

**ESCUELA POLITÉCNICA
SUPERIOR DE CÓRDOBA**
Universidad de Córdoba



TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Especialidad de Computación

**SimAS 3.0 descendente predictivo.
Simulador de analizadores
sintácticos descendentes
predictivos.**

Manual Técnico

Autor

D. Antonio Llamas García

Director

Prof. Dr. Nicolás Luis Fernández García

Septiembre, 2025



UNIVERSIDAD DE CÓRDOBA

Título

Incluir el título

Resumen

Incluir el resumen

Palabras claves

Incluir las palabras claves

Title

Incluir el título en inglés

Abstract

Incluir el resumen en inglés

Keywords

Incluir las palabras claves en inglés

Índice general

I Presentación	1
1. Introducción	3
1.1. Presentación	3
1.1.1. Compilación	3
1.1.2. Interpretación	4
1.1.3. Fases de traducción	4
1.2. Definición del problema real	5
1.3. Definición del problema técnico	6
1.3.1. Funcionamiento	7
1.3.2. Entorno	7
1.3.2.1. Interfaz con el usuario	7
1.3.2.2. Entorno de desarrollo	8
1.3.2.3. Entorno de ejecución	8
1.3.3. Ciclo de mantenimiento	8
1.3.4. Competencia	9
1.3.5. Aspecto externo	9
1.3.6. Estandarización	10
1.3.7. Calidad y fiabilidad	10
1.3.8. Validación y verificación	11
1.3.9. Programa de tareas	12
1.3.10. Pruebas	13
1.3.10.1. Tipos de Pruebas	13
1.3.10.2. Descripción de las Pruebas	13
1.3.11. Seguridad	14

ÍNDICE GENERAL

2. Objetivos	15
2.1. Objetivo principal	15
2.2. Objetivos específicos	15
2.3. Objetivos personales	16
 3. Antecedentes	17
3.1. Introducción	17
3.2. Fundamentos teóricos del análisis sintáctico	17
3.2.1. Tipos de analizadores sintácticos	17
3.2.2. Gramáticas de contexto libre	18
3.2.3. Derivación con una gramática de contexto libre	19
3.2.4. Árbol de derivación o árbol sintáctico	21
3.2.5. Analizadores sintácticos descendentes	22
3.2.6. Analizadores sintácticos ascendentes	24
3.3. SimAS 1.0	28
3.4. SimAS 2.0	31
3.4.1. Fallos generales identificados en SimAS 2.0	39
3.4.2. Fallos encontrados en Windows	40
3.4.3. Fallos encontrados en Ubuntu	41
3.4.4. Fallos encontrados en MacOS	43
3.5. Justificación del trabajo	44
3.5.1. Comparativa entre versiones	46
 4. Restricciones	47
4.1. Introducción	47
4.2. Factores iniciales	47
4.2.1. Límites temporales	47
4.2.2. Entorno de software	48
4.3. Factores estratégicos	48
4.3.1. Metodología	48
4.3.2. Lenguaje de programación	49
4.3.3. Entorno de desarrollo	49

4.3.4. Interfaz gráfica de la aplicación	50
4.3.5. Sistema Operativo	51
4.3.6. Representación de las gramáticas de contexto libre	51
4.3.7. Generación de informes de gramáticas y de simulación	51
4.3.8. Implementación de la ayuda	52
4.3.9. Implementación del tutorial	52
4.3.10. Representación de los árboles sintácticos	53
5. Recursos	55
5.1. Recursos humanos	55
5.2. Recursos de hardware	55
5.3. Recursos de software	56
5.3.1. Sistema Operativos	56
5.3.2. Entorno de desarrollo y lenguaje de programación	56
5.3.3. Herramientas de documentación	56
5.3.4. Herramientas para la edición de diagramas	56
II Análisis	57
6. Especificación de requisitos	59
6.1. Introducción	59
6.2. Actores	59
6.3. Descripción modular del sistema	59
6.3.1. Módulo del Editor de Gramáticas	60
6.3.2. Módulo del Simulador Gráfico Descendente Predictivo	61
6.3.2.1. Creación y carga de gramáticas	61
6.3.2.2. Generación de conjuntos <i>Primero</i> y <i>Siguiente</i>	61
6.3.2.3. Construcción de la tabla predictiva	61
6.3.2.4. Implementación de funciones de recuperación de errores .	61
6.3.2.5. Simulación del análisis sintáctico descendente predictivo .	62
6.3.2.6. Generación del árbol sintáctico descendente	62
6.3.2.7. Creación de informes	62

ÍNDICE GENERAL

6.3.3. Módulo del Tutorial	63
6.3.3.1. Estructura del tutorial	63
6.3.4. Módulo de Ayuda	63
6.3.4.1. Contenidos de la ayuda	63
6.4. Requisitos funcionales	64
6.4.1. Requisitos funcionales del Editor de gramáticas	64
6.4.2. Requisitos funcionales del Simulador	66
6.4.3. Requisitos funcionales del Tutorial	67
6.4.4. Requisitos funcionales de la Ayuda	67
6.5. Requisitos no funcionales	67
6.6. Requisitos de la información	68
6.6.1. Gramática de contexto libre	68
6.6.2. Editor de gramáticas	69
6.6.3. Simulador	70
6.6.4. Ayuda de la aplicación	71
6.6.5. Tutorial de la aplicación	71
6.6.6. Informes	71
6.7. Requisitos de la interfaz	71
7. Análisis funcional	73
7.1. Introducción	73
7.2. Actores del Sistema	74
7.3. Especificación de casos de uso	74
7.3.1. Diagrama de contexto: CU-0. Sistema	74
7.3.2. CU-1. Editar Gramáticas	76
7.3.2.1. CU-1.1. Crear gramática	80
7.3.2.2. CU-1.2. Editar vocabulario de la gramática	82
7.3.2.3. CU-1.3. Editar producción	84
7.3.2.4. CU-1.4. Seleccionar el símbolo inicial	86
7.3.2.5. CU-1.5. Cerrar gramática	88
7.3.2.6. CU-1.6. Recuperar gramática	88
7.3.2.7. CU-1.7. Guardar gramática	90

7.3.2.8. CU-1.8. Validar gramática	92
7.3.2.9. CU-1.9. Generar informe de gramática	95
7.3.2.10. CU-1.10. Transferir al simulador	96
7.3.2.11. CU-1.11. Obtener gramática	98
7.3.2.12. CU-1.12. Actualizar visualización	99
7.3.2.13. CU-1.13. Comprobar producción	100
7.3.3. CU-2. Simular análisis sintáctico descendente predictivo	101
7.3.3.1. CU-2.1. Simular Análisis Descendente Predictivo	102
7.3.3.2. CU-2.1.2. Gestionar conjunto Primero y Siguiente	106
7.3.3.3. CU-2.2. Buscar gramática	106
7.3.3.4. CU-2.3. Completar tabla con funciones de error	106
7.3.3.5. CU-2.4. Analizar cadena	107
7.3.3.6. CU-2.5. Selecciona velocidad de simulación	107
7.3.3.7. CU-2.6. Generar informe de simulación	107
7.3.3.8. CU-2.7. Construir Árbol Sintáctico Descendente	107
7.3.4. CU-3. Consultar Tutorial	107
7.3.5. CU-4. Consultar Ayuda	109
7.4. Validación de los casos de uso	110
7.4.1. Validación del Módulo de Edición de Gramática	111
7.4.2. Validación del Módulo de Simulación de Gramática	112
7.4.3. Validación del Módulo del Tutorial	112
7.4.4. Validación del Módulo de la Ayuda	112
III Diseño	115
8. Diseño de paquetes	117
8.1. Introducción	117
8.1.1. Arquitectura general del sistema	117
8.1.2. Principios de diseño aplicados	117
8.1.3. Métricas generales del sistema	118
8.2. Especificación detallada de los paquetes	118
8.2.1. Paquete bienvenida	118

ÍNDICE GENERAL

8.2.1.1.	Responsabilidades principales	119
8.2.1.2.	Clases del paquete	119
8.2.1.3.	Características técnicas	119
8.2.1.4.	Flujo de ejecución	119
8.2.2.	Paquete editor	119
8.2.2.1.	Arquitectura y diseño	120
8.2.2.2.	Patrones de diseño implementados	120
8.2.2.3.	Clases del paquete por funcionalidad	120
8.2.2.4.	Características técnicas	121
8.2.2.5.	Dependencias y relaciones	121
8.2.2.6.	Flujo de trabajo del asistente	122
8.2.3.	Paquete simulador	122
8.2.3.1.	Arquitectura y diseño	122
8.2.3.2.	Algoritmo implementado	122
8.2.3.3.	Patrones de diseño implementados	123
8.2.3.4.	Clases del paquete por funcionalidad	123
8.2.3.5.	Características técnicas avanzadas	124
8.2.3.6.	Dependencias críticas	125
8.2.3.7.	Flujo de análisis sintáctico	125
8.2.4.	Paquete gramatica	125
8.2.4.1.	Arquitectura del modelo de datos	125
8.2.4.2.	Patrones de diseño implementados	126
8.2.4.3.	Clases del paquete por funcionalidad	126
8.2.4.4.	Algoritmos implementados	127
8.2.4.5.	Características técnicas	127
8.2.4.6.	Relaciones críticas con el sistema	128
8.2.4.7.	Flujo de transformación grammatical	128
8.2.5.	Paquete centroayuda	128
8.2.5.1.	Arquitectura del sistema de ayuda	128
8.2.5.2.	Contenido del sistema de ayuda	129
8.2.5.3.	Características técnicas	129
8.2.5.4.	Relaciones con el sistema	130

8.2.6.	Paquete resources	130
8.2.6.1.	Arquitectura de recursos	130
8.2.6.2.	Categorización detallada de recursos	130
8.2.6.3.	Directorio icons/ especializado	131
8.2.6.4.	Características técnicas	132
8.2.6.5.	Sistema de carga y gestión	132
8.2.6.6.	Integración con el sistema	132
8.2.7.	Paquete utils	132
8.2.7.1.	Arquitectura de servicios transversales	133
8.2.7.2.	Patrones de diseño implementados	133
8.2.7.3.	Clases del paquete por funcionalidad	133
8.2.7.4.	Características técnicas avanzadas	134
8.2.7.5.	Algoritmos implementados	134
8.2.7.6.	Integración crítica con el sistema	135
8.2.7.7.	Métricas de impacto	135
8.2.8.	Paquete vistas	135
8.2.8.1.	Arquitectura de presentación	135
8.2.8.2.	Archivos FXML por categoría	136
8.2.8.3.	Sistema de estilos CSS	137
8.2.8.4.	Patrón MVC implementado	137
8.2.8.5.	Características técnicas	137
8.2.8.6.	Integración con el sistema	138
8.3.	Diagrama de paquetes de la aplicación	138
8.3.1.	Análisis de dependencias del diagrama	140
8.3.1.1.	Dependencias críticas del núcleo	140
8.3.1.2.	Dependencias de recursos	140
8.3.2.	Métricas cuantitativas de la arquitectura	140
8.3.2.1.	Distribución de clases por paquete	140
8.3.2.2.	Complejidad algorítmica por paquete	140
8.3.2.3.	Matriz de dependencias	141
8.4.	Conclusiones sobre la arquitectura de paquetes	141
8.4.1.	Principios de diseño cumplidos	141

ÍNDICE GENERAL

8.4.2. Patrones de diseño implementados	141
8.4.3. Ventajas de la arquitectura actual	142
8.4.3.1. Ventajas técnicas	142
8.4.3.2. Ventajas del dominio	142
8.4.4. Limitaciones identificadas	142
8.4.5. Recomendaciones para evolución futura	142
9. Diseño de clases	143
9.1. Introducción	143
9.2. Diseño de clases por paquetes	143
9.2.1. Paquete bienvenida	144
9.2.1.1. Clases del paquete bienvenida	144
9.2.1.2. Clase Bienvenida	144
9.2.1.3. Clase MenuPrincipal	145
9.2.1.4. Dependencias internas del paquete bienvenida	148
9.2.1.5. Patrones de diseño en el paquete bienvenida	149
9.2.2. Paquete gramática	149
9.2.2.1. Clases del paquete gramática	150
9.2.2.2. Clase Simbolo	150
9.2.2.3. Clase Terminal	151
9.2.2.4. Clase NoTerminal	151
9.2.2.5. Clase Consecuente	152
9.2.2.6. Clase Antecedente	153
9.2.2.7. Clase Produccion	153
9.2.2.8. Clase Gramatica	155
9.2.2.9. Clase FilaTablaPredictiva	157
9.2.2.10. Clase FuncionError	159
9.2.2.11. Clase TablaPredictiva	161
9.2.2.12. Clase TablaPredictivaPaso5	163
9.2.2.13. Dependencias internas del paquete gramática	166
9.2.2.14. Patrones de diseño en el paquete gramática	168
9.2.3. Paquete utils	168

9.2.3.1.	Clases del paquete utils	168
9.2.3.2.	Clase ActualizableTextos	169
9.2.3.3.	Clase LanguageItem	169
9.2.3.4.	Clase LanguageListCell	170
9.2.3.5.	Clase TabManager	171
9.2.3.6.	Clase TabPaneMonitor	173
9.2.3.7.	Clase SecondaryWindow	175
9.2.3.8.	Dependencias internas del paquete utils	177
9.2.3.9.	Patrones de diseño en el paquete utils	178
9.2.4.	Paquete editor	178
9.2.4.1.	Clases del paquete editor	179
9.2.4.2.	Clase EditorWindow	179
9.2.4.3.	Clase Editor	181
9.2.4.4.	Clase PanelCreacionGramatica	183
9.2.4.5.	Clase PanelCreacionGramaticaPaso1	184
9.2.4.6.	Clase PanelCreacionGramaticaPaso2	185
9.2.4.7.	Clase PanelCreacionGramaticaPaso3	187
9.2.4.8.	Clase PanelCreacionGramaticaPaso4	188
9.2.4.9.	Clase PanelProducciones	189
9.2.4.10.	Clase PanelSimbolosTerminales	190
9.2.4.11.	Clase PanelSimbolosNoTerminales	192
9.2.4.12.	Dependencias internas del paquete editor	194
9.2.4.13.	Patrones de diseño en el paquete editor	194
9.2.5.	Paquete simulador	195
9.2.5.1.	Clases del paquete simulador	195
9.2.5.2.	Interfaz PanelNuevaSimDescPaso	195
9.2.5.3.	Clase PanelNuevaSimDescPaso1	196
9.2.5.4.	Clase PanelNuevaSimDescPaso2	198
9.2.5.5.	Clase PanelNuevaSimDescPaso3	198
9.2.5.6.	Clase PanelNuevaSimDescPaso4	199
9.2.5.7.	Clase PanelNuevaSimDescPaso5	200
9.2.5.8.	Clase PanelNuevaSimDescPaso6	201

