (int i = 0; i<compInits.length; i++)
        {
            compinits += "private "+compInits[i]+"\\n          ";
        }

        for(int i = 0; i<compAdds.length; i++)
        {
            if(compAdds[i]!=null)
            {
                if(i<compAdds.length-1)
                    compadds += compAdds[i].toString()+"\\n          ";
                else
                    compadds += compAdds[i].toString();
            }
        }

        for (int i = 0; i<compPros.length; i++)

```

```

{
    ArrayList<String> st = compPros[i];

    for (int j = 0; j<st.size(); j++)
    {
        comppros += st.get(j).toString()+"\n                ";
    }

    if(st.size()!=0 & i<compPros.length-1)
        comppros += "\n                ";
}

code = code.replaceAll("#UUUUUUUUUU",classpro);

if(compPros.length>0)
code = code.replaceAll("#CCCCCCCCCC", "\n                "+comppros);
else
code = code.replaceAll("#CCCCCCCCCC", "");

if(compAdds.length>0)
code = code.replaceAll("#AAAAAAAAAA", "\n                "+compadds);
else
code = code.replaceAll("#AAAAAAAAAA", "");

if(compInits.length>0)
code = code.replaceAll("#TTTTTTTTTT",compinits);
else
code = code.replaceAll("#TTTTTTTTTT", "");

if(sintaxis.isUiListened())
{
    code = code.replaceAll("#EEEEEEEEEE", "\n
+sintaxis.getListenerInitDefSintaxis(ui.getName());

    code2 =
sintaxis.getListenerClassDefSintaxis(ui.getName(), "XXXXXXXXX1", "XXXXXXXXX2",
"XXXXXXXXX3", "XXXXXXXXX4", "NNNNNNNNNN");
    Object[] lclassMeth = sintaxis.getListenerClassMethSintaxis();
    String lclassmeth = "";

    for(int i = 0; i<lclassMeth.length; i++)
    {
        if(i<lclassMeth.length-1)
            lclassmeth += lclassMeth[i].toString()+"\n                ";
        else
            lclassmeth += lclassMeth[i].toString();
    }

    code2 = code2.replaceAll("MXXXXXXXXX",lclassmeth);

    Object[] classMeth = sintaxis.getClassMethSintaxis();
    String classmeth = "";

    for(int i = 0; i<classMeth.length; i++)
    {
        if(i<classMeth.length-1)
            classmeth += classMeth[i].toString()+"\n                ";
        else

```

```

        classmeth += classMeth[i].toString();
    }

    code = code.replaceAll("#MMMMMMMMMM",classmeth+"\n        ");
}

code = code.replaceAll("#EEEEEEEEEEE", "");
code = code.replaceAll("#MMMMMMMMMM", "");

this.codaux = code;
this.listcodaux = code2;
}

public String getCodeForCurrentUI()
{
    return this.codaux;
}

public String getListenerCodeForCurrentUI()
{
    return this.listcodaux;
}

public boolean isUiListened()
{
    return sintaxis.isUiListened();
}

private String codaux;
private String listcodaux;
private boolean go = false;
private GestorDeSintaxis sintaxis;
}

```

4.4 CONCLUSIÓN DE LA FASE DE ELABORACIÓN

Al comienzo de esta fase, se recibió de la fase de inicio un modelo de casos de uso completo y una descripción de la arquitectura candidata. Durante el desarrollo de los flujos de trabajo se obtuvieron las diferentes vistas de la arquitectura del sistema, estableciendo la prioridad de los casos de uso y la línea base de la arquitectura que soporta los casos de uso críticos del sistema.

Tambien se ha abarcado todo el dominio del proyecto, profundizando en los puntos críticos o riesgos importantes, se han actualizado todos los productos de la fase de inicio y los principales elementos de riesgo han sido abordados y resueltos. Con el cumplimiento de la planificación de esta fase, se han obtenido todos los elementos necesarios para completar la implementación del sistema en la siguiente fase.

CAPÍTULO V

FASE DE CONSTRUCCIÓN

5.1 INTRODUCCION

La finalidad principal de la fase de construcción es alcanzar la capacidad operacional del producto. Durante esta fase todos los componentes, características y requisitos identificados y detallados en las fases anteriores deben ser adecuados a las posibilidades de la herramienta de desarrollo que se va a usar, eliminando riesgos, para luego ser implementados, integrados y probados en su totalidad, obteniendo una versión aceptable del software.

En esta fase se hace hincapié en las iteraciones de implementación y prueba del flujo de trabajo normal, debido a que su meta es lograr el desarrollo del sistema con calidad de producción, mediante la implementación de toda la funcionalidad y realización de las pruebas.

5.1.1 Escogencia del Lenguaje de Programación

Para el desarrollo de LFUI Creator, se debe contar con un lenguaje que soporte la programación orientada a objetos, ya que esta promueve una mejor comprensión de los requisitos, diseños más limpios y sistemas mas fáciles de mantener. La herramienta seleccionada tambien debe poseer extensos recursos graficos y librerias optimizadas para un eficiente manejo de eventos. Es por esto que se escogio como lenguaje de programacion Java (Sun Microsystems), debido a que es un lenguaje

simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico.

Ademas cuenta con unas poderosas librerias graficas que hacen una gestion eficiente de componentes visuales y permiten lograr Intefaces de usuario de alto nivel, como tambien un robusto manejo de eventos.

Como herramienta de edicion de codigo Java se utilizo JCreator v_4.5, desarrollado por Xinox Software. Este IDE proporciona herramientas para compilar y ejecutar aplicaciones Java, ademas proporciona facilidades para insertar templates y auto completar codigo fuente. La figura 5.1 muestra la interfaz del JCreator en la edicion de codigo fuente.

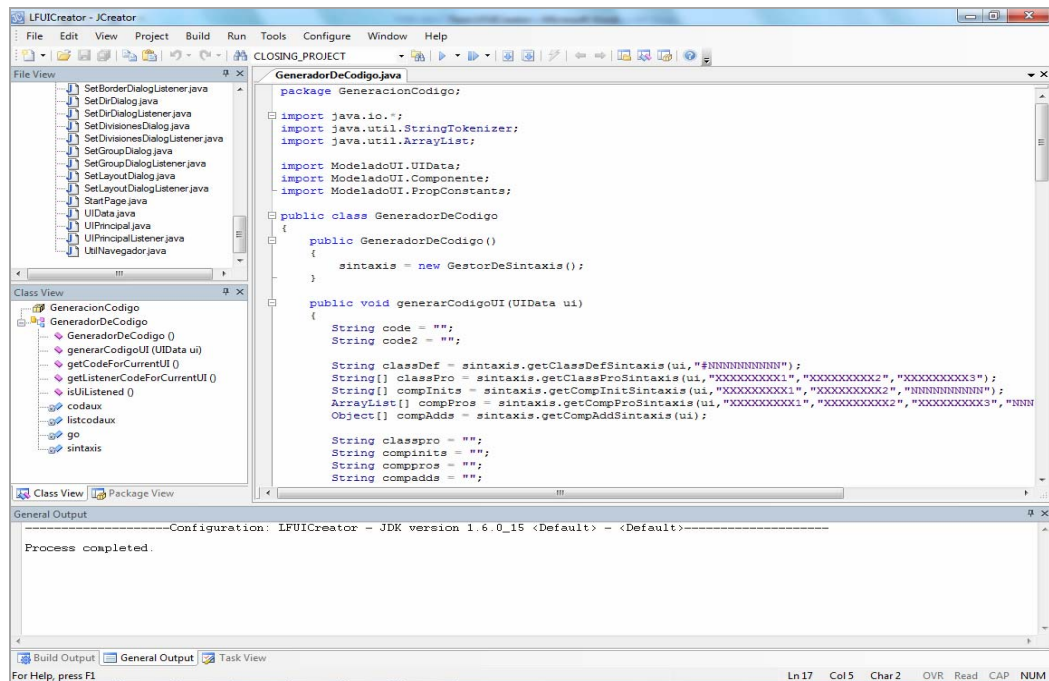


Figura 5.1: Interfaz del entorno de programación Java (JCreator)

Fuente: Elaboracion propia

5.2 IMPLEMENTACION

La implementación trata al sistema en términos de desarrollo de componentes y codificación del software, integración de módulos y la explicación acerca de las funcionalidades del sistema y su correcto uso, delatando así toda la estructura en forma de código abierto e identificación de las actividades que este puede realizar.

5.2.1 Análisis de formato para archivos de propiedades “PRV” y sintaxis “STX”

5.2.1.1 Formato PRV

Los archivos de extension PRV contienen las propiedades que el sistema permite administrar a cada componente. Todo componente se maneja con un maximo de 20 propiedades de distintos tipo y nombres, el fichero PRV posee el indice de la propiedad, la bandera de activacion y el identificador asociado.

INDEX ACT_FLAG IDENT

Cuando un componente es seleccionado para su edicion, el sistema carga el archivo PRV correspondiente y activa en la ventana de propiedades aquellos indices que posean un valor true en la bandera de activacion. En el archivo cada linea es un registro o propiedad y posee el formato antes visto, los campos estan separados entre si por espacios en blanco y el fin del archivo esta delimitado por la palabra “EOF”.

En la tabla 5.1 podemos observar el contenido de el archivo PRV correspondiente a el componente JButton

Tabla 5.1: Formato de Archivo PRV

INDEX	ACT_FLAG	IDENT
0	True	Nombre
1	True	Titulo
2	True	Icono
3	True	BG._Color
4	True	FG._Color
5	True	Tamaño
6	True	Ubicación
7	False	Esquema
8	True	Borde
9	False	Accelerator
10	True	Mnemonic
11	True	Tool_Tip_Text
12	False	Tamaño_prf.
13	True	Act._Lstner.
14	False	Key_Lstner.
15	False	MouseI_Lstner.
16	False	MouseW_Lstner.
17	False	Button_Group
18	False	Division
19	False	Padre
EOF		

Fuente: Elaboracion propia

5.2.1.2 Formato STX

Los archivos de extension STX contienen la sintaxis Java de algun determinado fragmento importante de codigo.

Hacen uso de cadenas o patrones de identificacion, los cuales son sustituidos en el modulo de Gestion de sintaxis por los valores o parametros que el usuario graficamente asigno a un componente determinado en relacion a una propiedad. Los archivos STX no tienen una estructura estandar debido a que el codigo fuente Java puede ser dividido en varios segmentos de estructura variable. Entonces cada archivo STX esta designado a contener un fragmento de codigo puntual en una estructura de clase Java. A continuacion se muestra el contenido de JFrameClass.STX:

```
/*
 *LFUICreator [Edición Beta]
 *#NNNNNNNNNN.java
 */

import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;

public class #NNNNNNNNNN extends JFrame
{
    public #NNNNNNNNNN()
    {
        #UUUUUUUUUU #CCCCCCCCC #AAAAAAAAAA
    }

    #MMMMMMMMMM #TTTTTTTTTT #EEEEEEEEEE
}
EOF
```

En la tabla 5.2 podemos observar el comportamiento de cada patron de identificacion.

Tabla 5.2: Patrones de Archivo STX

PATRON	USO
#NNNNNNNNNN	Sera sustituido por el nombre de la clase en gestion.
#UUUUUUUUUU	Sera sustituido por las propiedades de la interfaz en gestion.
#CCCCCCCCCC	Sera sustituido por las propiedades de los componentes agregados a la interfaz en gestion.
#AAAAAAAAAAA	En este instante los objetos son adheridos a la interfaz. Es decir, ese segmento contendra todas las llamadas al metodo add();
#MMMMMMMMMM	Sera sustituido por todos los metodos designados para la clase de interfaz en gestion.
#TTTTTTTTTTT	Sera sustituido por la declaracion e inicializacion de todos los componentes contenidos en la interfaz.
#EEEEEEEEEEE	Sera sustituido por el objeto Listener encargado de administrar los eventos de la interfaz en general.

Fuente: Elaboracion propia

5.2.2 Diagrama de Componentes Total

En esta fase se desarrolla el diagrama de componentes con la totalidad de los componentes del sistema, indicando las clases que necesitan cada uno. En la figura 5.2 podemos observar este diagrama.

5.3 PRUEBAS

Las pruebas son el instrumento que permite validar y verificar el software, es decir, son los procesos que determinan si el software satisface los requisitos y funciona de la manera establecida. Estas pruebas tienen como objeto verificar la interacción entre los objetos, al igual que la integración apropiada de componentes, verificar que se satisfagan los requerimientos, identificar los defectos y corregirlos antes de la instalación.

5.3.1 Pruebas por Unidad

Las pruebas por unidad se aplicaron mediante la aplicación de la prueba de la caja negra sobre los diversos componentes del sistema. Para realizar este tipo de pruebas, se identifican un conjunto de valores que pueden ser introducidos por un actor, y se expresan como clases de equivalencia para poder abarcar la totalidad de las ocurrencias de un evento de inserción de datos.

A continuación en la tabla 5.3 se representan las clases de equivalencia del componente AddUIDialog, el mismo se encarga de recibir los datos de una nueva GUI agregada al proyecto en gestión.

Tabla 5.3: Clases de equivalencia del componente AddUIDialog (1/2)

NRO.	CAMPO	CLASE EQUIVALENCIA	VALIDO	INVALIDO
1	Nombre	Dato Nulo		*
2	Nombre	Longitud de cadena < 40	*	
3	Nombre	Primer carácter numerico		*
4	Nombre	Primer carácter letra	*	

Tabla 5.3: Clases de equivalencia del componente AddUIDialog (2/2)

5	Nombre	Campo = palabra reservada		*
6	Nombre	Campo = nombre de clase		*
7	Nombre	Uso de espacios en blanco	*	
8	Nombre	Uso de caracteres especiales		*

Fuente: Elaboración propia

A continuación en la tabla 5.4 se representan las clases de equivalencia del componente NuevoProyectoDialog, quien se encarga de recibir los datos asociados a la creación de un proyecto.

Tabla 5.4: Clases de equivalencia del componente NuevoProyectoDialog

NRO.	CAMPO	CLASE EQUIVALENCIA	VALIDO	INVALIDO
1	Nombre	Dato Nulo		*
2	Nombre	Longitud de cadena < 40	*	
3	Nombre	Uso de caracteres especiales		*
4	Nombre	Uso de caracteres alfa-numericos	*	
5	Nombre	Uso de “ “	*	
6	Ruta	Dato Nulo		*
7	Ruta	Uso de caracteres especiales		*
8	Ruta	Uso de caracteres alfa-numericos	*	
9	Ruta	Uso de “\”	*	
10	Ruta	Uso de “/”	*	
11	Ruta	Ruta no encontrada en el sistema		*

Fuente: Elaboración propia

En la tabla 5.5 y 5.6 se presentan algunos casos de prueba de caja negra, donde se incluirán datos que serán cotejados por las clases de equivalencia referenciadas

anteriormente. Si se cumplen todas las clases involucra con dicho dato, entonces la salida será validada.

Tabla 5.5: Casos de prueba de caja negra para el componente AddUIDialog

CAMPO	CASO DE PRUEBA	SALIDA	CLASES CUBIERTAS
Nombre	""	Invalido	1
Nombre	MiClase/UI	Valido	8
Nombre	2daClase	Invalido	3
Nombre	this	Invalido	5
Nombre	int	Invalido	5
Nombre	JFrame	Invalido	6
Nombre	Integer	Invalido	6
Nombre	Mi Clase	Valido	2,4,7
Nombre	Mi2daClase	Valido	2,4

Fuente: Elaboración propia

Tabla 5.6: Casos de prueba de caja negra para el componente NuevoProyectoDialog

CAMPO	CASO DE PRUEBA	SALIDA	CLASES CUBIERTAS
Nombre	""	Invalido	1
Nombre	Nuevo Proyecto	Valido	2,4,5
Nombre	Proyecto 54	Valido	2,4,5
Nombre	Proyecto/UIs	Invalido	3
Ruta	Z:	Invalido	11
Ruta	C:/Mi*Proyecto	Invalido	7
Ruta	""	Invalido	6
Ruta	C:/Mis Documentos	Valido	8,10
Ruta	C:\Mis Documentos	Valido	8,9

Fuente: Elaboración propia

5.3.2 Pruebas de Integración

El objetivo general de las pruebas de integración es, detectar las fallas de interacción entre las distintas clases que componen al sistema. Debido a que cada clase probada por separado se inserta de manera progresiva dentro de la estructura, las pruebas de integración son realmente un mecanismo para comprobar el correcto ensamblaje del sistema completo. Al efectuar la integración de los módulos, se concentra el esfuerzo en la búsqueda de fallas que puedan provocar excepciones arrojadas por los métodos; el empleo de operaciones equivocada, e invocación inadecuada de los métodos.

Luego de verificar la calidad de los componentes, se procede a comprobar la eficiencia de un conjunto de componentes integrados por fase, estas se pueden observar resaltadas en la figura 5.3 en donde se despliega el diagrama de componentes por fase de integración del sistema.

Tabla 5.7: Leyenda de colores para fases del diagrama de componentes total

COLOR	FASE
	Fase 1
	Fase 2
	Fase 3
	Fase 4
	Fase 5
	Fase 6
	Fase 7
	Fase 8

Fuente: Elaboración propia

5.3.2.1 Integración de la Configuración de Proyecto

Este caso de prueba verifica la funcionalidad de la implementación de la fase 3.

Tabla 5.8: Caso de prueba para la fase de Configuración de Proyecto o Fase 3

CAMPO	DATOS
Nombre	Nuevo Proyecto
Ruta	C:\Libraries\Documents

Fuente: Elaboración propia

5.3.2.1.1 Resultados

- Se encontro necesario crear un proceso de validacion de la ruta para evitar sobrescribir directorios del sistema en el momento de generar el proyecto en la ruta dada y con el nombre dado.
- Luego de dicha correccion no se encontraron mas errores en esta fase.

5.3.2.1.2 Procedimiento de prueba

- Activar la interfaz principal.
- Acceder al menu de archivo y seleccionar la opcion “Nuevo Proyecto”.
- Se activa el componente NuevoProyectoDialog
- Llenar los campos “Nombre” y “Ruta” con los datos contemplados en la tabla 5.8.
- Presionar el boton Aceptar para capturar los datos.
- Se activa el componente GestorArchivoProyecto y hace la validacion de la ruta y el nombre para evitar sobreescritura.
- Se crea una nueva instancia de Proyecto.
- Se genera el directorio de proyecto. El cual contiene el archivo de proyecto y los directorios “Formularios”, “Codigos” e “Imagenes”. Estos para el almacenamiento de UIs, codigos generados e imagenes usadas en el proyecto, respectivamente.

A continuacion se presentan un conjunto de imágenes que muestran los resultados del procedimiento.

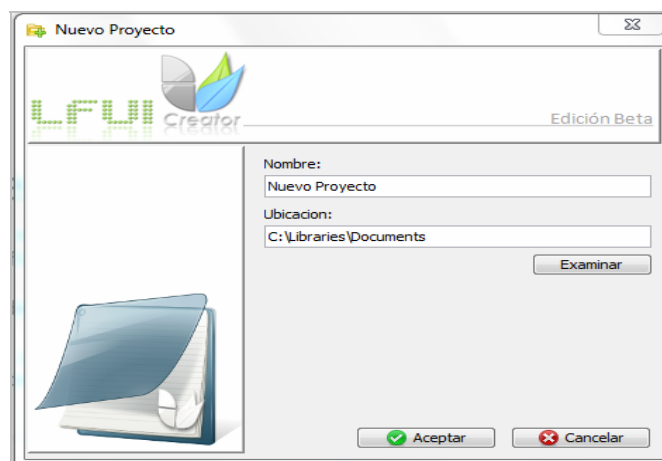


Figura 5.4: Componente NuevoProyectoDialog recibiendo la data de Nuevo Proyecto

Fuente: Elaboración propia

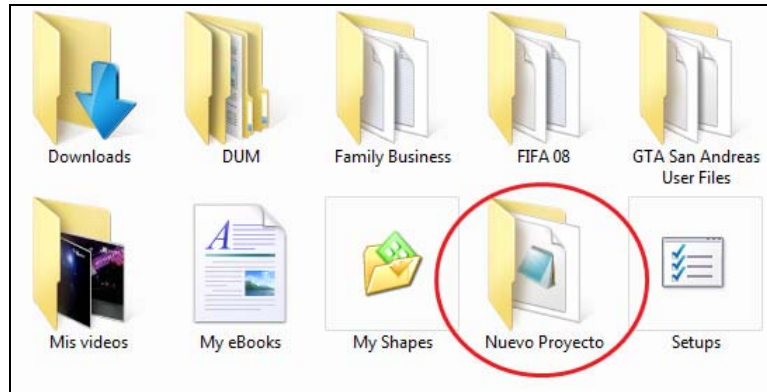


Figura 5.5: Directorio de Nuevo Proyecto generado.

Fuente: Elaboración propia

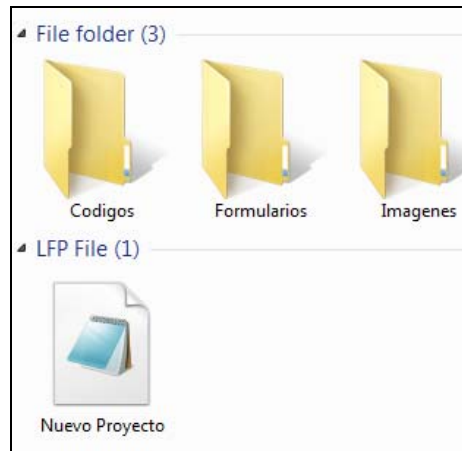


Figura 5.6: Contenido del directorio de Nuevo Proyecto generado.

Fuente: Elaboración propia

5.3.2.2 Código de componentes en la fase de Configuración de Proyecto

5.3.2.2.1 Proyecto.java