ÍNDICE GENERAL

9.2.5.9. Clase PanelSimuladorDesc	202
9.2.5.10. Clase PanelSimulacion	205
9.2.5.11. Clase SimulacionFinal	206
9.2.5.12. Clase PanelGramaticaOriginal	208
9.2.5.13. Clase NuevaFuncionError	209
9.2.5.14. Clase EditorCadenaEntradaController	210
9.2.5.15. Dependencias internas del paquete simulador	210
9.2.5.16. Patrones de diseño en el paquete simulador	211
9.3. Paquetes de soporte	213
9.3.1. Paquete centroayuda	213
9.3.2. Paquete vistas	213
9.3.3. Paquete resources	214
9.4. Dependencias entre Paquetes	214
9.4.1. Dependencias Detalladas por Paquete	214
9.4.1.1. Dependencias del paquete bienvenida	214
9.4.1.2. Dependencias del paquete editor	215
9.4.1.3. Dependencias del paquete gramática	215
9.4.1.4. Dependencias del paquete simulador	216
9.4.1.5. Dependencias del paquete utils	216
9.4.1.6. Dependencias transversales del sistema	217
9.4.2. Análisis de acoplamiento	217
9.4.2.1. Acoplamiento bajo (recomendable)	217
9.4.2.2. Acoplamiento alto (áreas de atención)	217
9.4.3. Principios SOLID aplicados	217
9.4.3.1. Principio de Responsabilidad Única (SRP)	218
9.4.3.2. Principio de Abierto/Cerrado (OCP)	218
9.4.3.3. Principio de Sustitución de Liskov (LSP)	219
9.4.3.4. Principio de Segregación de Interfaces (ISP)	219
9.4.3.5. Principio de Inversión de Dependencias (DIP)	220
9.4.4. Evaluación de Calidad Arquitectural	221
9.4.4.1. Métricas de Acoplamiento y Cohesión	221
9.4.4.2. Puntos Fuertes de la Arquitectura	221

9.4.4.3. Áreas de Mejora Identificadas	221
9.5. Conclusiones	222
10. Diseño de la interfaz	223
10.1. Introducción	223
10.2. Menú principal	224
10.3. Barras de herramientas	225
10.4. Ventana del editor	226
10.4.1. Panel de edición	226
10.5. Ventana del simulador	229
10.5.1. Panel de simulación	230
10.5.2. Ventana de Gramática Original	231
10.6. Ventana de simulación	231
10.7. Ventana de Visualización del Árbol Sintáctico	232
10.8. Ventana de Derivación Sintáctica	232
10.9. Ventana de asistente	232
10.10. Ventanas de edición	237
IV Pruebas	239
11. Pruebas	241
11.1. Introducción	241
11.2. Pruebas del módulo Editor	241
11.2.1. Almacenamiento y recuperación de gramáticas	241
11.2.2. Validación de gramáticas	241
11.3. Pruebas del módulo Simulador	241
Bibliografía	243
Bibliografía	243

Índice de figuras

1.1. Fases Principales en el Proceso de Traducción	5
3.1. Ejemplo de árbol sintáctico asociado a una derivación de una gramática de contexto libre.	21
3.2. Árbol sintáctico: paso inicial del análisis sintáctico descendente predictivo .	23
3.3. Árbol sintáctico: paso final del análisis sintáctico descendente predictivo .	23
3.4. Árbol sintáctico: pasos (a) inicial y (b) final del análisis sintáctico ascendente SLR	27
3.5. Interfaz del editor de SimAS 1.0.	29
3.6. Panel del editor de SimAS 1.0	30
3.7. Ventana del simulador de SimAS 1.0.	30
3.8. Interfaz del editor de SimAS 2.0.	33
3.9. Corrección de gramáticas de SimAS 2.0.	34
3.10. Conjunto Primero y Siguiente de SimAS 2.0.	34
3.11. Tabla predictiva de SimAS 2.0.	35
3.12. Tabla predictiva de SimAS 2.0.	36
3.13. Tabla predictiva con funciones de errores de SimAS 2.0.	37
3.14. Tabla predictiva con funciones de errores de SimAS 2.0.	38
3.15. Generación repititiva de la visualización de la gramática	42
3.16. Creación de una nueva función de error	42
3.17. Abrir gramática en MacOS.	43
3.18. Duplicación símbolos no terminales MacOS	44
6.1. Descomposición modular del sistema	60
7.1. Diagrama de contexto del Sistema	74
7.2. CU-1: Editar Gramáticas	77

ÍNDICE DE FIGURAS

7.3. CU-1.1: Crear gramática	81
7.4. CU-1.2: Editar vocabulario de la gramática	82
7.5. CU-1.3: Editar producción	84
7.6. CU-1.4: Seleccionar el símbolo inicial	86
7.7. CU-1.6: Recuperar gramática	89
7.8. CU-1.7: Guardar gramática	91
7.9. CU-1.8: Validar gramática	93
7.10. CU-1.9: Generar informe de gramática	95
7.11. CU-1.10: Transferir al simulador	97
7.12. CU-1.12: Actualizar visualización de la gramática	99
7.13. CU-2 Simular análisis sintáctico predictivo	101
7.14. CU-2.1 Simular Análisis descendente	103
7.15. CU-3: Consultar Tutorial	108
7.16. CU-4: Consultar Ayuda	109
8.1. Diagrama UML de arquitectura de paquetes de SimAS 3.0	139
9.1. Diagrama de clases del paquete gramática - SimAS 3.0	167
9.2. Diagrama de dependencias del paquete utils - SimAS 3.0	178
9.3. Diagrama de clases del paquete editor - SimAS 3.0	197
9.4. Diagrama de dependencias del paquete simulador - SimAS 3.0	212
10.1. Menú principal de SimAS	224
10.2. Barra de herramientas de SimAS.	225
10.3. Ventana del editor.	227
10.4. Panel de edición.	228
10.5. Ventana de simulación.	229
10.6. Panel de simulación.	230
10.7. Gramática Original.	231
10.8. Ventana de simulación.	233
10.9. Ventana de simulación.	234
10.10. Ventana de simulación.	235
10.11. Ventana de asistente.	236

ÍNDICE DE FIGURAS

10.12Ventana de edición.	237
----------------------------------	-----

Índice de tablas

3.1. Tabla de análisis descendente predictivo	24
3.2. Pasos del análisis descendente predictivo	25
3.3. Tabla de análisis sintáctico SLR	26
3.4. Pasos del análisis sintáctico ascendente SLR	26
3.5. Comparación entre versiones de SimAS.	46
7.1. Caso de uso: CU-0. Sistema	75
7.2. Caso de uso: CU-1. Edición de gramáticas	78
7.3. Caso de uso: CU-1.1 Crear gramática	81
7.4. Caso de uso: CU-1.2 Editar vocabulario de la gramática	82
7.5. Caso de uso: CU-1.3: Editar producción	84
7.6. Caso de uso: CU-1.4 Seleccionar el símbolo inicial	87
7.7. Caso de uso: CU-1.6 Recuperar gramática	89
7.8. Caso de uso: CU-1.7 Guardar gramática	92
7.9. Caso de uso: CU-1.8 Validar gramática	94
7.10. Caso de uso: CU-1.9 Generar informe de gramática	95
7.11. Caso de uso: CU-1.10 Transferir al simulador	97
7.12. Caso de uso: CU-1.12 Actualizar visualización de la gramática	99
7.13. Caso de uso: CU-2 Simular análisis sintáctico	102
7.14. Caso de uso: CU-2.1 Simular Análisis descendente predictivo	104
7.15. Caso de uso: CU-3. Consultar Tutorial	108
7.16. Caso de uso: CU-4. Consultar Ayuda	110
7.17. Matriz de correspondencia RF-Casos de uso	111
7.18. Matriz de correspondencia RF-Casos de uso	112
7.19. Matriz de correspondencia RF-CU, Módulo tutorial	112

ÍNDICE DE TABLAS

7.20. Matriz de correspondencia RF-CU, Módulo ayuda	113
8.1. Distribución de clases Java por paquete	140
8.2. Complejidad algorítmica de los paquetes principales	140
8.3. Matriz de dependencias entre paquetes	141
9.1. Clase Bienvenida	144
9.2. Clase MenuPrincipal - Interfaz principal de navegación	145
9.3. Clase Simbolo - Clase base para símbolos gramaticales	150
9.4. Clase Terminal - Símbolos terminales de la gramática	151
9.5. Clase NoTerminal - Símbolos no terminales de la gramática	151
9.6. Clase Consecuente - Parte derecha de las producciones	152
9.7. Clase Antecedente - Parte izquierda de las producciones	153
9.8. Clase Produccion - Reglas de producción gramatical	154
9.9. Clase Gramatica - Núcleo del modelo de datos	155
9.10. Clase FilaTablaPredictiva - Filas de la tabla predictiva	158
9.11. Clase FuncionError - Funciones de manejo de errores	159
9.12. Clase TablaPredictiva - Tabla de análisis sintáctico	161
9.13. Clase TablaPredictivaPaso5 - Tabla predictiva extendida	164
9.14. Clase ActualizableTextos - Interfaz de internacionalización	169
9.15. Clase LanguageItem - Elemento de idioma	169
9.16. Clase LanguageListCell - Celda de lista de idiomas	170
9.17. Clase TabManager - Gestor central de pestañas	171
9.18. Clase TabPaneMonitor - Monitor de pestañas	173
9.19. Clase SecondaryWindow - Ventanas secundarias	175
9.20. Clase EditorWindow - Ventana principal del editor	179
9.21. Clase Editor - Interfaz principal de edición	181
9.22. Clase PanelCreacionGramatica - Asistente de creación	183
9.23. Clase PanelCreacionGramaticaPaso1 - Paso 1: Metadatos	185
9.24. Clase PanelCreacionGramaticaPaso2 - Paso 2: Símbolos	186
9.25. Clase PanelCreacionGramaticaPaso3 - Paso 3: Producciones	187
9.26. Clase PanelCreacionGramaticaPaso4 - Paso 4: Validación	188
9.27. Clase PanelProducciones - Gestión de producciones	189

9.28. Clase PanelSimbolosTerminales - Gestión de terminales	190
9.29. Clase PanelSimbolosNoTerminales - Gestión de no terminales	192
9.30. Interfaz PanelNuevaSimDescPaso - Interfaz de pasos de simulación	195
9.31. Clase PanelNuevaSimDescPaso1 - Paso 1 de simulación	196
9.32. Clase PanelNuevaSimDescPaso2 - Paso 2 de simulación	198
9.33. Clase PanelNuevaSimDescPaso3 - Paso 3 de simulación	198
9.34. Clase PanelNuevaSimDescPaso4 - Paso 4 de simulación	199
9.35. Clase PanelNuevaSimDescPaso5 - Paso 5 de simulación	200
9.36. Clase PanelNuevaSimDescPaso6 - Paso 6 (Simulador) de simulación	201
9.37. Clase PanelSimuladorDesc - Controlador principal de simulación	202
9.38. Clase PanelSimulacion - Panel de simulación interactiva	205
9.39. Clase SimulacionFinal - Simulación completa avanzada	206
9.40. Clase PanelGramaticaOriginal - Panel de gramática original	208
9.41. Clase NuevaFuncionError - Creación de funciones de error	209
9.42. Clase EditorCadenaEntradaController - Editor de cadenas de entrada	210

ÍNDICE DE TABLAS

Parte I

Presentación

Capítulo 1

Introducción

1.1. Presentación

La elección del lenguaje de programación adecuado es una decisión fundamental en el desarrollo de software. Este proceso implica considerar varios factores, como los requisitos del proyecto, la eficiencia, la facilidad de mantenimiento y la disponibilidad de herramientas y bibliotecas. Una vez seleccionado el lenguaje, surge la necesidad de traducir el código escrito en ese lenguaje a instrucciones que una computadora pueda entender y ejecutar.

La traducción del código se realiza a través de dos enfoques principales: compilación e interpretación. Estos enfoques difieren en su forma de transformar el código fuente en código ejecutable.

1.1.1. Compilación

La compilación implica la traducción del código de alto nivel a código máquina, que es el lenguaje específico de la computadora. Para ilustrar este proceso, consideremos un ejemplo simple en el lenguaje C:

```
#include <stdio.h>

int main() {
    printf("Hola, mundo!\n");
    return 0;
}
```

Cuando compilamos este programa, el compilador traduce el código C a instrucciones específicas del procesador en código máquina. Estas instrucciones se almacenan en un archivo ejecutable, que puede ser ejecutado por la computadora.

1.1.2. Interpretación

En contraste, la interpretación implica ejecutar el código fuente directamente, línea por línea, sin generar un archivo ejecutable independiente. Un ejemplo común de un lenguaje interpretado es Python:

```
print("Hola, mundo!")
```

En este caso, el intérprete de Python ejecuta cada línea de código a medida que se encuentra, generando la salida esperada sin producir un archivo ejecutable.

1.1.3. Fases de traducción

Tanto en la compilación como en la interpretación, el proceso de traducción consta de varias fases:

1. **Análisis:** en esta fase se comprueba que el programa fuente está bien escrito.
 - Análisis léxico: divide el código en componentes léxicos (*tokens*) significativos.
 - Análisis sintáctico: verifica la estructura gramatical del código.
 - Análisis semántico: examina el significado y la coherencia del código.
2. **Síntesis:** esta fase se encarga de generar el código del programa ejecutable.
 - Generación de código intermedio: crea una representación intermedia del código. Es una fase opcional pero recomendable.
 - Optimización del código intermedio: permite mejorar el código sin tener en cuenta el dispositivo donde se ejecutará. Es una fase opcional pero recomendable.
 - Generación de código final: produce el código ejecutable o la salida del programa.
 - Optimización: mejora el código para aumentar su eficiencia.

Estas fases aseguran que el código se traduzca de manera precisa y eficiente, garantizando su funcionamiento correcto.

Además, en el proceso de traducción, juegan un papel fundamental dos componentes auxiliares:

- **Administrador de la tabla de símbolos:** este componente almacena información crucial sobre variables, funciones y otros elementos del programa. Proporciona un registro detallado que permite al traductor acceder y gestionar eficientemente los elementos del código fuente.
- **Gestor de errores:** este componente es esencial para manejar cualquier inconveniente que pueda surgir durante la traducción. Detecta y gestiona errores de sintaxis, semántica u otros problemas que podrían comprometer la integridad y funcionalidad del programa final.