```
package ConfigProyecto;

public class Proyecto
{
    public Proyecto()
    {
        NUis = 0;
    }

    public String getNombre() {
        return (this.Nombre);
    }

    public void setNombre(String Nombre) {
        this.Nombre = Nombre;
    }

    public String getRuta() {
        return (this.Ruta);
    }

    public String getRutaF() {
        return (this.Ruta+"\\\\"+"Formularios");
    }

    public String getRutaC() {
        return (this.Ruta+"\\\\"+"Codigos");
    }

    public String getRutaI() {
        return (this.Ruta+"\\\\"+"Imágenes");
    }

    public void setRuta(String Ruta) {
        this.Ruta = Ruta;
    }

    public String getFecha() {
        return (this.Fecha);
    }

    public void setFecha(String Fecha) {
        this.Fecha = Fecha;
    }

    public int getNUis() {
        return (this.NUis);
    }
}
```

```

    public void setNUis(int NUis) {
        this.NUis = NUis;
    }

    public String[] getUis() {
        return (this.Uis);
    }

    public void setUis(String[] Uis) {
        this.Uis = Uis;
    }

    private String Nombre;
    private String Ruta;
    private String Fecha;
    private int NUis;
    private String[] Uis;
}

```

5.3.2.2.2 GestorArchivoProyecto.java

```

package ConfigProyecto;

import java.io.*;
import javax.swing.JOptionPane;

public class GestorArchivoProyecto
{
    public GestorArchivoProyecto()
    {
        proyecto = new Proyecto();
    }

    public void crearProyecto(String Nombre,String Ruta,String Fecha)
    throws IOException
    {
        proyecto.setNombre(Nombre);
        proyecto.setRuta(Ruta+"\\ "+Nombre);
        proyecto.setFecha(Fecha);

        ProycFile = new File(Ruta+"\\ "+Nombre);

        if(ProycFile.exists())
        {
            if(JOptionPane.showConfirmDialog(null,"El directorio
que intenta crear ya existe, ¿Desea reemplazarlo?","Directorio
existente",JOptionPane.OK_CANCEL_OPTION,JOptionPane.WARNING_MESSAGE) ==
JOptionPane.OK_OPTION)
                go = true;
            else

```

```

        go = false;
    }
    else
    go = true;

    if(go)
    {
        ProycFile.mkdir();

        FormsFile = new File(Ruta+"\\ "+Nombre+"\\Formularios");
        FormsFile.mkdir();

        CodesFile = new File(Ruta+"\\ "+Nombre+"\\Codigos");
        CodesFile.mkdir();

        ImagesFile = new File(Ruta+"\\ "+Nombre+"\\Imágenes");
        ImagesFile.mkdir();

        try
        {
            escritor = new ObjectOutputStream(new
FileOutputStream(Ruta+"\\ "+Nombre+"\\ "+Nombre+".lfp"));

            escritor.writeObject(Nombre);
            escritor.writeObject(Ruta+"\\ "+Nombre);
            escritor.writeObject(Fecha);
            escritor.writeInt(proyecto.getNUis());
            escritor.writeObject("EOF");

            escritor.close();
        }catch(Exception exc){}

    }

    public void abirProyecto(String totalPath) throws IOException
    {
        try
        {
            lector = new ObjectInputStream(new FileInputStream(totalPath));

            proyecto.setNombre(lector.readObject().toString());
            lector.readObject();
            proyecto.setRuta(totalPath.substring(0,totalPath.lastIndexOf("\\"));
            proyecto.setFecha(lector.readObject().toString());
            proyecto.setNUis(lector.readInt());

            go = true;
        }catch(Exception exc)
        {new JOptionPane().showMessageDialog(null,"No se puede encontrar el
archivo \n"+
totalPath,"Error",JOptionPane.ERROR_MESSAGE);
            go = false;
        }

        if(go)

```



```

    {
        String[] uis = null;
        if(proyecto.getNUis()>0)
        {
            uis = new String[proyecto.getNUis()];
            for (int i = 0; i<proyecto.getNUis(); i++)
            {
                try
                {
                    uis[i]=lector.readObject().toString();
                }
                catch(Exception exc){};
            }
        }
        proyecto.setUis(uis);
    }

lector.close();
}

public void guardarProyecto(String[] uis)
{
    proyecto.setUis(uis);
    proyecto.setNUis(uis.length);

    try
    {
        escritor = new ObjectOutputStream(new
FileOutputStream(proyecto.getRuta()+"\\"+proyecto.getNombre()+".lfp"));

        escritor.writeObject(proyecto.getNombre());
        escritor.writeObject(proyecto.getRuta());
        escritor.writeObject(proyecto.getFecha());
        escritor.writeInt(proyecto.getNUis());

        for (int i = 0; i<uis.length; i++)
        {
            escritor.writeObject(uis[i]);
        }

        escritor.writeChars("EOF");

        escritor.close();

    }catch(Exception exc){}
}

public void guardarComoProyecto(String Nombre,String Ruta,String
Fecha,String[] uis)
{
    String rutaI = proyecto.getRutaI();
    File nf = new File(rutaI);

    try {
        crearProyecto(Nombre,Ruta,Fecha);
        guardarProyecto(uis);
    }
}

```

```

        catch (Exception ex) {}

        String[] files = nf.list();

        File f1,f2;
        for (int i = 0; i<files.length; i++)
        {
            f1 = new File(rutaI+"\\\\"+files[i]);
            f2 = new File(proyecto.getRutaI()+"\\\\"+files[i]);

            if(f1.exists()&!f2.exists())
            {
                try
                {
                    UIs.UtilNavegador.copyFile(f1.getAbsolutePath(),f2.getAbsolutePath());
                }catch(Exception e){}
            }
        }

        public boolean getGo()
        {
            return go;
        }

        public Proyecto getProyecto()
        {
            return proyecto;
        }

        private ObjectInputStream lector = null;
        private ObjectOutputStream escritor = null;

        private File ProycFile;
        private File CodesFile;
        private File FormsFile;
        private File ImagesFile;

        private Proyecto proyecto;
        private boolean go = false;
    }

```

5.3.2.2.3 NuevoProyectoDialog.java

```

package ConfigProyecto;

import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;

```

```

import javax.swing.JLabel;
import javax.swing.ImageIcon;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.border.*;
import java.awt.Color;

public class NuevoProyectoDialog extends JDialog
{
    public NuevoProyectoDialog(JFrame owner)
    {
        this.setTitle("Nuevo Proyecto");
        this.setIconImage(new
ImageIcon("Recursos\\Iconos\\np.PNG").getImage());
        this.setLocation(owner.getX()+(owner.getWidth()-
459)/2,owner.getY()+(owner.getHeight()-397)/2);
        this.setLayout(null);
        this.setResizable(false);
        this.setModal(true);
        this.setSize(459,397);

        Banner.setBounds(2,2,450,85);

        Logo.setBounds(2,89,154,275);
        Logo.setBackground(Color.WHITE);

        Banner.setBorder(new
javax.swing.border.BevelBorder(BevelBorder.RAISED));
        Logo.setBorder(new
javax.swing.border.BevelBorder(BevelBorder.RAISED));

        l1.setBounds(170,95,60,20);
        t1.setBounds(170,115,275,20);
        l2.setBounds(170,140,60,20);
        t2.setBounds(170,160,275,20);
        b1.setBounds(360,185,86,20);

        b2.setBounds(235,340,100,20);
        b3.setBounds(345,340,100,20);

        this.add(Banner);
        this.add(Logo);
        this.add(l1);
        this.add(l2);
        this.add(t1);
        this.add(t2);
        this.add(b1);
        this.add(b2);
        this.add(b3);

        b1.setActionCommand("brw1");
        b2.setActionCommand("acep");
        b3.setActionCommand("canc");

        b1.addActionListener(listener);
        b2.addActionListener(listener);
        b3.addActionListener(listener);
        t1.addKeyListener(listener);

```

```

fchooser.setSelectionMode(JFileChooser.DIRECTORIES_ONLY);
    }

    public void b1Action()
    {

if(fchooser.showDialog(this, "Aceptar")==JFileChooser.APPROVE_OPTION)
    t2.setText(fchooser.getSelectedFile().getAbsolutePath());
    }

    public void setData1(String Data1)
    {
        this.Data1 = Data1;
        t1.setText(Data1);
    }

    public void setData2(String Data2)
    {
        this.Data2 = Data2;
        t2.setText(Data2);
    }

    public void b2Action()
    {
        if(t1.getText().trim() != "")
        {
            Data1 = t1.getText().trim();
            accepted = true;
        }
        else
        {
            JOptionPane.showMessageDialog(this, "El nombre de proyecto
no puede\ner procesado", "LFUICreator", JOptionPane.ERROR_MESSAGE, new
ImageIcon("Recursos\\Iconos\\Icono 64.png"));
            accepted = false;
        }

        if(t2.getText() != "")
            Data2 = t2.getText();
        else
            Data2 = "C:/";

        this.setVisible(false);
    }

    public void b3Action()
    {
        accepted = false;
        this.setVisible(false);
    }

    public boolean ifAccepted()
    {
        return accepted;
    }

```

```

        public String getData1()
        {
            return Data1;
        }

        public String getData2()
        {
            return Data2;
        }

        private JLabel Banner = new JLabel(new
        ImageIcon("Recursos\\Imágenes\\GeneralDialogBanner.png"));
        private JLabel Logo = new JLabel(new
        ImageIcon("Recursos\\Imágenes\\NewProjectDialog.png"));

        private JLabel l1 = new JLabel("Nombre:");
        private JLabel l2 = new JLabel("Ubicacion:");

        private JTextField t1 = new JTextField();
        private JTextField t2 = new JTextField();

        private JButton b1 = new JButton("Examinar");

        private JButton b2 = new JButton("Aceptar",new
        ImageIcon("Recursos\\Iconos\\acep.png"));
        private JButton b3 = new JButton("Cancelar",new
        ImageIcon("Recursos\\Iconos\\canc.png"));

        private NuevoProyectoDialogListener listener = new
        NuevoProyectoDialogListener(this);

        private JFileChooser fchooser = new JFileChooser();
        private String Data1;
        private String Data2;

        private boolean accepted = false;
    }

```

5.3.2.2.4 NuevoProyectoDialogListener.java

```

package ConfigProyecto;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.KeyListener;
import java.awt.event.KeyEvent;

class NuevoProyectoDialogListener implements ActionListener, KeyListener
{
    private NuevoProyectoDialog UiReference;

    public NuevoProyectoDialogListener(NuevoProyectoDialog UiReference)
    {
        this.UiReference = UiReference;
    }

```

```
    }  
  
    public void actionPerformed(ActionEvent e)  
    {  
        if(e.getActionCommand() == "brw1")  
            UiReference.b1Action();  
  
        if(e.getActionCommand() == "acep")  
            UiReference.b2Action();  
  
        if(e.getActionCommand() == "canc")  
            UiReference.b3Action();  
    }  
  
    public void keyPressed(KeyEvent e) {  
        if(e.getKeyCode() == KeyEvent.VK_ENTER)  
        {  
            UiReference.b2Action();  
            return;  
        }  
  
        if(e.getKeyCode() == KeyEvent.VK_ESCAPE)  
        {  
            UiReference.b3Action();  
            return;  
        }  
    }  
  