En la Figura 1.1, se presenta un diagrama que resume las fases principales en el proceso de traducción, desde el análisis hasta la generación del código final.

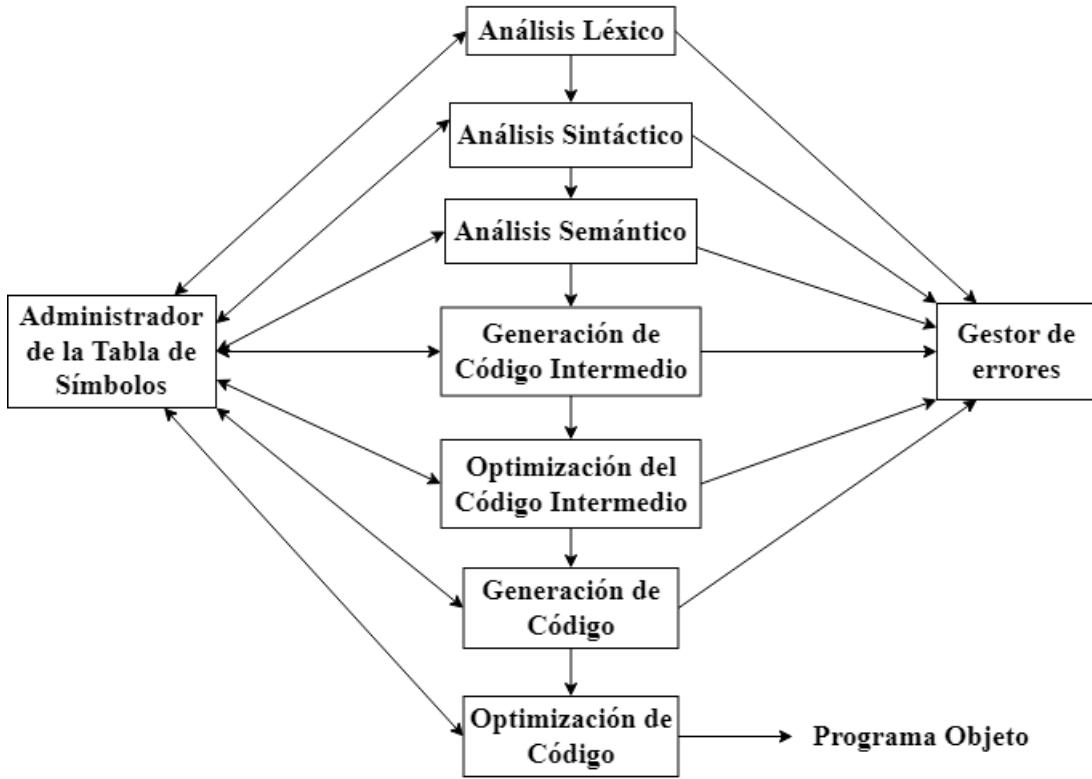


Figura 1.1: Fases Principales en el Proceso de Traducción

1.2. Definición del problema real

Este proyecto de fin de carrera se enfoca en una tarea crucial: la corrección, mejora y ampliación de “SimAS: Simuladores de Analizadores Sintácticos”, que es una aplicación desarrollada en java que tiene como objetivo ayudar a la docencia de la asignatura de Procesadores de Lenguajes de tercer curso de la especialidad de Computación del Grado de Ingeniería Informática.

SimAS proporciona a los estudiantes una comprensión profunda del funcionamiento de la fase de análisis sintáctico en compiladores e intérpretes. Tiene la capacidad de simular tanto analizadores sintácticos descendentes como ascendentes, facilitando que los estudiantes comprendan mejor el funcionamiento de estos métodos de análisis sintáctico.

SimAS es una aplicación de software que posee dos versiones. SimAS 1.0 fue desarrollado por Vanesa González Pérez [27] como Proyecto de Fin de Carrera de Ingeniería Informática en 2015. Esta versión permitía realizar el análisis sintáctico descendente predictivo y el análisis ascendente LR (SLR, LR - canónico y LALR). Esta versión fue corregida y ampliada en la versión SimAS 2.0, que fue desarrollada por Juan Antonio Fernández Díaz [5] en su Trabajo de Fin de Grado en 2023. Esta versión permitía generar informes en formato pdf y generar los árboles sintácticos asociados a la derivaciones de los analizadores sintácticos.

A pesar de la funcionalidad satisfactoria de SimAS, se han detectado una serie de errores y deficiencias que demandan atención inmediata.

- Al cargar o modificar gramáticas, la aplicación debe ordenar sus reglas de producción en función del símbolo no terminal de la parte izquierda.
- Al crear o modificar gramáticas, la aplicación no genera bien el archivo XML [30] con dicha gramática.
- La aplicación no genera o genera de forma incorrecta los informes en formato PDF.
- No genera correctamente los conjuntos Primero y Siguiente de algunas gramáticas. Estos conjuntos son necesarios para realizar el análisis sintáctico descendente predictivo y el análisis sintáctico ascendente SLR.
- La aplicación no genera correctamente los árboles sintácticos asociados a las derivaciones.
- La interfaz produce algunas ventanas no interactivas o que no aportan valor.

Además, se pueden incorporar las siguientes mejoras para enriquecer la aplicación:

- Diseñar una interfaz más intuitiva que genere todas las ventanas dentro de una misma interfaz y no en diferentes ventanas.
- Permitir el trabajo con varias gramáticas a la vez.
- Implementar la posibilidad de cambiar el idioma de la aplicación.

Una descripción más detallada de las versiones de SimAS se puede consultar en el capítulo 3 de Antecedentes.

Aunque las versiones anteriores SimAS 1.0 y 2.0 permiten simular los análisis sintácticos descendente y ascendente, el presente Trabajo de Fin de Grado se va a centrar exclusivamente en corregir todos los fallos encontrados y en mejorar el análisis sintáctico descendente predictivo. La nueva aplicación se denominará “SimAS 3.0 descendente predictivo”. Se considera que abordar también la corrección de los errores de la simulación del análisis sintáctico ascendente requeriría un esfuerzo superior al exigido en un Trabajo de Fin de Grado. Se debe tener en cuenta que la versión SimAS 2.0 [5] fue realizada en un Trabajo de Fin de Grado de doble especialidad que tiene una carga docente de 450 horas, superior a las 300 horas de un Trabajo de Fin de Grado de una única especialidad.

1.3. Definición del problema técnico

El presente Trabajo de Fin de Grado pretende desarrollar una aplicación informática multiplataforma, denominada “SimAS 3.0 descendente predictivo”, que permita corregir y ampliar la simulación del análisis sintáctico descendente predictivo desarrollado en las versiones anteriores de SimAS 1.0 [27] y 2.0 [5].

1.3.1. Funcionamiento

La aplicación propuesta se diseñará como una herramienta de escritorio multiplataforma con el objetivo de proporcionar a los usuarios una experiencia interactiva y educativa. Los principales módulos que integrará esta aplicación son los siguientes:

- **Editor de gramáticas de contexto libre:** este módulo permitirá a los usuarios crear, cargar, guardar y modificar gramáticas de contexto libre. La interfaz proporcionará opciones intuitivas para trabajar con las reglas de producción, símbolos terminales y no terminales, así como la definición del símbolo inicial.
- **Simulador gráfico descendente:** esta funcionalidad simulará el proceso de análisis sintáctico descendente predictivo, utilizando la gramática especificada en el editor. El simulador generará una derivación por la izquierda y mostrará el árbol sintáctico descendente resultante.
- **Generación de árboles sintácticos descendentes:** se implementará la funcionalidad para generar árboles sintácticos de forma precisa y visualmente comprensible. Esta característica permitirá a los usuarios visualizar de manera clara la estructura jerárquica de las derivaciones sintácticas.
- **Tutorial interactivo:** este módulo proporcionará una guía paso a paso sobre los fundamentos y el funcionamiento de los métodos de análisis sintáctico. Los usuarios podrán acceder a explicaciones detalladas y ejemplos prácticos para comprender mejor los conceptos.
- **Ayuda integrada:** se incluirá una sección de ayuda en la interfaz para brindar asistencia instantánea sobre el uso de la aplicación. Los usuarios podrán acceder a información detallada sobre cada componente de la interfaz y los distintos módulos que componen la aplicación.

El funcionamiento detallado de cada uno de estos módulos se abordará en el capítulo 6 de Especificación de requisitos para proporcionar una comprensión completa de la aplicación.

1.3.2. Entorno

El entorno en el que se desarrolla y ejecuta una aplicación es fundamental para garantizar su funcionalidad, usabilidad y despliegue efectivo. En este apartado, se detallan los aspectos relacionados con la interfaz de usuario, el entorno de desarrollo y el entorno de ejecución de la aplicación “SimAS 3.0 descendente predictivo”, proporcionando una visión general de los componentes y requisitos clave para su funcionamiento.

1.3.2.1. Interfaz con el usuario

La aplicación contará con una interfaz gráfica de usuario (GUI¹) diseñada para optimizar la experiencia del usuario, especialmente para los estudiantes. Se busca que

¹ *Graphical User Interface*

todo el programa se ejecute en una única ventana, posibilitando la aparición de distintas pestañas, con el fin de simplificar la navegación.

La interfaz estará diseñada para ser simple y completa al mismo tiempo, permitiendo a los usuarios realizar todas las operaciones necesarias para trabajar con gramáticas. Entre las funcionalidades específicas que ofrecerá la interfaz, se incluyen la capacidad de crear nuevas gramáticas e importar y editar gramáticas creadas previamente. Además, se proporcionará soporte para interacciones complejas, como la entrada de texto y la selección de archivos.

1.3.2.2. Entorno de desarrollo

El desarrollo y mantenimiento del código de la aplicación se realizará utilizando diversas fuentes de información disponibles en Internet, así como técnicas de inteligencia artificial. El entorno de desarrollo integrado (IDE) principal utilizado será IntelliJ [13], que ofrece una amplia gama de características y herramientas para el desarrollo de aplicaciones Java. Además, se utilizarán herramientas adicionales, como sistemas de control de versiones (Git [29]) para el control del código fuente, y se gestionará la documentación y los recursos del proyecto en un repositorio de GitHub [9].

1.3.2.3. Entorno de ejecución

La aplicación estará diseñada para ser ejecutada en un entorno multiplataforma y compatible con los principales sistemas operativos, incluyendo Windows, macOS y Linux. No se requerirán configuraciones específicas del sistema operativo o del entorno de ejecución para garantizar el funcionamiento correcto de la aplicación. Además, la aplicación se ejecutará en una máquina virtual de Java [19], lo que proporcionará una mayor portabilidad y compatibilidad con diferentes sistemas. Esto significa que los usuarios finales no necesitarán instalar ninguna dependencia externa ni cumplir con requisitos adicionales de instalación, lo que facilitará su despliegue y uso.

1.3.3. Ciclo de mantenimiento

La gestión de actualizaciones de software es un proceso crítico que garantiza la evolución continua y la mejora de la aplicación a lo largo del tiempo. En el caso de “SimAS 3.0 descendente predictivo”, se implementará un ciclo de mantenimiento diseñado para abordar nuevas funcionalidades, correcciones de errores y adaptaciones a los cambios en los entornos de ejecución.

El ciclo de mantenimiento de “SimAS 3.0 descendente predictivo” se basará en un enfoque proactivo y modular, permitiendo la incorporación de mejoras de manera eficiente y sin interrupciones significativas en la experiencia del usuario. Aunque el autor principal del proyecto no asumirá la responsabilidad directa del mantenimiento futuro, se establecerán procedimientos claros para la gestión de actualizaciones, asegurando la continuidad y la calidad del software.

Este enfoque modular no solo facilitará el mantenimiento y la actualización del

software, sino que también fomentará la colaboración y la contribución de la comunidad de usuarios y desarrolladores. De esta manera, SimAS 3.0 podrá adaptarse de manera ágil a las necesidades cambiantes de los usuarios y a las demandas del entorno tecnológico.

1.3.4. Competencia

El desarrollo de SimAS 3.0 se sitúa en un contexto de continua evolución y expansión en el ámbito de las herramientas educativas para el análisis sintáctico en la informática. Aunque no buscamos competir comercialmente, es importante comprender el panorama actual y las características distintivas de SimAS 3.0 en relación con otras soluciones disponibles.

A través de un análisis exhaustivo en el Capítulo 3, se explorará el terreno de las herramientas educativas existentes para el análisis sintáctico. Se examinarán tanto las aplicaciones desarrolladas en proyectos académicos previos, como otras soluciones disponibles en el mercado. Este análisis permitirá identificar fortalezas, debilidades y oportunidades de mejora para SimAS 3.0.

Si bien se reconocen las contribuciones de otras herramientas educativas, SimAS 3.0 buscará destacarse por su enfoque innovador y sus funcionalidades distintivas. Entre estas funcionalidades, se encuentra la capacidad de generar árboles sintácticos tanto descendentes como ascendentes, ofreciendo una experiencia de aprendizaje única y completa para los estudiantes de informática.

1.3.5. Aspecto externo

En esta sección, se abordarán aspectos clave relacionados con la experiencia del usuario y la documentación asociada a SimAS 3.0.

- **Diseño de la interfaz de usuario:** la interfaz de usuario de SimAS 3.0 se diseñará cuidadosamente para garantizar una experiencia visual atractiva y una navegación intuitiva. Se prestará especial atención a la disposición de los elementos, la coherencia visual y la accesibilidad, con el objetivo de proporcionar un entorno de trabajo cómodo y eficiente para los usuarios.
- **Documentación detallada:**
 - *Manual técnico:* este documento proporcionará una descripción exhaustiva de los aspectos técnicos del desarrollo de SimAS 3.0. Se incluirán detalles sobre la arquitectura del software, las tecnologías utilizadas y las decisiones de diseño fundamentales.
 - *Manual de usuario:* el manual de usuario de SimAS 3.0 contendrá instrucciones claras y concisas sobre la instalación, configuración y uso de la aplicación. Se incluirán ejemplos prácticos y capturas de pantalla para guiar al usuario a través de las diferentes funcionalidades de la aplicación.
 - *Manual de código:* este documento estará dirigido a desarrolladores y proporcionará una visión detallada del código fuente de SimAS 3.0. Se describirá la

estructura del proyecto, las convenciones de codificación y la documentación interna del código para facilitar su comprensión y mantenimiento.

- **Distribución y almacenamiento:** SimAS 3.0 estará disponible para su descarga en línea a través de un repositorio público, lo que facilitará el acceso a la última versión del software y de la documentación asociada. Además, se explorarán opciones para distribuir la aplicación en medios físicos o mediante servicios de almacenamiento en la nube, asegurando así su disponibilidad y accesibilidad para los usuarios.