    public void keyTyped(KeyEvent e) {  
    }  
  
    public void keyReleased(KeyEvent e) {  
    }  
}
```

5.4 CONCLUSIÓN DE LA FASE DE CONSTRUCCIÓN

Durante la fase de construcción se complementó el diseño de la arquitectura base de LFUI Creator y se codificaron e integraron todos los componentes del sistema.

Las pruebas de unidad se aplicaron mediante el método de caja negra, adicionalmente se aplicaron pruebas de integración lo cual permitió detectar y

corregir fallas en el funcionamiento de los componentes y en la forma de acoplarse unos a otros.

Durante la construcción, se amortizaron los riesgos correspondientes a esta fase, logrando el propósito de tener lista la primera versión operativa del sistema.

CAPÍTULO VI

FASE DE TRANSICIÓN

6.1 INTRODUCCIÓN

La fase de transición tiene como finalidad poner el producto en manos de los usuarios finales, durante la cual se pueden desarrollar nuevas versiones actualizadas del producto, completar la documentación mediante la realización del manual de usuario, entrenar al usuario en el manejo del producto, y en general se realizan tareas relacionadas con el ajuste, configuración, instalación y facilidad de uso del producto.

6.2 MANUAL DE USUARIO

Como último producto se desarrolló el documento que servirá como base principal para los usuarios del sistema, para su correcto uso, este es el Manual de usuario. A continuación se presentará el mismo, en la siguiente página, como parte de esta fase de Transición.



LFUI Creator

manual de usuario

6.2.1 Introducción

Bienvenido al manual de usuario de LFUI Creator. Nuestro software le permitirá la generación automática de código fuente de interfaces gráficas de usuario Java Swing. Posicione y dé tamaño a componentes Java Swing sobre una ventana real, haga click en JLabels, JButtons, JTextFields, y JComboBoxes entre otros, y edite sus propiedades directamente en su lugar.

IMPORTANTE Este artículo está diseñado sólo como herramienta de referencia rápida y no está pensado con fines de enseñanza o de aprendizaje. También se espera que sea de completo entendimiento para el lector.

6.2.2 Interfaz de Inicio LFUI Creator

Proporciona un tiempo adicional de carga para establecer la configuración principal del sistema y garantizar el correcto despliegue de componentes de la Interfaz principal.

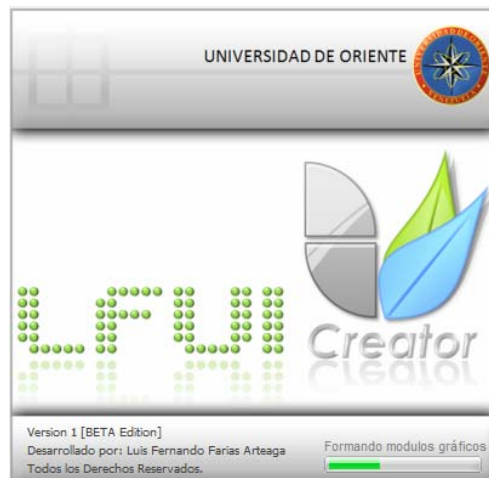


Figura 6.1: Pantalla de inicio del sistema

Fuente: Elaboración propia

6.2.3 Elementos de la Interfaz Principal

6.2.3.1 Opciones de Archivo de Proyecto

En las figuras 6.2 y 6.3 se muestran las opciones del sistema para la gestión de archivos de proyecto. Dichas opciones cumplen en totalidad el caso de uso Configurar Archivo. Aunado a esto está la opción Salir la cual anula la ejecución del sistema.

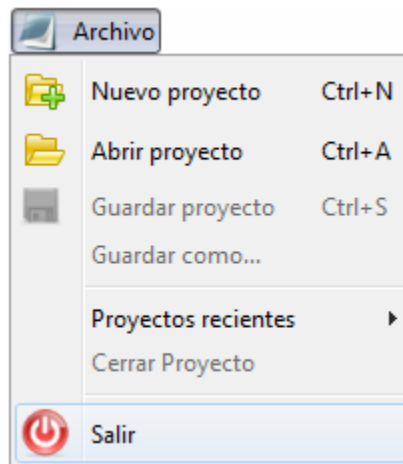


Figura 6.2: Opciones de Archivo de Proyecto en la Barra de Menú

Fuente: Elaboración propia

Nota: Las opciones en la barra de herramienta no están identificadas con una cadena de texto pero mantienen el icono en relación con su equivalente en la barra de menú. Además todas las opciones cuentan con un Tool Tip, o Tip de Herramienta, que indica la acción asociada a cada botón.



Figura 6.3: Opciones de Archivo de Proyecto en la Barra de Herramientas

Fuente: Elaboración propia

6.2.3.2 Opciones de Proyecto o gestión de UIs.

En las figuras 6.4 y 6.5 se muestran las opciones del sistema para la gestión de UIs. Estas opciones cumplen en su totalidad con el caso de uso Configurar Proyecto. Es importante agregar que las opciones en la barra de menú están simplificadas en partes simples. Esto quiere decir que cada opción puede realizarse para una UI en general o para todas las UIs pertenecientes al proyecto activo. Esto con la finalidad de dar mayor rapidez al usuario al momento de interactuar con el sistema, pero las acciones siguen permaneciendo iguales por lo que no representan otros casos de uso.

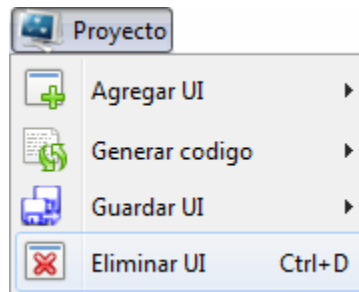


Figura 6.4: Opciones de Proyecto o Gestión de UIs en la Barra de Menú

Fuente: Elaboración propia



Figura 6.5: Opciones de Proyecto o Gestión de UIs en la Barra de Herramientas

Fuente: Elaboración propia

6.2.3.3 Opciones de Configuración de Sistema

La figura 6.6 muestra las opciones de Configuración del Sistema. En este caso no existe un equivalente en la barra de herramienta para estas opciones por lo poco usuales que tienden a ser este tipo de configuraciones. Este menu cumple a totalidad el caso de uso Configurar Sistema.

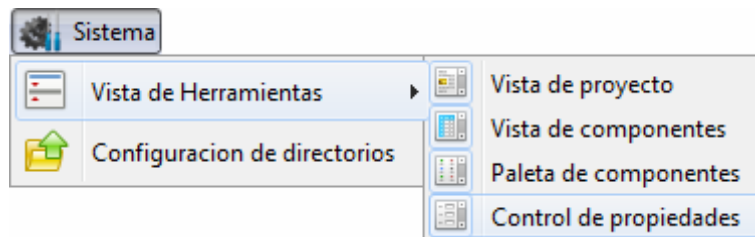


Figura 6.6: Opciones de Configuración de Sistema.

Fuente: Elaboración propia

6.2.3.4 Opciones de Ayuda del Sistema

La figura 6.7 muestra las distintas ayudas que el sistema le proporciona al usuario. LFUI Creator mantiene un enlace con la Documentación Java y permite al usuario consultarla en cualquier momento. De la misma forma trabaja con el manual de la

aplicacion para facilitar el acceso a la ayuda en la menor perdida de tiempo posible. El enlace Acerca de solo refleja los creditos de la aplicacion.

Las opciones del menu Ayuda cumplen con el caso de uso Mostrar Ayudas.

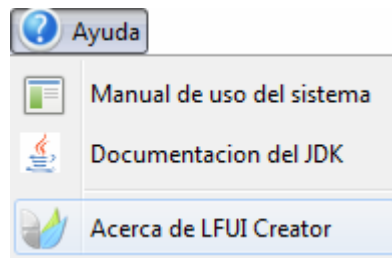


Figura 6.7: Opciones de Ayuda del Sistema

Fuente: Elaboración propia

6.2.3.5 Modo de Edición

El modo de edicion desactiva las opciones de insercion de componentes y permite la seleccion de algun objeto de la GUI activa para ser editado.

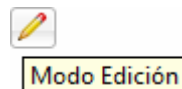


Figura 6.8: Modo de Edición

Fuente: Elaboración propia

6.2.3.6 Control de Vistas de Proyecto

La figura 6.9 muestra el control de vistas del proyecto activo, este es un selector de pestañas que nos permite alternar entre las UIs en edicion y los Codigos generados para alguna UI. Cuando no existe un proyecto activo el control activa y despliega la Pagina de Inicio o Bienvenida del sistema.

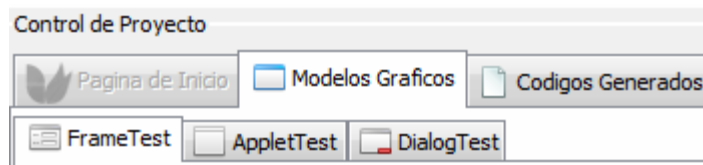


Figura 6.9: Control de Vistas del Proyecto

Fuente: Elaboración propia

6.2.3.7 Vista de Proyecto

La figura 6.10 muestra la vista del proyecto, en esta aparecen las Uis activas en el proyecto en edicion y nos permite navegar entre las mimas seleccionado alguna de ellas en el arbol de proyecto.

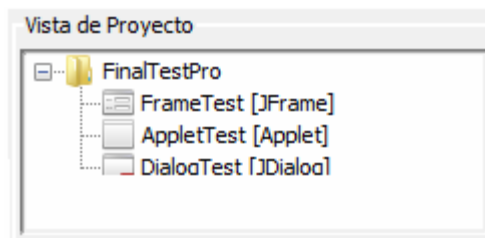


Figura 6.10: Vista de Proyecto

Fuente: Elaboración propia

6.2.3.8 Vista de Componentes

La figura 6.11 muestra la vista de componentes del sistema. La misma es un árbol que refleja los objetos que pertenecen a la UI en edición y la jerarquía de contenedores diseñada por el usuario. Esta vista denota cada componente con su nombre como variable, el tipo de objeto y un icono característico, todo esto para facilitar al usuario la comprensión de la UI que se está editando.

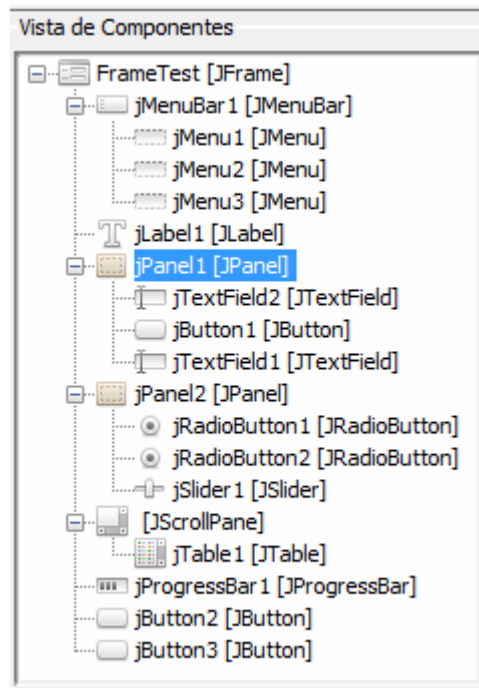


Figura 6.11: Vista de Componentes

Fuente: Elaboración propia

6.2.3.9 Paleta de Componentes

La figura 6.11 muestra la paleta de componentes, que no es mas que el panel de componentes que LFUI Creator permite agregar a una UI. Cada componente esta representado con una imagen relacionada a su forma grafica y cada boton posee tips asociados a su descripcion. Esta paleta proporciona todas las opciones reflejadas en el caso de uso Construir Modelo Grafico.

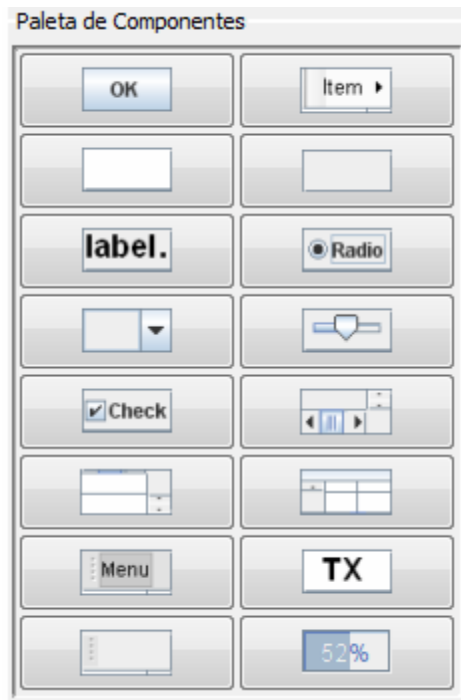


Figura 6.12: Paleta de Componentes

Fuente: Elaboración propia

6.2.3.10 Control de Propiedades

La figura 6.13 muestra el control de propiedades que es la ventana que muestra todos los atributos que el LFUI Creator permite editar para un tipo de componente seleccionado. Este control posee un selector manual de componentes y botones para actualizar y eliminar un componente. Este control permite llevar a cabo el caso de uso Definir propiedades de componente y en conjunto con la paleta de componentes completa el caso de uso Modelar Interfaz Grafica.

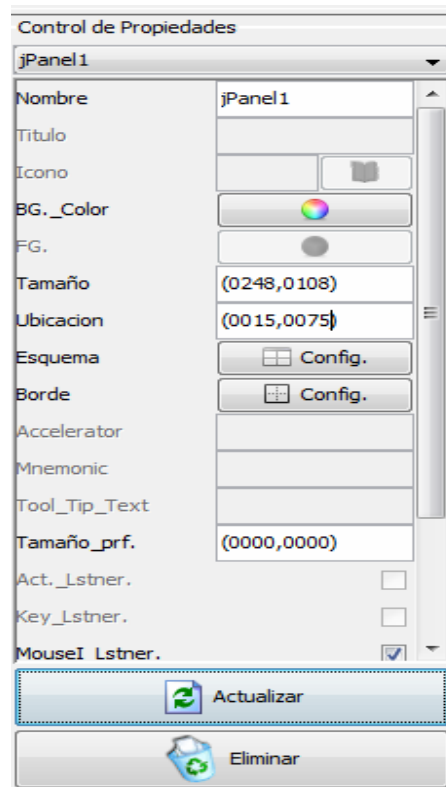




Figura 6.13: Control de Propiedades

Fuente: Elaboración propia

6.2.4 Gestión de Proyectos

6.2.4.1 Crear proyecto

Para crear un proyecto es necesario seleccionar la opción  Nuevo proyecto **Ctrl+N** en el menú Archivo o presionar el botón  en la barra de herramientas. Lo que sucede a continuación es que el sistema despliega el diálogo de Nuevo Proyecto. (Figura 6.14). La interfaz visible en este momento espera recibir un nombre válido y una ruta igualmente válida para generar un directorio de proyecto.

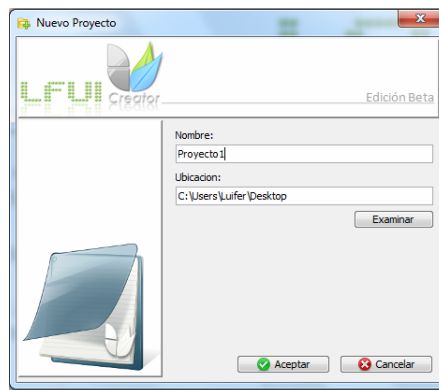




Figura 6.14: Diálogo de Nuevo Proyecto

Fuente: Elaboración propia

6.2.4.2 Abrir Proyecto

Para abrir un proyecto es necesario seleccionar la opción  Abrir proyecto **Ctrl+A** en el menú archivo a presionar el botón  en la barra de herramientas. Lo que sucede a continuación es que el sistema despliega el diálogo de Abrir Proyecto (Figura 6.15), el cual solicita la ruta del archivo de proyecto que se desea abrir.

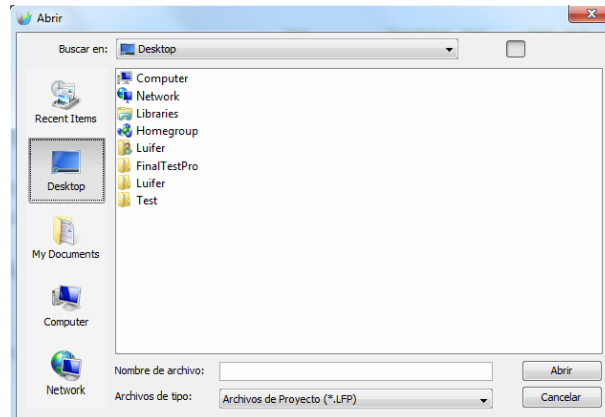
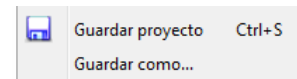


Figura 6.15: Dialogo para Abrir Proyecto.

Fuente: Elaboración propia

6.2.4.3 Guardar y Guardar como

Para guardar un proyecto es necesario seleccionar la opción en la barra de menú del sistema. “Guardar como” hace uso de la función “Guardar”, solicitando una nueva ruta de proyecto.



6.2.4.4 Proyectos recientes

La opción de “Proyectos Recientes” hace uso de la función “Abrir proyecto” usando las rutas de los últimos 10 proyectos editados por el sistema.


6.2.4.5 Cerrar proyecto

Cerrar proyecto solo retira el proyecto activo de la interfaz principal y coloca la Pagina de Inicio como simbolo de espera de alguna opcion del usuario.

NOTA: Todas las opciones que requieran hacer un cambio de archivo de proyecto notificaran al usuario de posibles cambios y daran la posibilidad de guardar proyecto, ignorar cambios o cancelar la accion.

6.2.5 Modelado de una Interfaz Gráfica

6.2.5.1 Agregar una nueva UI

Para agregar una nueva Interfaz Grafica al proyecto es necesario seleccionar la opcion  en el menu “Proyecto” de la barra de menu. A continuacion de despliega el dialogo de Nueva UI, (Figura 6.16), el cual solicita un nombre valido para la nueva GUI y un tipo de clase. De no ser seleccionado algun tipo se tomara por defecto la primera opcion.

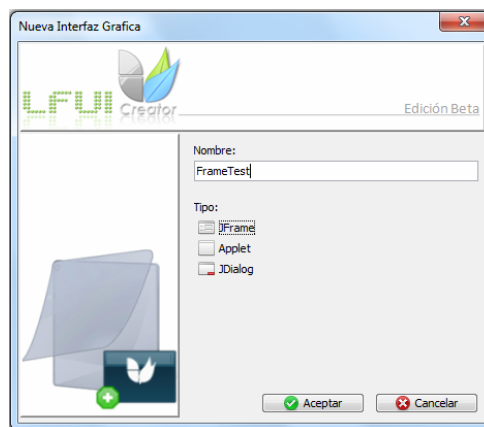


Figura 6.16: Dialogo de nueva UI

Fuente: Elaboración propia

6.2.5.2 Eliminar una UI

Esta opción remueve del archivo de proyecto los datos de la UI seleccionada al momento de la acción y luego elimina la referencia gráfica de dicha UI en la interfaz principal.

6.2.5.7 Agregar una UI Existente.

Esta opción hace uso de la opción Agregar UI pero no solicita la data de una nueva UI ni genera un formulario.

6.2.5.8 Agregar componente a UI.

Al seleccionar alguna opción en la paleta de componentes se abandona el modo edición y se entra en el modo inserción, este modo permite agregar componentes a la Interfaz en edición por medio de métodos gráficos sencillos.