1.3.6. Estandarización

En esta sección, se describirán los principios de diseño y usabilidad que guiarán el desarrollo de SimAS 3.0, con el objetivo de garantizar una experiencia de usuario coherente y satisfactoria.

- **Diseño centrado en el usuario:** SimAS 3.0 se diseñará pensando en las necesidades y expectativas del usuario final. Se realizarán investigaciones de usuarios y pruebas de usabilidad para asegurar que la interfaz de usuario sea intuitiva y fácil de usar.
- **Consistencia y coherencia:** la interfaz de usuario de SimAS 3.0 mantendrá una apariencia y comportamiento coherentes en todas sus funciones y elementos. Se seguirán estándares de diseño reconocidos y se aplicarán patrones de diseño comunes para garantizar una experiencia consistente para el usuario.
- **Sensibilidad y guía del usuario:** la interfaz de SimAS 3.0 guiará al usuario a través de la aplicación, proporcionando retroalimentación clara y orientación en cada paso del proceso. Se utilizarán mensajes informativos y visuales para informar al usuario sobre el estado y las acciones disponibles.
- **Interacción intuitiva:** se priorizará una interacción sencilla y fluida en SimAS 3.0. Los controles y las funciones de la aplicación se diseñarán de manera intuitiva, minimizando la necesidad de aprendizaje por parte del usuario y facilitando el uso de la aplicación desde el primer momento.
- **Claridad y legibilidad:** la interfaz gráfica de SimAS 3.0 se diseñará con un enfoque en la claridad y la legibilidad. Se utilizarán colores y tipografías adecuados para garantizar una fácil comprensión de la información presentada, evitando la sobrecarga visual y la confusión del usuario.

1.3.7. Calidad y fiabilidad

En esta sección, se abordarán los aspectos de calidad, fiabilidad y seguridad que son fundamentales en el desarrollo y uso de SimAS 3.0. Se buscará garantizar el correcto funcionamiento de la aplicación, así como proteger la integridad y privacidad de los usuarios.

- **Calidad y fiabilidad:** SimAS 3.0 se desarrollará con un enfoque en la calidad del software y la fiabilidad de su funcionamiento. Se utilizarán herramientas actualizadas como IntelliJ IDEA y la Máquina Virtual de Java para respaldar la calidad del código y su ejecución. Además, la aplicación será sometida a exhaustivas pruebas para detectar y corregir errores antes de su lanzamiento.
- **Seguridad:** la seguridad de SimAS 3.0 será una prioridad, tanto en términos de protección contra acciones incorrectas por parte del usuario como en la seguridad del dispositivo de almacenamiento. Se implementarán mecanismos de seguridad en la aplicación para evitar cualquier manipulación maliciosa o acceso no autorizado a los datos del usuario. Se garantizará que el usuario no tenga acceso a ninguna funcionalidad aparte de aquellas proporcionadas explícitamente por la aplicación, asegurando así que no se realicen acciones no deseadas ni se acceda a datos sensibles. Asimismo, se garantizará que la distribución y uso de la aplicación cumplan con los principios de software libre, siguiendo la licencia GPL de GNU para proteger su distribución y uso adecuados.

1.3.8. Validación y verificación

El proceso de validación y verificación de SimAS 3.0 es esencial para garantizar su correcto funcionamiento y la detección temprana de posibles errores.

- **Validación:** se centrará en asegurar que la aplicación cumple con los requisitos y expectativas del usuario final. Se llevarán a cabo pruebas funcionales para verificar que todas las funcionalidades operan según lo esperado, y se realizarán pruebas de aceptación para garantizar que la aplicación satisface las necesidades del usuario.
- **Verificación:** por otro lado, se centrará en asegurar que el código y la lógica de la aplicación son correctos y libres de errores. Se realizarán pruebas unitarias para verificar el correcto funcionamiento de cada componente individualmente, así como pruebas de integración para garantizar que los componentes interactúan correctamente entre sí.

El proceso estará dividido en varias etapas, cada una con su respectiva documentación detallada:

- **Planificación de pruebas:** se definirán los objetivos y alcance de las pruebas, así como los recursos y herramientas necesarios para llevarlas a cabo.
- **Diseño de pruebas:** se elaborará un plan detallado de las pruebas a realizar, incluyendo los casos de prueba específicos y los criterios de aceptación.
- **Ejecución de pruebas:** se llevarán a cabo las pruebas según el plan establecido, registrando los resultados obtenidos y cualquier incidencia detectada.
- **Análisis de resultados:** se analizarán los resultados de las pruebas para identificar posibles áreas de mejora o corrección. Se documentarán los errores encontrados y se propondrán soluciones para su resolución.

El capítulo 11 detallará cada una de estas etapas y proporcionará una visión completa del proceso de validación y verificación de SimAS 3.0.

1.3.9. Programa de tareas

El desarrollo del Trabajo de Fin de Grado estará compuesto por las siguientes actividades:

1. Exploración inicial

- Comprensión y prueba de SimAS 2.0 [5].
- Definición del problema técnico.
- Establecimiento de los objetivos.
- Revisión de los antecedentes.
- Identificación de los recursos iniciales y estratégicos.
- Selección de las herramientas de software y hardware.
- Investigación del entorno IntelliJ [13] y del lenguaje de programación Java [19].

2. Análisis y diseño

- Especificación de requisitos: funcionales, no funcionales, de información y de la interfaz.
- Elaboración de los casos de uso.
- Validación de los casos de uso.
- Elaboración de los diagramas de secuencia.

3. Diseño

- Diseño de la arquitectura del sistema.
- Diseño de paquetes y clases.
- Diseño de la interfaz

4. Implementación

- Desarrollo del código.
- Implementación de pruebas unitarias.
- Validación del código desarrollado.

5. Documentación

- Elaboración del manual técnico, manual de usuario y manual de código.
- La redacción del manual técnico se realizará durante todo el proceso de desarrollo del Trabajo de Fin de Grado.

1.3.10. Pruebas

Las pruebas y la validación son elementos esenciales en el desarrollo de cualquier aplicación. Estos procesos permiten asegurar que la aplicación cumpla con los estándares de calidad y funcionalidad esperados. En el caso de SimAS 3.0, se llevarán a cabo diversas actividades de evaluación y validación para garantizar su correcto funcionamiento y fiabilidad.

1.3.10.1. Tipos de Pruebas

Se llevarán a cabo varios tipos de pruebas para evaluar distintos aspectos de SimAS 3.0:

- **Pruebas de Funcionalidad:** estas pruebas se centran en verificar que todas las funciones y características de SimAS 3.0 funcionen como se espera. Se realizarán pruebas exhaustivas para cada módulo y componente de la aplicación.
- **Pruebas de Interfaz de Usuario:** estas pruebas se enfocarán en la usabilidad y la experiencia del usuario. Se evaluará la interfaz gráfica para garantizar que sea intuitiva y fácil de usar.
- **Pruebas de Rendimiento:** se llevarán a cabo pruebas para evaluar el rendimiento y la velocidad de SimAS 3.0, especialmente en situaciones de carga y con grandes conjuntos de datos.
- **Pruebas de Compatibilidad:** se realizarán pruebas en diferentes sistemas operativos y entornos de ejecución para garantizar que SimAS 3.0 funcione correctamente en diversas configuraciones.

1.3.10.2. Descripción de las Pruebas

Cada prueba estará estructurada de la siguiente manera:

- **Objetivo:** se definirá claramente el objetivo de la prueba, es decir, qué se espera evaluar o validar.
- **Problema:** se identificarán y documentarán los problemas o errores detectados durante la ejecución de la prueba, junto con información detallada sobre cómo reproducirlos.
- **Solución:** se propondrán soluciones y acciones correctivas para abordar los problemas identificados durante las pruebas, asegurando así la mejora continua de la aplicación.

El próximo capítulo 11 detallará los procedimientos y resultados de las pruebas llevadas a cabo en “SimAS 3.0 descendente predictivo” como parte de este Trabajo de Fin de Grado.

1.3.11. Seguridad

La seguridad en la presente aplicación se centra principalmente en garantizar su correcto funcionamiento sin comprometer la integridad de los sistemas donde se ejecute. Dado que la aplicación no maneja ni almacena datos personales de los usuarios, el enfoque de seguridad se orienta hacia la prevención de posibles daños a los equipos o entornos de ejecución.

Se implementarán medidas para asegurar que durante la ejecución de la aplicación, los usuarios no puedan realizar acciones que puedan comprometer su funcionamiento o la estabilidad del sistema. Además, se prestará especial atención a la protección del dispositivo de almacenamiento donde se ejecute la aplicación, garantizando que no se produzcan daños o alteraciones no deseadas.

En línea con la versión anterior, la aplicación se distribuirá libremente, sin restricciones de uso, copia o distribución. Esto se fundamenta en su naturaleza académica y en su carácter no lucrativo. No se aplicarán medidas de protección contra la distribución libre de la aplicación, permitiendo su acceso y uso por parte de cualquier usuario interesado en aprender o utilizarla con fines educativos.

En cuanto a la licencia, la aplicación seguirá los principios de la Licencia Pública General de GNU (GPL), que garantiza la libertad de uso, modificación y redistribución del software. Esta licencia ayuda a prevenir acciones malintencionadas relacionadas con la copia o reproducción del software, asegurando que permanezca accesible para la comunidad académica y de desarrollo de software.

Capítulo 2

Objetivos

2.1. Objetivo principal

El objetivo principal de este Trabajo de Fin de Grado es desarrollar la aplicación informática “SimAS 3.0 descendente predictivo”, que permita la simulación de analizadores sintácticos descendentes predictivos, así como la generación de árboles sintácticos asociados a sus derivaciones sintácticas.

2.2. Objetivos específicos

El objetivo principal se desglosa en los siguientes objetivos específicos:

- Revisar la versión anterior de SimAS 2.0 [5] para identificar y corregir errores y deficiencias del análisis sintáctico descendente predictivo, tales como:
 - Mejorar la edición y el almacenamiento de gramáticas de contexto libre.
 - Depurar la validación automática de gramáticas de contexto libre: comprobar que la gramática no tiene símbolos inútiles.
 - Permitir la visualización de las gramáticas originales y transformadas.
 - Definir funciones para el tratamiento automático de errores.
 - Garantizar la simulación correcta de gramáticas que incluyan reglas épsilon.
 - Generar correctamente los conjuntos Primero y Siguiiente.
 - Asegurar la generación adecuada de informes en formato PDF.
- Ampliar las capacidades de SimAS para incluir la generación de árboles sintácticos de las derivaciones producidas por los analizadores sintácticos descendente predictivos:
 - Permitir la generación paso a paso de los árboles sintácticos asociados a las derivaciones.
 - Facilitar la exportación de los árboles sintácticos generados en diversos formatos.

2.3. Objetivos personales

En el ámbito personal y profesional, se plantean los siguientes objetivos a alcanzar durante el desarrollo de este Trabajo de Fin de Grado:

- **Contribuir a la educación en informática:** desarrollar una aplicación informática que no solo cumpla con los requisitos académicos del proyecto, sino que también pueda utilizarse como herramienta educativa en el ámbito de la enseñanza de los analizadores sintácticos. Se busca que esta aplicación sea útil para estudiantes y profesores en el aprendizaje y la enseñanza de este tema fundamental.
- **Profundizar en Java y Java Swing:** aprovechar este proyecto como una oportunidad para mejorar mis habilidades en el lenguaje de programación Java y en el uso de sus bibliotecas, especialmente Java Swing [20]. Se busca obtener un conocimiento más sólido y práctico en el desarrollo de aplicaciones gráficas de usuario, lo que puede ser beneficioso para futuros proyectos y oportunidades profesionales.
- **Explorar otras herramientas de desarrollo:** esta tarea implica investigar y adquirir conocimientos sobre diversas herramientas y tecnologías utilizadas en el desarrollo de aplicaciones informáticas, además de Java y Java Swing. Entre estas herramientas, un ejemplo destacado es Scene Builder [10]. Scene Builder es una herramienta gráfica que permite diseñar interfaces de usuario de manera intuitiva y visual, utilizando la tecnología JavaFX. Permite arrastrar y soltar componentes de la interfaz gráfica, configurar propiedades y establecer el diseño de la aplicación de forma rápida y eficiente. Explorar y familiarizarse con herramientas como Scene Builder amplía el repertorio de herramientas disponibles para el desarrollo de interfaces de usuario, lo que permite evaluar y seleccionar las mejores soluciones para proyectos futuros.
- **Aprender sobre el ciclo de desarrollo de software:** obtener una comprensión más profunda del proceso de desarrollo de software, desde la planificación y el diseño hasta la implementación, las pruebas y la documentación. Se busca adquirir experiencia práctica en todas las etapas del ciclo de vida del desarrollo de software y aprender a aplicar metodologías y buenas prácticas de ingeniería de software en un proyecto real.

Capítulo 3

Antecedentes

3.1. Introducción

Este capítulo proporcionará una visión general de los antecedentes que contextualizan el desarrollo del presente Trabajo de Fin de Grado.

En primer lugar, se abordarán aspectos teóricos del análisis sintáctico para comprender mejor el marco conceptual en el que se enmarca este trabajo (Sección 3.2).

A continuación, se llevará a cabo un análisis exhaustivo de la versión anterior de la aplicación SimAS 2.0, la cual se basa en la versión original SimAS 1.0 (Sección 3.3) desarrollada por Vanesa Martínez García [27]. La versión 2.0 fue desarrollada por Juan Antonio Fernández Díaz [5] en 2023. En este análisis se examinarán en detalle las funcionalidades principales de SimAS 2.0, así como los desafíos y limitaciones encontrados en su implementación (Sección 3.4).

Por último, se presentarán las razones y justificaciones que respaldan la realización de este Trabajo de Fin de Grado. Se discutirá la necesidad de mejorar y ampliar las capacidades de la aplicación SimAS para proporcionar una herramienta más eficaz y completa para el aprendizaje del análisis sintáctico en el contexto académico (Sección 3.5).

3.2. Fundamentos teóricos del análisis sintáctico

3.2.1. Tipos de analizadores sintácticos

El análisis sintáctico es una fase muy importante del proceso de traducción o interpretación de un programa escrito en un lenguaje de programación. Existen diferentes tipos de analizadores sintácticos: generales, descendentes y ascendentes. El presente Trabajo de Fin de Grado simulará el funcionamiento de algunos analizadores sintácticos descendentes y ascendentes, que se describen, brevemente, en las secciones 3.2.5 y 3.2.6, respectivamente.

Los analizadores sintácticos están basados en un tipo de gramática formal, deno-

minada *gramática independiente del contexto* o *gramática de contexto libre*, que permite generar muchos aspectos de los lenguajes de programación. Las características de las gramáticas de contexto libre se describen en la sección 3.2.2.

Una descripción más detallada del los fundamentos teóricos del análisis sintáctico se puede consultar en el libro de A. V. Aho et al. [1].

3.2.2. Gramáticas de contexto libre

Definición 3.1 Una *gramática de contexto libre* G se define como

$$G = (V_N, V_T, P, S) \quad (3.1)$$

donde

- V_N es un conjunto finito de símbolos que se denomina **vocabulario** o **alfabeto no terminal** y también puede ser denotado por Σ_N o N .
- V_T es un conjunto finito de símbolos que se denomina **vocabulario** o **alfabeto terminal** y también puede ser denotado por Σ_T o T ,
- $S \in V_N$ y es el **axioma** o **símbolo inicial** o **distinguido** de la gramática y
- P es el **conjunto de reglas de reescritura** o **de producción**.

Se verifica que

$$V_N \cup V_T = V \quad (3.2)$$

$$V_N \cap V_T = \emptyset \quad (3.3)$$

V es el **alfabeto** o **vocabulario** de la gramática y también suele ser denotado por Σ .

El conjunto de producciones P se define como

$$P = \{(A, \beta) | A \in V_N, \beta \in V^* = (V_N \cup V_T)^*\} \quad (3.4)$$

El par $(A\beta)$ suele ser denotado por

$$A \rightarrow \beta \quad (3.5)$$

siendo A la parte izquierda de la regla de producción y β la parte derecha.

Las gramáticas de contexto libre también se denominan *gramáticas independientes del contexto* o *gramáticas de tipo 2* según la jerarquía de Noam Chomsky [1].

Muchas de las sentencias de los lenguajes de programación pueden ser generadas por gramáticas de contexto libre. En el siguiente ejemplo, se muestra una gramática que permite generar sentencias de asignación de expresiones numéricas a un identificador.

Sea la gramática de contexto libre $G = (V_N, V_T, P, S)$ donde

- $V_N = \{S, E\}$
- $V_T = \{\text{identificador}, =, +, *, (,), \text{número}\}$
- $S \in V_N$
- y el conjunto de reglas de producción es:

$$\begin{aligned}
 P &= \{ \\
 (1) \quad &S \rightarrow \text{identificador} = E \\
 (2) \quad &E \rightarrow E + E \\
 (3) \quad &E \rightarrow E * E \\
 (4) \quad &E \rightarrow (E) \\
 (5) \quad &E \rightarrow \text{identificador} \\
 (6) \quad &E \rightarrow \text{número} \\
 \}
 \end{aligned}$$

3.2.3. Derivación con una gramática de contexto libre

Las gramáticas de contexto libre utilizan el concepto de derivación para generar las palabras de una lenguaje de contexto libre. A continuación, se define los conceptos de **derivación inmediata** y **derivación general** o, simplemente, **derivación**.

Definición 3.2 Si $G = (V_N, V_T, P, S)$ es una gramática de contexto libre, $\delta A \eta$ es una cadena de símbolos donde $A \in V_N$ y $\delta, \eta \in V^*$ y $A \rightarrow \beta$ es una regla de producción de la gramática entonces

$$\delta A \eta \xrightarrow[G]{\beta} \delta \beta \eta \quad (3.6)$$

es una **derivación inmediata** realizada una producción de la gramática G .

Una derivación inmediata de una gramática de contexto libre solamente sustituye un símbolo no terminal por una cadena de símbolos que formen la parte derecha de alguna de sus reglas.

Definición 3.3 Una **derivación** es una secuencia de cadenas $\alpha_0, \alpha_1, \dots, \alpha_n$ ($n \geq 0$), de forma que la cadena $\alpha_{i+1} = \delta \beta_i \eta$ se obtiene mediante una derivación inmediata de la cadena $\alpha_i = \delta A_i \eta$ ($0 \leq i \leq n - 1$), es decir,

$$\alpha_i = \delta A_i \eta \xrightarrow[G]{\beta} \delta \beta_i \eta = \alpha_{i+1} \quad (3.7)$$

donde $A_i \in V_N$ y $\delta, \eta, \beta_i \in V^*$.

Encadenando todas las derivaciones inmediatas, se tiene que

$$\alpha_0 \xrightarrow[G]{\beta} \alpha_1 \xrightarrow[G]{\beta} \alpha_2 \xrightarrow[G]{\beta} \dots \xrightarrow[G]{\beta} \alpha_i \xrightarrow[G]{\beta} \alpha_{i+1} \xrightarrow[G]{\beta} \dots \xrightarrow[G]{\beta} \alpha_n \quad (3.8)$$

o, abreviadamente,

$$\alpha_0 \xrightarrow[G]{*} \alpha_n \quad (3.9)$$

indicando que α_0 deriva a α_n en cero o más pasos o derivaciones inmediatas.

- Siempre se verifica que

$$\alpha \xrightarrow[G]{0} \alpha \quad (3.10)$$

- Si $n \geq 1$ entonces

$$\alpha_0 \xrightarrow[G]{+} \alpha_n \quad (3.11)$$

En este caso, α_0 deriva a α_n en uno o más pasos o derivaciones inmediatas.

- Si

$$\alpha \xrightarrow[G]{n} \beta \quad (3.12)$$

entonces se dice que α deriva a β en n pasos o derivaciones inmediatas.

- Una **derivación** es *inicial* si $\alpha_0 = S$.
- Una **derivación** es *terminal* si a partir de α_n no se puede aplicar ninguna regla de la gramática.

La gramática definida en la sección 3.2.2 permite generar la siguiente derivación que corresponde a una sentencia de asignación de una expresión aritmética:

$$\begin{aligned} S &\xrightarrow[1]{} \text{identificador} = E \\ &\xrightarrow[2]{} \text{identificador} = E + E \\ &\xrightarrow[3]{} \text{identificador} = E + E * E \\ &\xrightarrow[6]{} \text{identificador} = E + \text{número} * E \\ &\xrightarrow[5]{} \text{identificador} = E + \text{número} * \text{identificador} \\ &\xrightarrow[5]{} \text{identificador} = \text{identificador} + \text{número} * \text{identificador} \end{aligned}$$

Algunas derivaciones se pueden clasificar a su vez en dos categorías: **derivación por la izquierda** y **derivación por la derecha**.

Definición 3.4 Se dice que una **derivación** $\alpha_0, \alpha_1, \dots, \alpha_n$ ($n \geq 0$), es por la **izquierda** si $\forall i \in \{0, \dots, n-1\} \exists A_i \longrightarrow \beta \in P$ tal que

$$\alpha_i = xA_i\eta \xrightarrow[I]{} x\beta_i\eta = \alpha_{i+1} \quad (3.13)$$

donde $x \in V_T^*$, $A_i \in V_N$ y $\eta, \beta \in V^* = (V_N \cup V_T)^*$

Las derivaciones por la izquierda siempre aplican, en cada paso o derivación inmediata, una regla asociada al símbolo no terminal situado más a la izquierda. La definición de derivación por la derecha es análoga.

El siguiente ejemplo muestra la derivación por la izquierda correspondiente a la derivación general del ejemplo anterior.

$$\begin{aligned}
 S &\stackrel{1}{\Rightarrow} \text{identificador} = E \\
 &\stackrel{2}{\Rightarrow} \text{identificador} = E + E \\
 &\stackrel{5}{\Rightarrow} \text{identificador} = \text{identificador} + E \\
 &\stackrel{3}{\Rightarrow} \text{identificador} = \text{identificador} + E * E \\
 &\stackrel{6}{\Rightarrow} \text{identificador} = \text{identificador} + \text{número} * E \\
 &\stackrel{5}{\Rightarrow} \text{identificador} = \text{identificador} + \text{número} * \text{identificador}
 \end{aligned}$$

3.2.4. Árbol de derivación o árbol sintáctico

El **árbol de derivación** o **árbol sintáctico** asociado a una **derivación** de una gramática de contexto libre es un árbol sintáctico que posee las siguientes características:

1. Los nodos del árbol están etiquetados con símbolos gramaticales (terminales o no terminales) o con la palabra vacía.
2. La raíz del árbol está etiquetada con el símbolo inicial de la gramática.
3. Si un nodo tiene al menos un descendiente, le corresponde como etiqueta un símbolo no terminal.
4. Si tenemos un nodo etiquetado con A y se ha aplicado la regla $A \rightarrow X_1X_2\dots X_k$, donde $X_i \in V \cup \{\epsilon\}$ $\forall i \in \{1, 2, \dots, k\}$, entonces dicho nodo tendrá k descendientes inmediatos (“hijos”) etiquetados con los símbolos X_1, X_2, \dots, X_k .

La figura 3.1 muestra el árbol correspondiente a la derivación del ejemplo de la sección anterior.

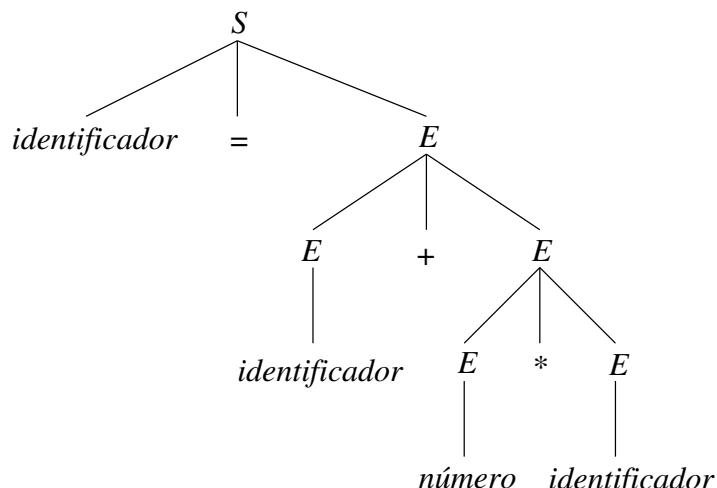


Figura 3.1: Ejemplo de árbol sintáctico asociado a una derivación de una gramática de contexto libre.

3.2.5. Analizadores sintácticos descendentes

El análisis sintáctico descendente comprueba si una gramática de contexto libre puede generar la cadena de símbolos de entrada, como, por ejemplo, una sentencia de un lenguaje de programación, utilizando alguna de las siguientes **estrategias**, que son complementarias:

- Construcción de una **derivación por la izquierda** de la cadena de entrada.
- Construcción de un **árbol sintáctico** de forma **descendente** desde la raíz hasta las hojas.

Considérese la siguiente gramática que genera sentencias de asignación de expresiones aritméticas:

$$\begin{aligned} P &= \{ \\ (1) \quad S &\rightarrow \text{identificador} = E \\ (2) \quad E &\rightarrow T E' \\ (3) \quad E' &\rightarrow + T E' \\ (4) \quad E' &\rightarrow \epsilon \\ (5) \quad T &\rightarrow F T' \\ (6) \quad T' &\rightarrow * F T' \\ (7) \quad T' &\rightarrow \epsilon \\ (8) \quad F &\rightarrow (E) \\ (9) \quad F &\rightarrow \text{identificador} \\ (10) \quad F &\rightarrow \text{número} \\ &\} \end{aligned}$$

Esta gramática permite generar la siguiente derivación por la izquierda.

$$\begin{aligned} S &\stackrel{1}{\Rightarrow} \text{identificador} = E \\ &\stackrel{2}{\Rightarrow} \text{identificador} = T E' \\ &\stackrel{5}{\Rightarrow} \text{identificador} = F T' E' \\ &\stackrel{9}{\Rightarrow} \text{identificador} = \text{identificador} T' E' \\ &\stackrel{7}{\Rightarrow} \text{identificador} = \text{identificador} \epsilon E' \\ &\stackrel{3}{\Rightarrow} \text{identificador} = \text{identificador} \epsilon + T E' \\ &\stackrel{5}{\Rightarrow} \text{identificador} = \text{identificador} \epsilon + F T' E' \\ &\stackrel{10}{\Rightarrow} \text{identificador} = \text{identificador} \epsilon + \text{n} T' E' \\ &\stackrel{6}{\Rightarrow} \text{identificador} = \text{identificador} \epsilon + \text{n} * F T' E' \\ &\stackrel{9}{\Rightarrow} \text{identificador} = \text{identificador} \epsilon + \text{n} * \text{identificador} T' E' \\ &\stackrel{7}{\Rightarrow} \text{identificador} = \text{identificador} \epsilon + \text{n} * \text{identificador} \epsilon E' \\ &\stackrel{4}{\Rightarrow} \text{identificador} = \text{identificador} \epsilon + \text{n} * \text{identificador} \epsilon \underline{\epsilon} \end{aligned}$$

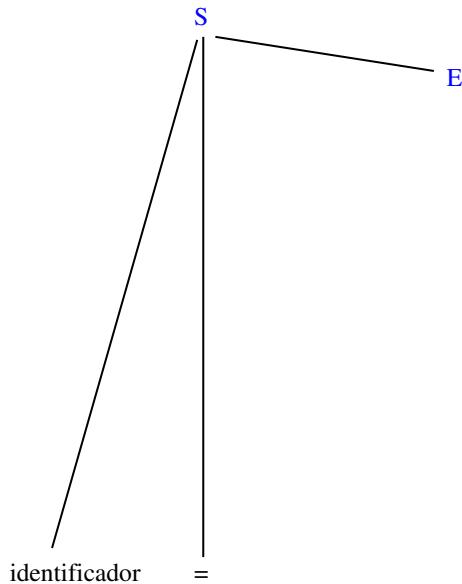


Figura 3.2: Árbol sintáctico: paso inicial del análisis sintáctico descendente predictivo

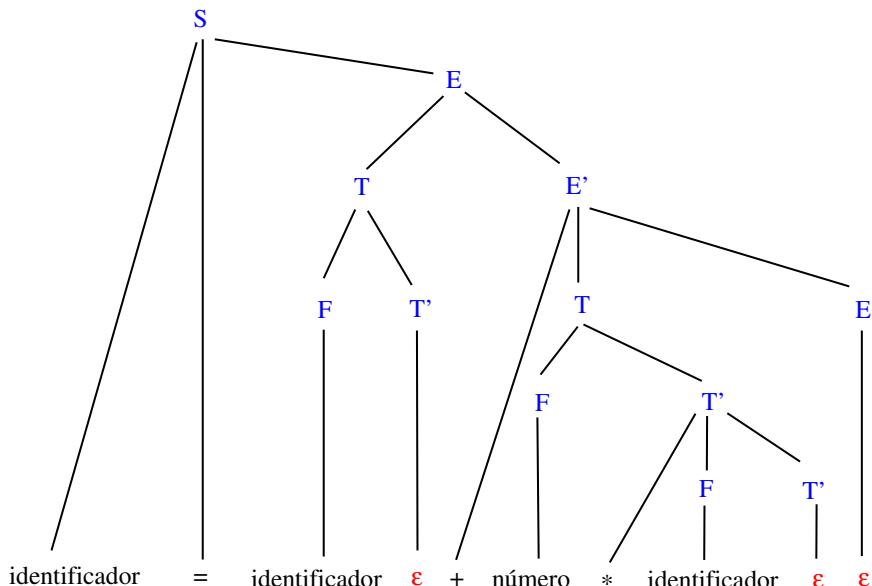


Figura 3.3: Árbol sintáctico: paso final del análisis sintáctico descendente predictivo

Las Figuras 3.2 y 3.3 muestran el paso inicial y final, respectivamente, del árbol sintáctico generado por el análisis descendente. Este árbol invertido ha sido generado de forma descendente desde la raíz hasta las hojas.