Si el contenedor al que se desea agregar un componente posee algún tipo de esquema de administración, solo bastará hacer doble click sobre el mismo para agregar el componente, de lo contrario el usuario debe delimitar con el ratón el área del nuevo objeto a insertar. La figura 6.17 muestra este proceso.

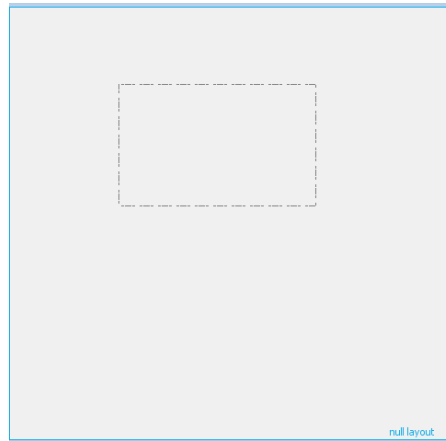


Figura 6.17: Delimitación de nuevo componente

Fuente: Elaboración propia

6.2.5.8 Modificar componente en UI

Para reflejar una modificación en algún componente de la UI es necesario seleccionar el mismo. Esto se puede lograr presionando el identificador relacionado al nombre del componente en la “Vista de Componentes” o directamente en el selector de la ventana de “Control de Propiedades”. También es posible hacer esto haciendo click sobre el componente que se desea editar. Ver figura 6.18

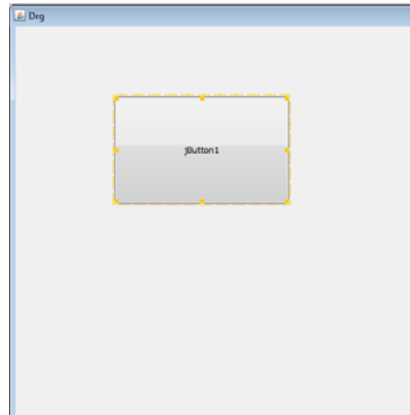

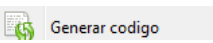



Figura 6.18: Selección de un componente

Fuente: Elaboración propia

Una vez seleccionado el component deseado se debe proceder a escribir los valores que se desean modificar en la “Ventana de Propiedades” y se refleja finalmente el cambio presionando el boton  de dicha ventana.

6.2.6 Generación de Código

La generacion de codigo de una UI se puede llevar a cabo seleccionando la opcion  en el menu “Proyecto” de la barra de Menu o presionando el boton  en la barra de herramienta. Esta opcion activa la vista de codigos en el “Control de Vistas” y agrega una pestaña con el codigo fuente generado para la UI en edicion y posiblemente otra pestaña de haber eventos asociados a los componentes en dicha UI. Esta segunda pestaña contendria el codigo fuente generado para una clase extra que gestione los eventos de la UI. Esto da mas elegancia al codigo fuente y libera a la clase de la UI de la pesada gestion de objetos capturadores de eventos. En las figuras 6.19 y 6.20 se puede observar la vista de codigo con las clases de codigo fuente generadas por LFUI Creator.

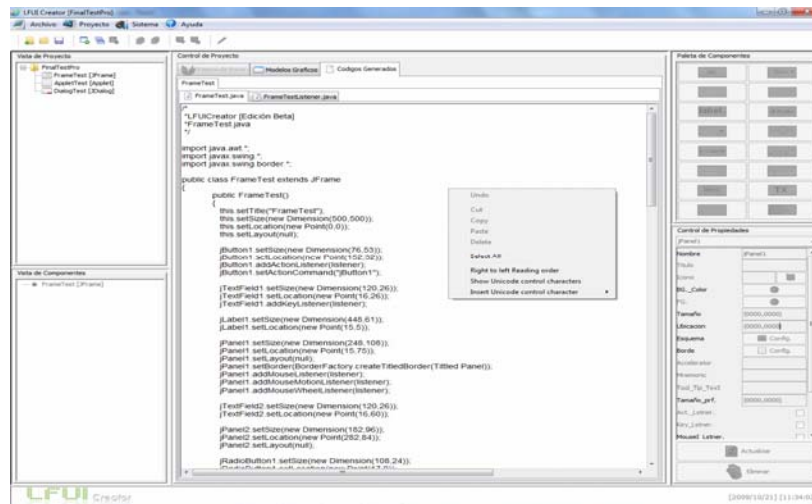


Figura 6.19: Código Generado para UI en edición

Fuente: Elaboración propia

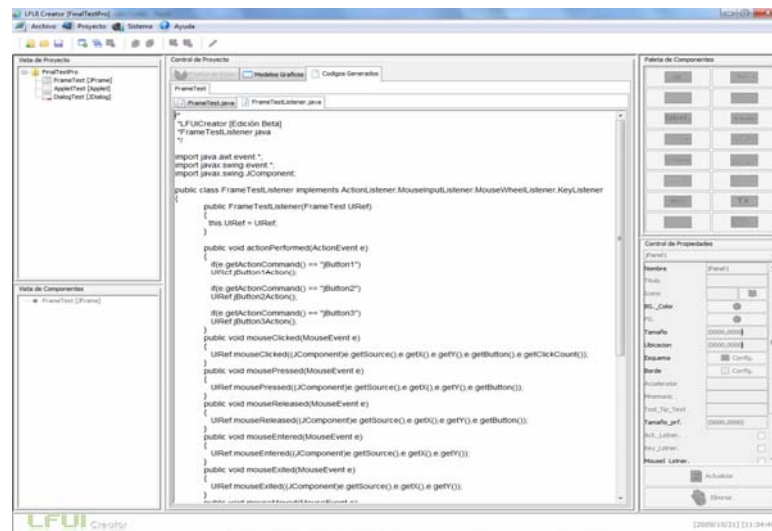


Figura 6.20: Código Generado para clase manejadora de eventos de la UI en edición

Fuente: Elaboración propia

CONCLUSIONES

Mediante este proyecto de investigación se creó una herramienta de programación, que haciendo uso de metodos graficos sencillos, permitirá crear paso a paso una GUI (Interfaz Grafica de Usuario), asi mismo se generará el código fuente java correspondiente, siguiendo fielmente los fundamentos de la Programación Orientada a Objetos, dicho código podrá ser manipulado por los usuarios de dicha herramienta.

Este proyecto se diseño con el objetivo principal de servir de apoyo, a las asignaturas Objetos y Abstracción de Datos, y Taller Objetos y Abstracción de Datos, ya que permitirá que el estudiante o profesor, este en contacto directo con todo el proceso de crear un código fuente de una Interfaz Grafica de Usuario Java.

El RUP, (Rational Unified Process), como metodología elegida, aunado al uso de UML como herramienta principal para la documentación y guía, permitieron la organización y desarrollo del sistema, cumpliendo con todas las etapas de este proceso y ayudó a la prevención de fallas que se pudieron presentar, evitando así un replanteamiento del proyecto, canalizando los diferentes flujos de trabajo necesarios por cada una de las fases de desarrollo.

La fase de inicio del proyecto jugó un papel fundamental, puesto que permitió obtener una noción clara del contexto del sistema, a través del Modelo de Dominio y el diagrama de Casos de Uso, logrando con ello la obtención de los requisitos necesarios para llevar a cabo el diseño. Los casos de uso tambien fueron de gran importancia para representar las condiciones y requerimientos del sistema, logrando esto a través de una buena documentación y análisis.

La fase de elaboración permitió eliminar la mayor cantidad de riesgos posibles empezando por los más altos o de más relevancia. También se definieron los flujos de trabajo en diseño y organización del sistema, representados por los diagramas correspondientes.

En la fase de construcción se procedió con la codificación de todos los componentes hasta obtener una versión beta del sistema. Los requisitos funcionales y no funcionales además de toda información obtenida en las etapas anteriores, fueron refinadas en esta fase, y adecuadas a las posibilidades de la herramienta de desarrollo que se usó.

La elección de Java como lenguaje de programación permitió hacer una limpia codificación del sistema ya que en las etapas anteriores se planteó una sólida estructura de clases en conformidad con el paradigma orientado a objetos y Java es un lenguaje de programación orientado a objetos de última generación y figura como uno de los lenguajes con un mayor crecimiento y amplitud. Por otra parte Java es netamente orientado a aplicaciones gráficas y posee potentes herramientas para el desarrollo de GUI's, lo que facilitó la construcción de Interfaces Gráficas organizadas, comunicativas y muy amigables.

Se realizó de manera exitosa la integración, pruebas y documentación de funcionamiento del sistema, se evaluó y depuró el sistema en varias iteraciones, obteniendo un software altamente funcional, al que puede realizarse mantenimiento y actualizaciones.

RECOMENDACIONES

- Mantener actualizados los documentos de diseño y manuales de usuario, con respecto a cualquier modificación o actualización que se realice al sistema y llevar un registro de estos.
- Agregar mas componentes a la gestion del sistema, asi la herramienta se hara mas versatil.