Existen diferentes dos tipos de analizadores sintácticos descendentes [1]:

1. Método de descenso recursivo **con** retroceso o *backtracking*.
2. Método de descenso **predictivo**, es decir, **sin** retroceso.

A su vez, existen dos versiones del análisis descendente predictivo:

- Versión recursiva, que utiliza funciones asociadas a los símbolos no terminales de la gramática de contexto libre.

- Versión iterativa, que utiliza una pila para simular el análisis sintáctico descendente.

Ambas versiones del análisis descendente predictivo utilizan una **tabla predictiva** para generar la derivación por la izquierda o el árbol sintáctico de forma descendente. La Tabla 3.1 muestra la Tabla de análisis descendente predictivo de la gramática que genera sentencias de asignación de expresiones aritméticas y que se definió previamente.

Tabla 3.1: Tabla de análisis descendente predictivo

		Símbolo de entrada							
		id	=	+	*	()	número	\$
S	1								
E	2				2			2	
E'			3			4			4
T	5				5			5	
T'			7	6		7			7
F	9				8			10	

El primer símbolo de cada fila es un símbolo no terminal de la gramática. El primer símbolo de cada columna es o un símbolo terminal de la gramática o el símbolo \$, que representa el fin de la cadena de símbolos que se va a analizar en la entrada. El número que aparece en las celdas interiores de la tabla representa el número de la regla de producción de la gramática que se debe utilizar cuando se está derivando un símbolo no terminal y en la entrada aparece un símbolo terminal o el símbolo \$.

La Tabla 3.2 muestra cómo se utiliza la Tabla 3.1 para realizar el análisis descendente predictivo de la siguiente sentencia: **id = id + n * id \$**. La columna **Acción** de dicha tabla indica las reglas de producción que se deben utilizar para generar la derivación por la izquierda y el árbol sintáctico de forma descendente. Véanse las Figuras 3.2 y 3.3.

3.2.6. Analizadores sintácticos ascendentes

El análisis sintáctico ascendente comprueba si una gramática de contexto libre puede generar la cadena de símbolos de entrada, como, por ejemplo, una sentencia de un lenguaje de programación, utilizando alguna de las siguientes **estrategias**, que son complementarias:

- Construcción de una **derivación por la derecha en orden inverso** de la cadena de entrada.
- Construcción de un **árbol sintáctico** de forma **ascendente** desde las hojas hasta la raíz.

Se han propuesto diferentes métodos de análisis sintáctico ascendente [1]:

- Métodos basados en la precedencia
 - Métodos de precedencia simple.

Tabla 3.2: Pasos del análisis descendente predictivo

Pila	Entrada	Acción
\$ S	id = id + n * id \$	1) S → identificador = E
\$ <u>E = id</u>	id = id + n * id \$	Emparejar
\$ E =	= id + n * id \$	Emparejar
\$ E	id + n * id \$	2) E → T E'
\$ <u>E' T</u>	id + n * id \$	5) T → F T'
\$ <u>E' T' F</u>	id + n * id \$	9) F → identificador
\$ E' T' <u>id</u>	id + n * id \$	Emparejar
\$ E' T'	+ n * id \$	7) T' → ε
\$ E'	+ n * id \$	3) E' → + T E'
\$ E' T' +	+ n * id \$	Emparejar
\$ E' T	n * id \$	
\$ E' <u>T' F</u>	n * id \$	10) F → número
\$ E' T' <u>n</u>	n * id \$	Emparejar
\$ E' T'	* id \$	6) T' → * F T'
\$ E' <u>T' F *</u>	* id \$	Emparejar
\$ E' T' F	id \$	9) F → identificador
\$ E' T' <u>id</u>	id \$	Emparejar
\$ E' T'	\$	7) T' → ε
\$ E'	\$	4) E' → ε
\$	\$	Aceptar

- Métodos de precedencia débil.
 - Métodos de precedencia extendida.
 - Métodos de precedencia de estrategia mixta.
 - Métodos de precedencia de operadores.
- Métodos de análisis LR
- Método SLR
 - Método LR - canónico
 - Método LALR

A continuación se muestra un ejemplo de aplicación del método de análisis sintáctico ascendente SLR. Considérese la siguiente gramática de contexto libre que permite generar prototipos de funciones:

$$\begin{aligned}
 P = & \{ \\
 (1) \quad & S \rightarrow T \text{ id} (L) ; \\
 (2) \quad & T \rightarrow T * \\
 (3) \quad & T \rightarrow \text{int} \\
 (4) \quad & L \rightarrow L , T \\
 (5) \quad & L \rightarrow T
 \end{aligned}$$

}

A partir de esta gramática, se genera la Tabla de análisis sintáctico SLR que se muestra en la Tabla 3.3, donde $d\ n$ indica que se desplaza un símbolo de la entrada a la pila y se pasa al estado n ; por su parte, $r\ k$ representa que se utiliza la regla número k de la gramática para reducir el contenido de la cima de la pila al símbolo no terminal de la parte izquierda de la regla k . Véase la Tabla 3.4. Una descripción más detallada de este método SLR y de los demás métodos LR se puede consultar en el libro de Aho et al. [1].

Tabla 3.3: Tabla de análisis sintáctico SLR

Acción									Ir-a		
	id	()	;	*	int	,	\$	S	T	L
0						d 3			1	2	
1								Aceptar			
2	d 4				d 5						
3	r 3		r 3		r 3		r 3				
4		d 6									
5	r 2		r 2		r 2		r 2				
6					d 3				8	7	
7			d 9				d 10				
8			r 5		d 5		r 5				
9				d 11							
10					d 3				12		
11								r 1			
12			r 4		d 5		r 4				

Tabla 3.4: Pasos del análisis sintáctico ascendente SLR

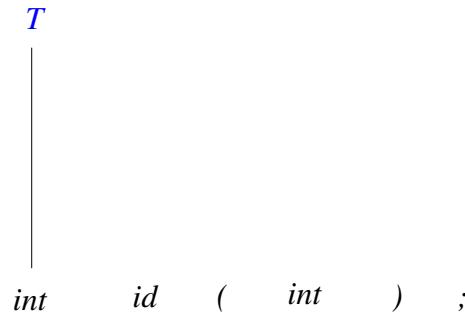
Pila	Entrada	Acción
0	int id (int) ; \$	desplazar 3
0 <u>int</u> 3	id (int) ; \$	reducir 3) T → int
0 T 2	id (int) ; \$	desplazar 4
0 T 2 id 4	(int) ; \$	desplazar 6
0 T 2 id 4 (6	int) ; \$	desplazar 3
0 T 2 id 4 (6 <u>int</u> 3) ; \$	reducir 3) T → int
0 T 2 id 4 (6 <u>int</u> 3) ; \$	reducir 3) T → int
0 T 2 id 4 (6 <u>T</u> 8) ; \$	reducir 5) L → T
0 T 2 id 4 (6 L 7) ; \$	desplazar 9
0 T 2 id 4 (6 L 7) 9	; \$	desplazar 11
0 T 2 id 4 (6 L 7) 9 ; 11	\$	reducir 1) S → T id (L) ;
0 S 1	\$	Aceptar

Usando la Tabla 3.3, se pueden realizar los pasos del análisis sintáctico ascendente que se muestran en la Tabla 3.4 y que permiten generar la derivación por la derecha de que se indica a continuación:

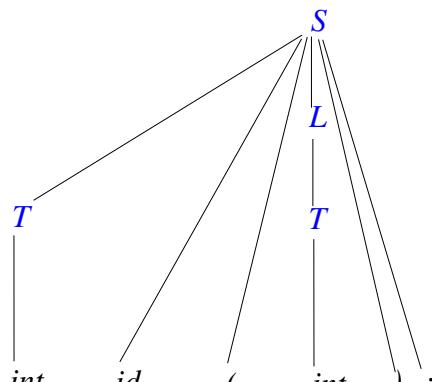
3.2. FUNDAMENTOS TEÓRICOS DEL ANÁLISIS SINTÁCTICO

$$\begin{array}{lcl}
 S & \xrightarrow{1} & T \text{ id } (\text{ L }) ; \\
 & \xrightarrow{5} & T \text{ id } (\underline{T}) ; \\
 & \xrightarrow{3} & T \text{ id } (\underline{\text{int}}) ; \\
 & \xrightarrow{3} & \underline{\text{int}} \text{ id } (\text{ int }) ;
 \end{array}$$

La Figura 3.4 muestra el paso inicial y final de la generación del árbol sintáctico (invertido) construido de forma ascendente desde las hojas hasta la raíz.



(a)



(b)

Figura 3.4: Árbol sintáctico: pasos (a) inicial y (b) final del análisis sintáctico ascendente SLR

3.3. SimAS 1.0

Este Trabajo de Fin de Grado se centra en el desarrollo de un simulador de analizadores sintácticos descendentes, habiendo existido una primera versión previa denominada SimAs 1.0 [27], la cual fue posteriormente mejorada. En esta sección, se describen las características principales de dicha versión inicial, la cual servirá como referencia para el desarrollo de la nueva versión.

SimAs 1.0. es una aplicación de escritorio y multiplataforma desarrollada en 2015 con los siguientes recursos de software:

- Sistemas operativos: Ubuntu Linux (versión 12.10) [4] y Microsoft Windows 7 [17].
- Lenguaje de programación: Java SE (JDK) 7u40 [19].
- Entorno de desarrollo: NetBeans 7.3.1. [2].
- Biblioteca para la interfaz gráfica: Java Swing [20].
- Biblioteca iText 5.5.0 de Java para la generación de informes en formato pdf [12].
- XML [30]: formato utilizado para almacenar las gramáticas formales.

En las siguientes figuras, se describen algunas de las características más importantes del funcionamiento de la versión 1.0 de SimAs.

La figura 3.5 muestra la interfaz del editor de SimAS 1.0. En dicha figura, se han numerado algunos componentes que son descritos a continuación:

1. Se muestra la barra de menús donde se puede escoger entre opciones: editor, simulador o ayuda.
2. La barra de herramientas permite acceder rápidamente a las opciones más frecuentes, como crear abrir o guardar una gramática, verificarla o incluso escoger entre si se prefiere usar el análisis ascendente o descendente.
3. El editor está compuesto por diferentes formularios para indicar el nombre de la gramática, añadir una pequeña descripción, sus símbolos terminales y no terminales, establecer el símbolo inicial y definir sus reglas de producción

La figura 3.6 muestra cómo se pueden asignar funciones de tratamiento de errores en el método SLR de análisis sintáctico ascendente. Los componentes numerados se describen a continuación:

1. Se indica el tipo de simulación que se llevará a cabo.
2. Se muestra la gramática de contexto libre que se va a utilizar.
3. Funciones de tratamiento de errores que se pueden utilizar en la simulación del análisis sintáctico.
4. Tabla de análisis LR

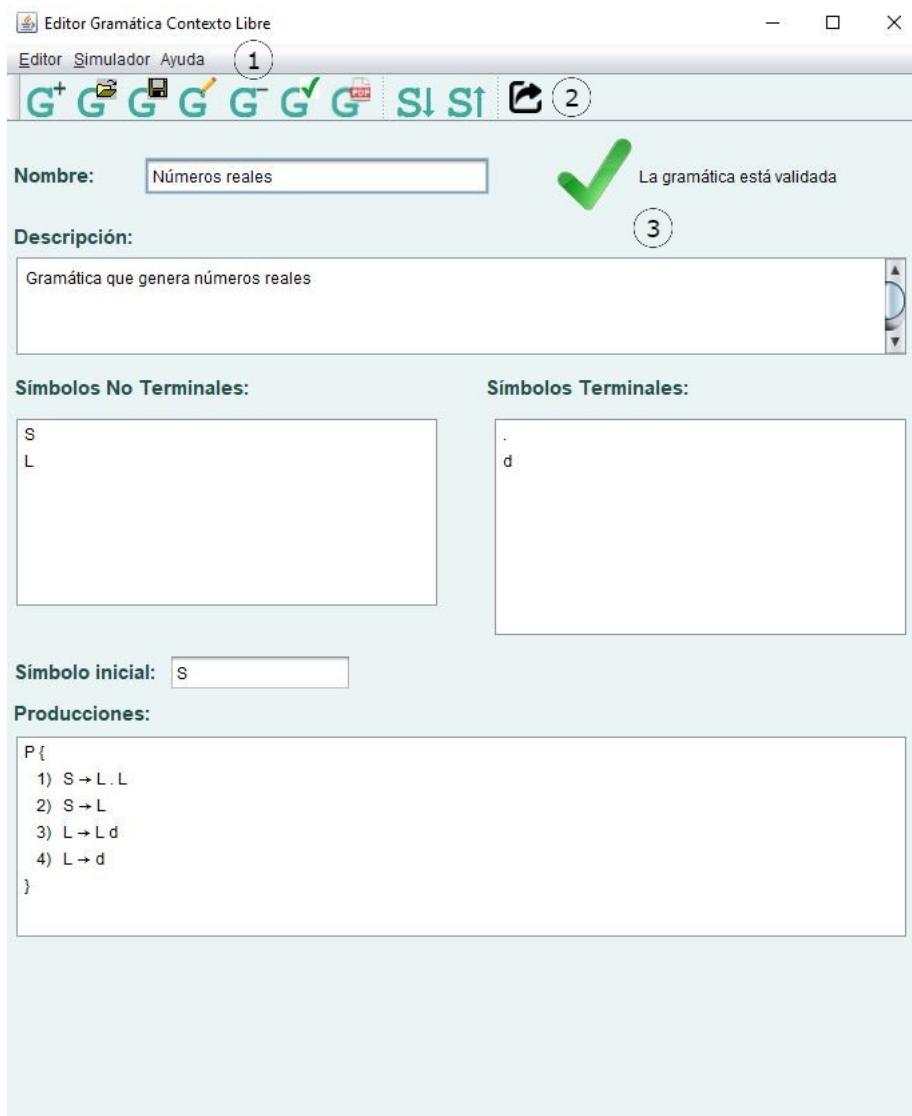


Figura 3.5: Interfaz del editor de SimAS 1.0.

Por último, la figura 3.7 muestra la ventana para la simulación del método SLR de análisis sintáctico ascendente:

1. Se indica el tipo de simulación que se va a realizar: simulación ascendente SLR.
2. Formulario para introducir y editar una cadena de entrada para poder realizar la simulación.
3. Se muestran una serie de flechas que permiten reproducir la simulación hacia adelante, hacia atrás, etc.
4. En este último apartado, se muestran los resultados de la simulación.

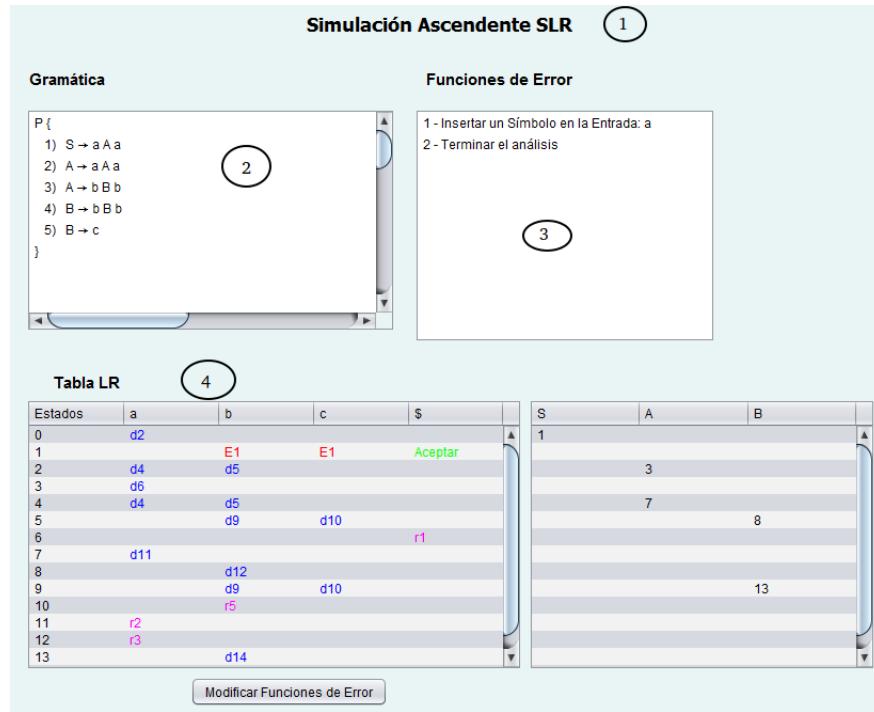


Figura 3.6: Panel del editor de SimAS 1.0



Figura 3.7: Ventana del simulador de SimAS 1.0.

3.4. SimAS 2.0

SimAS 2.0 [5] se desarrolló para corregir y ampliar la versión de SimAS 1.0 [27]. SimAS 1.0 se instaló en diferentes equipos y se comprobó que permitía simular el funcionamiento de los analizadores sintácticos descendentes y ascendentes. Sin embargo, se detectaron algunos fallos y limitaciones que se intentaron corregir con mejoras que se enumeran a continuación:

1. Mejorar el almacenamiento de las gramáticas de contexto libre en formato XML.
 - Procesar de forma individualizada los símbolos de la parte derecha de una regla de producción.
 - Al cargar una gramática, respetar el orden de sus reglas de producción. No se respeta si ya hay otra gramática previamente cargada.
2. Validar la gramática cuando se cargue o se modifique, es decir, comprobar que no tiene símbolos y reglas inútiles. En la versión anterior, se debía pulsar un botón para validar.
3. Al eliminar la recursividad por la izquierda o factorizar por la izquierda, permitir que se pueda conservar la gramática original y la nueva gramática. Además, también debe permitir la enumeración de las reglas de producción de la nueva gramática.
4. Permitir que la gramática que se está procesando se pueda mostrar o consultar en cualquier momento de forma simultánea en cualquier paso del proceso de simulación de los analizadores sintácticos.
5. Al construir los conjuntos Primero y Siguiente, mostrar las dependencias entre los conjuntos.
6. Simular correctamente las gramáticas con reglas épsilon. En particular, calcular correctamente el conjunto Primero en gramáticas reglas épsilon.
7. Crear funciones predefinidas de recuperación de errores en las simulaciones de los analizadores sintácticos.
8. Permitir que se puedan completar las celdas vacías de la tabla de análisis sintáctico descendente con reglas épsilon, cuando proceda, sin tener que predefinir funciones de error.
9. Permitir que se puedan completar las celdas vacías de la tabla de análisis sintáctico ascendente con reducciones, cuando proceda, sin tener que predefinir funciones de error.
10. Mostrar las derivaciones obtenidas por los analizadores sintácticos.
11. Permitir la generación de informes en formato pdf. La aplicación SimAS podía generar estos informes, pero la actualización del sistema operativo no lo permite ahora.

Además, se intentó que la nueva versión SimAS 2.0 permitiese la generación “paso a paso” de los árboles sintácticos de las derivaciones generadas al simular los analizadores sintácticos descendentes o ascendentes.

En su momento, se consideró que el desarrollo de la versión 2.0 estaba justificado por las siguientes razones:

- La nueva versión del simulador SimAS 2.0 sería de gran ayuda para que los estudiantes pudiesen aprender “paso a paso” el funcionamiento de los analizadores sintácticos descendentes y ascendentes.
- La corrección de los errores de SimAS 1.0 evitará que los estudiantes puedan cometerlos.
- La inclusión de la generación de los árboles sintácticos asociados a las derivaciones permitiría que los estudiantes comprendan mejor el funcionamiento de los analizadores sintácticos.

En las siguientes figuras, se describen algunas de las características más importantes del funcionamiento de la versión 2.0 de SimAS.

La figura 3.8 muestra la interfaz del editor de SimAS 2.0, la cual contiene pocas variaciones respecto a la versión original. En dicha figura, se han numerado algunos componentes que son descritos a continuación:

1. La interfaz presenta una barra de menús que brinda opciones como editor, simulador, ayuda y un tutorial detallado para comprender su funcionamiento.
2. Además, la barra de herramientas facilita el acceso a las funciones más utilizadas, como la creación, apertura y guardado de gramáticas, así como la verificación de las mismas. También permite elegir entre el análisis ascendente y el descendente.
3. En cuanto al editor, se compone de distintos formularios que permiten especificar el nombre de la gramática, agregar una breve descripción, definir los símbolos terminales y no terminales, establecer el símbolo inicial y definir las reglas de producción.

La figura 3.9 muestra el primer paso a realizar para el análisis de gramáticas en SimAS 2.0, que consiste en la eliminación de recursividad de la gramática original. En dicha figura, se han numerado algunos componentes que son descritos a continuación:

1. En primer lugar, aparece un mensaje indicativo con los cambios realizados a la gramática original.
2. En segundo lugar, se muestra la nueva gramática generada tras realizar los cambios oportunos a la gramática original.
3. Además, nos da la opción de consultar la gramática original para poder comparar ambas gramáticas.

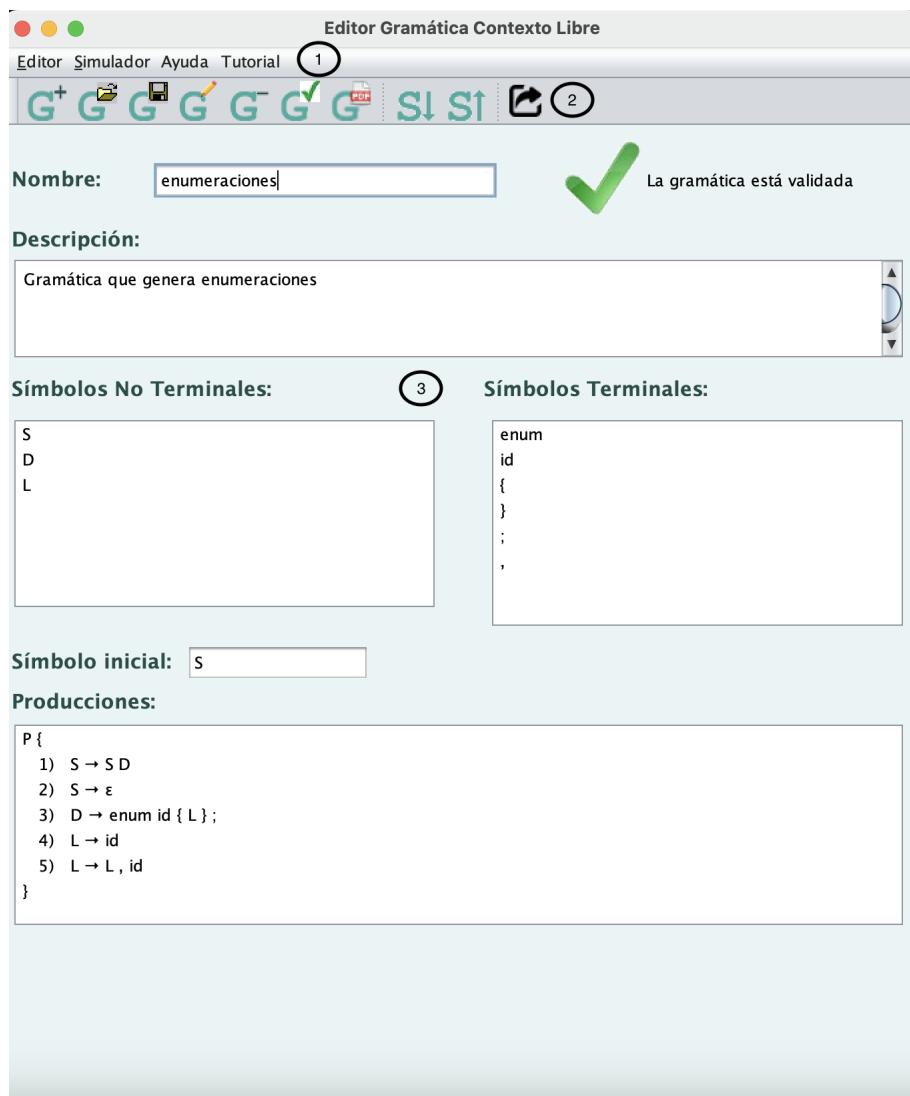


Figura 3.8: Interfaz del editor de SimAS 2.0.

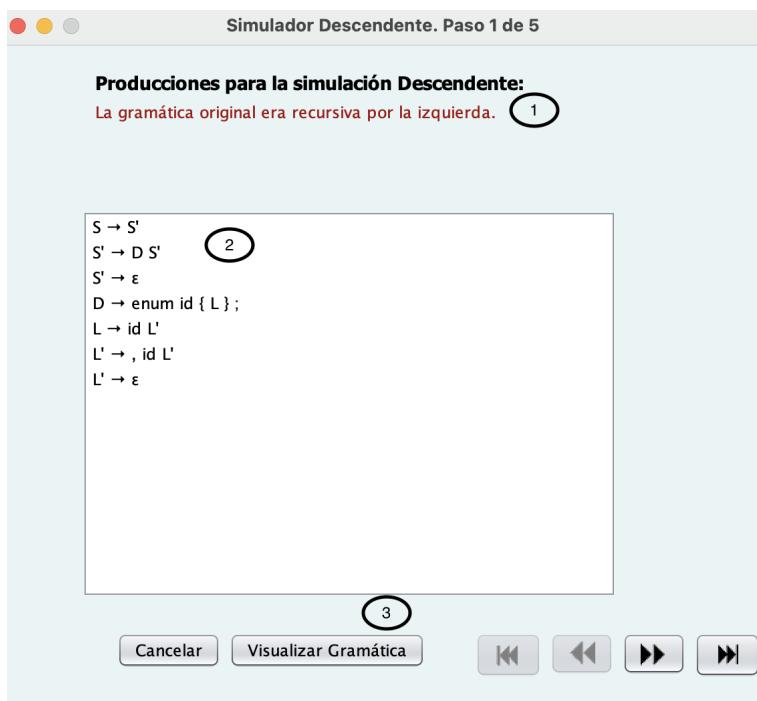


Figura 3.9: Corrección de gramáticas de SimAS 2.0.

The screenshot shows the 'Simulador Descendente. Paso 2 de 5' window titled 'Conjunto Primero y Siguiente'. Circled number 1 is above the table header. The table has three columns: 'Símbolo', 'Conjunto primero', and 'Conjunto siguiente'. The data is as follows:

Símbolo	Conjunto primero	Conjunto siguiente
S	ε enum	\$
D	enum	enum \$
L	id	}
S'	ε enum	\$
L'	, ε	}

At the bottom, circled number 2 is next to the 'Visualizar Gramática' button. There are also navigation buttons: 'Cancelar', 'Visualizar Gramática', and four arrows pointing left, right, or both.

Figura 3.10: Conjunto Primero y Siguiente de SimAS 2.0.

La figura 3.10 muestra los conjuntos Primero y Siguiente generados para el análisis de gramáticas en SimAS 2.0. En dicha figura, se han numerado algunos componentes que son descritos a continuación:

1. El usuario puede ver una tabla con los conjuntos Primero y Siguiente correspondientes a cada símbolo no terminal.
2. También tenemos un menú común a todos los pasos, que anteriormente no se había mencionado, que permite avanzar paso a paso o avanzar directamente al último paso.

	enum	id	{	}	;	,	\$
S	1						
D		4					
L			5				
S'	2						3
L'					6		

Figura 3.11: Tabla predictiva de SimAS 2.0.

La figura 3.11 muestra la tabla predictiva generada por el análisis de gramáticas en SimAS 2.0. En dicha figura, se han numerado algunos componentes que son descritos a continuación:

1. En primer lugar, la interfaz contiene la tabla predictiva generada por el programa para la gramática.
2. Al igual que en el resto de pasos intermedios, el programa contiene la posibilidad de visualizar la gramática original, y el menú de avance y retroceso.

La figura 3.12 muestra las funciones de error predefinidas o que podemos definir respecto para las gramáticas en SimAS 2.0. En dicha figura, se han numerado algunos componentes que son descritos a continuación:

1. En primer lugar, podemos ver las gramáticas predefinidas que contiene el programa, que comparando con la versión anterior, vemos que son muchas más.