- Hacer mantenimiento a los archivos de sintaxis y de propiedad para prevenir algun fallo en la lectura de los mismos.

BIBLIOGRAFÍA

[1] **Barreto, A. y López, Y.** (1997). “Diseño e implementación de un case orientado a la solución de problemas matemáticos utilizando las técnicas del organigrama o diagrama de flujo”.

Trabajo de Grado. Universidad de Oriente, Anzoátegui.

[2] **Cortínez, C. y Martínez, J.** (1997). “Desarrollo de un tutor para la enseñanza de los algoritmos usados en las estructuras de datos utilizando técnicas de animación”.

Trabajo de Grado. Universidad de Oriente, Anzoátegui.

[3] **Rivas, N.** (2001). “Desarrollo de un software interactivo para la enseñanza de la asignatura Química I (10-1814) del área científica-tecnológica”.

Trabajo de Grado. Universidad de Oriente, Anzoátegui.

[4] **Grupo Soluciones Innova.** (2007). <http://www.rational.com.ar>

Disponible en: <http://www.rational.com.ar/herramientas/rup.html>

[5] **Wikipedia, La enciclopedia libre.** (2009). <http://es.wikipedia.org>

Disponible en: <http://es.wikipedia.org/wiki/RUP>.

[6] **Benevento, M. y Sánchez, C.** (2008). “Diseño de un sistema de información que permita la automatización de las actividades relacionadas con el mantenimiento a equipos utilizados en líneas de producción en una planta ensambladora de vehículos.”.

Trabajo de Grado. Universidad de Oriente, Anzoátegui.

- [7] **Romero N. y Chapa S.** (2008). “Diseño de Interfaces Visuales”.
Disponible en: <http://www.cs.cinvestav.mx/CursoVis/prinvisual.html>
- [8] **Boyus J.** (1998). “Java technology: The early years”.
- [9] **Sun Microsystems.** (2009). <http://java.com/es>
Disponible en: <http://java.com/es/about/>

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

TÍTULO	DESARROLLO DE UN SOFTWARE QUE PERMITA LA GENERACIÓN AUTOMÁTICA DE CÓDIGO FUENTE DE INTERFACES GRÁFICAS DE USUARIO (GUI), COMO APOYO A LA ASIGNATURA "TALLER DE OBJETOS Y ABSTRACCIÓN DE DATOS", IMPARTIDA EN EL DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS.
SUBTÍTULO	

AUTOR (ES):

APELLIDOS Y NOMBRES	CÓDIGO CULAC / E MAIL
Farias A., Luis F.	CVLAC: V-17.286.694 E MAIL: luifer_rocknroll@hotmail.com
	CVLAC: E MAIL:
	CVLAC: E MAIL:
	CVLAC: E MAIL:

PALÁBRAS O FRASES CLAVES:

INTERFACES GRÁFICAS

OBJETOS Y ABSTRACCIÓN DE DATOS

COMPUTACIÓN

CÓDIGO FUENTE

JAVA

MODELADO GRÁFICO

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

ÁREA	SUBÁREA
Ingeniería y Ciencias Aplicadas	Ingeniería en Computación

RESUMEN (ABSTRACT):

En el Dpto. de Computación y Sistemas del Núcleo de Anzoátegui, de la Universidad de Oriente, son dictadas las asignaturas Objetos y Abstracción de Datos, y Taller Objetos y Abstracción de Datos, siendo la segunda el apoyo práctico de la primera. Como herramienta de programación utilizada para la codificación de los proyectos en estas materias actualmente se usa Java el cual es un lenguaje de programación netamente orientado a aplicaciones gráficas y posee potentes herramientas para el desarrollo de GUI's. Debido a esto la enseñanza del uso de dichos objetos visuales requiere de un gran tiempo, y en consecuencia se ven reducidas las horas de clases dedicadas a los otros aspectos del contenido programático de las cátedras. En vista de esta situación, surge la necesidad de desarrollar un software con el que se pretende crear una herramienta de apoyo para estas materias. Esta herramienta debe permitir la generación automática de código fuente Java para GUI's diseñadas por el usuario, utilizando sencillos métodos gráficos. Dicho código seguirá las pautas de la Programación Orientada a Objetos y será presentado en forma elegante, esto con la finalidad de reforzar en los estudiantes los conocimientos necesarios para el desarrollo de una interfaz gráfica.

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

CONTRIBUIDORES:

APELLIDOS Y NOMBRES	ROL / CÓDIGO CVLAC / E_MAIL				
	García R., Zulirais A.	ROL	CA	AS X	TU
CVLAC:		V-10.299.576			
E_MAIL		zzzuliii@hotmail.com			
E_MAIL					
Bastardo, Jose L.	ROL	CA	AS	TU	JU X
	CVLAC:	V-6.890.832			
	E_MAIL	fractalcaos@gmail.com			
	E_MAIL				
Cortinez, Claudio.	ROL	CA	AS	TU	JU X
	CVLAC:	V-12.155.334			
	E_MAIL	cortinezclaudio@hotmail.com			
	E_MAIL				
	ROL	CA	AS	TU	JU X
	CVLAC:				
	E_MAIL				
	E_MAIL				

FECHA DE DISCUSIÓN Y APROBACIÓN:

2009	10	23
AÑO	MES	DÍA

LENGUAJE. SPA

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

ARCHIVO (S):

NOMBRE DE ARCHIVO	TIPO MIME
Tesis Lfuicreator.doc	Aplication/msword

CARACTERES EN LOS NOMBRES DE LOS ARCHIVOS: A B C D E F G H I J K L M N O P Q
R S T U V W X Y Z . a b c d e f g h i j k l m n o p q r s t u v w x y z . 0 1 2 3 4 5 6 7 8 9 .

ALCANCE

ESPACIAL: _____ (OPCIONAL)

TEMPORAL: _____ (OPCIONAL)

TÍTULO O GRADO ASOCIADO CON EL TRABAJO:

_____ Ingeniero en Computación _____

NIVEL ASOCIADO CON EL TRABAJO:

_____ Pre-Grado _____

ÁREA DE ESTUDIO:

_____ Departamento de Computación y Sistemas _____

INSTITUCIÓN:

_____ Universidad de Oriente – Núcleo de Anzoátegui _____

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

DERECHOS

_____ De acuerdo con el artículo 44 del reglamento de trabajo de grado: _____

“Los trabajos de grado son de exclusiva propiedad de la Universidad de Oriente y sólo podrán ser utilizados a otros fines con el consentimiento del consejo de núcleo respectivo, quien lo participará al Consejo Universitario”. _____

AUTOR

Farias A., Luis F.

AUTOR

AUTOR

TUTOR

García, Zulirais

JURADO

Cortinez, Claudio

JURADO

Bastardo, José L.

POR LA SUBCOMISIÓN DE TESIS