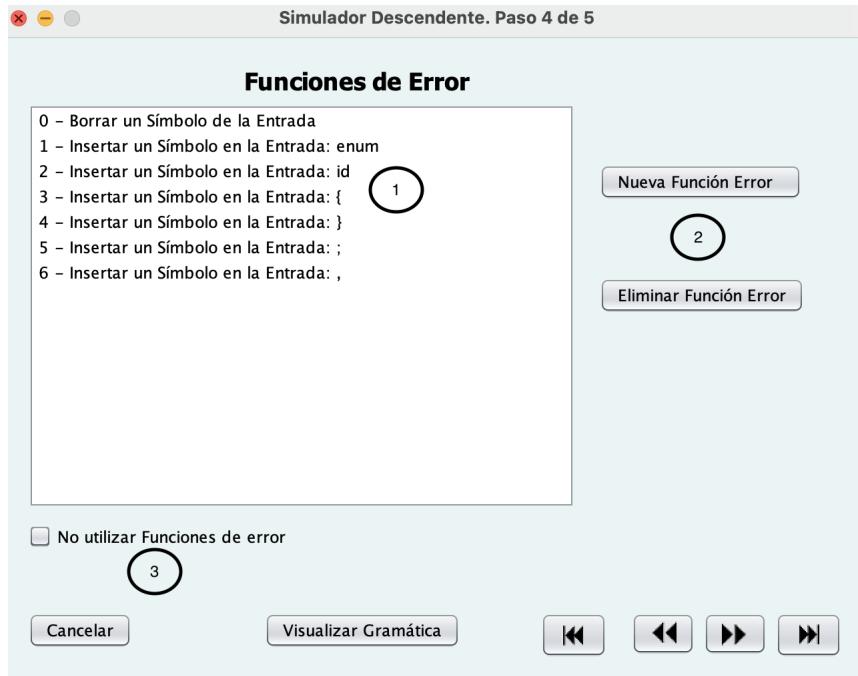


Figura 3.12: Tabla predictiva de SimAS 2.0.

2. Además, un cambio importante respecto a la versión anterior, es que ésta da la posibilidad de añadir o eliminar funciones de error directamente desde esta interfaz.
3. Otro cambio significativo respecto a la versión anterior es que se da la opción de no usar funciones de error, lo que hace que el programa salte directamente a la parte final, sin pasar por el paso 5.

La figura 3.13 muestra las funciones de error predefinidas o que podemos definir respecto para las gramáticas en SimAS 2.0. En dicha figura, se han numerado algunos componentes que son descritos a continuación:

1. En primer lugar, se ofrece un menú con las funciones de error definidas en el paso anterior.
2. En segundo lugar, se tiene la tabla predictiva aumentada para que el usuario añada dichas funciones de forma intuitiva y muy rápida.
3. Se añade la opción de eliminar alguna función de error que el usuario estime como incorrecta.
4. Finalmente, el programa da la opción de llenar la tabla con las producciones épsilon que pertenezcan a dicha gramática.

Simulador Descendente. Paso 5 de 5

Incluir Funciones de Error

Pulse en la tabla para insertar la función de error:

3 – Insertar un Símbolo en la Entrada: {

	enum	id	{	}	;	,	\$
S	1						
D	4						
L		5					
S'	2						3
L'	E0			7		6	
enum	Emparejar						
id		Emparejar					
{	E3		Emparejar	E5			
}		E0		Emparejar			
;				Emparejar			
,					Emparejar		
\$			E0			Aceptar	
enum	Emparejar						
id		Emparejar					
{			Emparejar				
}				Emparejar			
;					Emparejar		
,						Emparejar	
\$						Aceptar	

Visualizar Gramática Eliminar Función de Error Rellenar con producciones épsilon

Cancelar << >> Finalizar

Figura 3.13: Tabla predictiva con funciones de errores de SimAS 2.0.

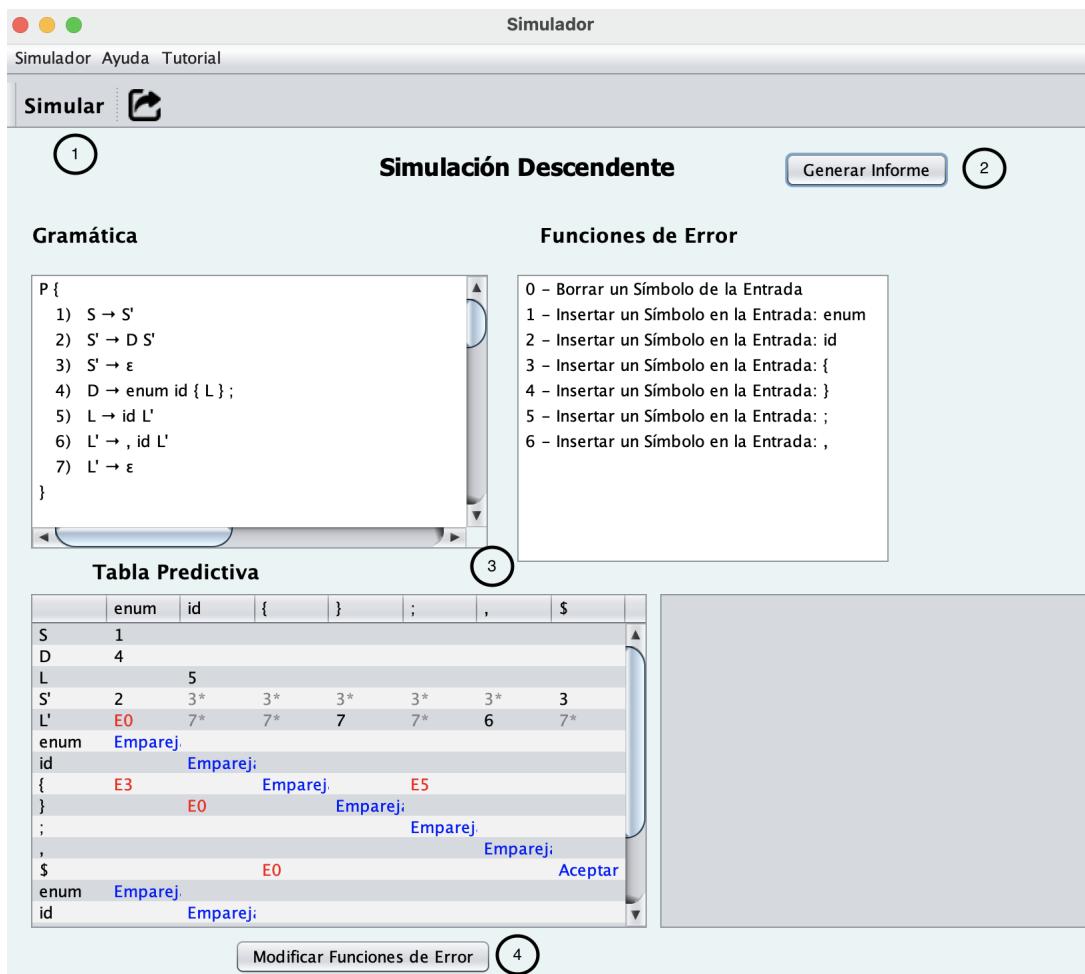


Figura 3.14: Tabla predictiva con funciones de errores de SimAS 2.0.

La figura 3.14 muestra todo lo obtenido anteriormente respecto al análisis de las gramáticas en SimAS 2.0. En dicha figura, se han numerado algunos componentes que son descritos a continuación:

1. En primer lugar, nos da la posibilidad de realizar una simulación introduciendo algunos elementos de la gramática.
2. En segundo lugar, se ofrece la posibilidad de generar un informe con todos los datos obtenidos en el análisis de la gramática.
3. A continuación, se muestra un resumen con toda la información obtenida hasta el momento, como lo es la gramática sin recursividad, las funciones de error usadas y la tabla predictiva ya completa.
4. Además, se tiene la opción de modificar las funciones de error que se han elegido con anterioridad.

3.4.1. Fallos generales identificados en SimAS 2.0

Durante el análisis y uso de la aplicación SimAS 2.0, se han identificado varios fallos y limitaciones que afectan a su funcionamiento y su utilidad como herramienta didáctica. A continuación, se detallan los fallos más relevantes:

1. **Problemas en la persistencia de gramáticas:** al crear o modificar gramáticas en SimAS 2.0, se han observado ocasiones en las que los cambios realizados no se guardan correctamente en el sistema. Esto puede resultar en la pérdida de datos y en la necesidad de rehacer el trabajo. Por otro lado, cuando las gramáticas se guardan correctamente, la interpretación de los símbolos epsilon no se realiza de manera adecuada en algunos casos, lo que puede generar confusiones y errores en el análisis sintáctico.
2. **Limitación en la generación de informes:** la funcionalidad de generación de informes presenta una limitación significativa, ya que solo se realiza correctamente en sistemas operativos Windows. En otros sistemas operativos, como Linux o macOS, la generación de informes puede fallar, lo que afecta la portabilidad y la usabilidad de la aplicación.
3. **Problemas al añadir errores:** se ha observado que, en ciertas situaciones, al intentar agregar errores, la aplicación no permite su inserción en la parte de la tabla correspondiente, normalmente en la parte inferior. Este problema afecta la capacidad de los usuarios para simular y corregir errores sintácticos en gramáticas definidas.
4. **Complejidad en la gestión de producciones gramaticales:** la gestión de producciones gramaticales, como añadir, modificar o eliminar producciones, presenta una complejidad innecesaria. Los usuarios encuentran dificultades al realizar estas operaciones, lo que afecta la eficiencia y la experiencia de uso de la aplicación.

5. **Error ocasional en la generación de conjuntos Primero y Siguiente:** se han observado fallos intermitentes en el proceso de generación de conjuntos Primero y Siguiente. Estos conjuntos son fundamentales para varios algoritmos de análisis sintáctico, por lo que los errores en su cálculo pueden afectar a la precisión y la fiabilidad de la simulación.
6. **Generación parcial de árboles de análisis:** actualmente, SimAS 2.0 no cuenta con la capacidad total de generar árboles sintácticos que representen las derivaciones realizadas durante el análisis sintáctico. El programa genera correctamente los árboles descendentes, excepto los nodos con el símbolo épsilon, y parcialmente los ascendentes en el sistema operativo Windows, pero, en otros sistemas operativos, no los genera bien. Esta funcionalidad es crucial para la comprensión visual del proceso de análisis y su ausencia limita el potencial educativo de la aplicación.

La corrección y mejora de estos fallos son aspectos prioritarios para garantizar el correcto funcionamiento y la utilidad pedagógica de SimAS 2.0 como herramienta de aprendizaje en el ámbito del análisis sintáctico de lenguajes formales.

En las siguientes secciones, se profundizará con más detalle en los fallos encontrados para cada sistema operativo en el que se ha probado, ya que dependiendo de éste, la aplicación actúa de distinta forma.

Se va a utilizar la siguiente gramática de contexto libre para describir los fallos detectados en la aplicación SimAS 2.0:

$$\begin{aligned} P = \{ & \\ & S \rightarrow SD \\ & S \rightarrow \varepsilon \\ & D \rightarrow TL; \\ & T \rightarrow \text{REAL} \\ & T \rightarrow \text{ENTERO} \\ & L \rightarrow L, \text{ID} \\ & L \rightarrow \text{ID} \\ & \} \end{aligned}$$

Esta es una gramática que genera declaraciones de variables tipo entero y real.

3.4.2. Fallos encontrados en Windows

Durante las pruebas realizadas en el sistema operativo Windows 11, se han identificado varios fallos en el funcionamiento de la aplicación SimAS. Uno de los problemas más destacados se refiere a la modificación de la gramática, tanto en los símbolos no terminales como en los símbolos terminales. Los usuarios encuentran dificultades al realizar operaciones como quitar o poner símbolos, así como al intentar renombrar un símbolo.

Además, se observaron inconsistencias en la validación de la gramática, lo que puede llevar a errores en la definición de la misma. Otro problema identificado está relacionado

con la funcionalidad de guardar la gramática en el directorio correspondiente. En algunas ocasiones, la gramática no se guarda correctamente, lo que puede resultar en la pérdida de datos importantes para el usuario.

Durante las pruebas, también se identificaron fallos en la generación de los conjuntos Primero y Siguiente, que no se calculan de manera precisa. Esto puede afectar a la precisión de los análisis sintácticos realizados con la aplicación.

Además de los problemas mencionados anteriormente, se han detectado errores relacionados con la paginación del informe generado por la aplicación. La generación del árbol sintáctico para el símbolo épsilon también presenta problemas en algunas situaciones específicas.

Por último, se observó que al volver a acceder a la aplicación después de salir de la simulación descendente, los datos de la simulación anterior persisten de manera inesperada. Esto puede causar confusiones al usuario y afectar a la integridad de los resultados obtenidos.

3.4.3. Fallos encontrados en Ubuntu

En el entorno de Ubuntu 22, se han identificado varios problemas que afectan a la experiencia y funcionalidad de la aplicación SimAS.

Al eliminar un símbolo de la gramática y posteriormente intentar volver a añadirlo, se observa que es necesario cerrar la ventana emergente correspondiente y abrir una nueva para llevar a cabo la acción deseada. Esta situación puede resultar frustrante para el usuario, ya que interrumpe el flujo de trabajo y genera una experiencia poco fluida en la edición de la gramática.

Otro inconveniente detectado radica en el comportamiento al hacer clic repetidamente en la opción “Visualizar Gramática Original”. En lugar de abrir la ventana una sola vez, se observa que la aplicación abre múltiples instancias de la ventana, lo cual puede resultar confuso para el usuario y generar una sensación de ineficiencia en la navegación por la interfaz, como se puede observar en la imagen 3.15.

Además, se ha detectado que la aplicación no enumera las reglas al eliminar la recursividad en la gramática. Esta omisión puede dificultar la identificación y seguimiento de los cambios realizados en la estructura de la gramática, lo que puede conducir a errores o malentendidos durante el proceso de edición.

Otro problema importante afecta a la generación de los conjuntos Primero y Siguiente, los cuales no se calculan correctamente. Estos conjuntos son fundamentales para varios algoritmos de análisis sintáctico, por lo que errores en su cálculo pueden impactar negativamente en la precisión y fiabilidad de la simulación y validación de la gramática.

En relación con las funciones de error, se ha observado que la función de error para terminar el análisis no está predefinida, lo que puede dificultar la gestión de errores durante el análisis sintáctico y generar resultados inesperados.

Asimismo, al añadir una nueva regla de error, la aplicación no asigna automáticamente el siguiente número disponible, lo que puede generar confusiones en la gestión de

CAPÍTULO 3. ANTECEDENTES

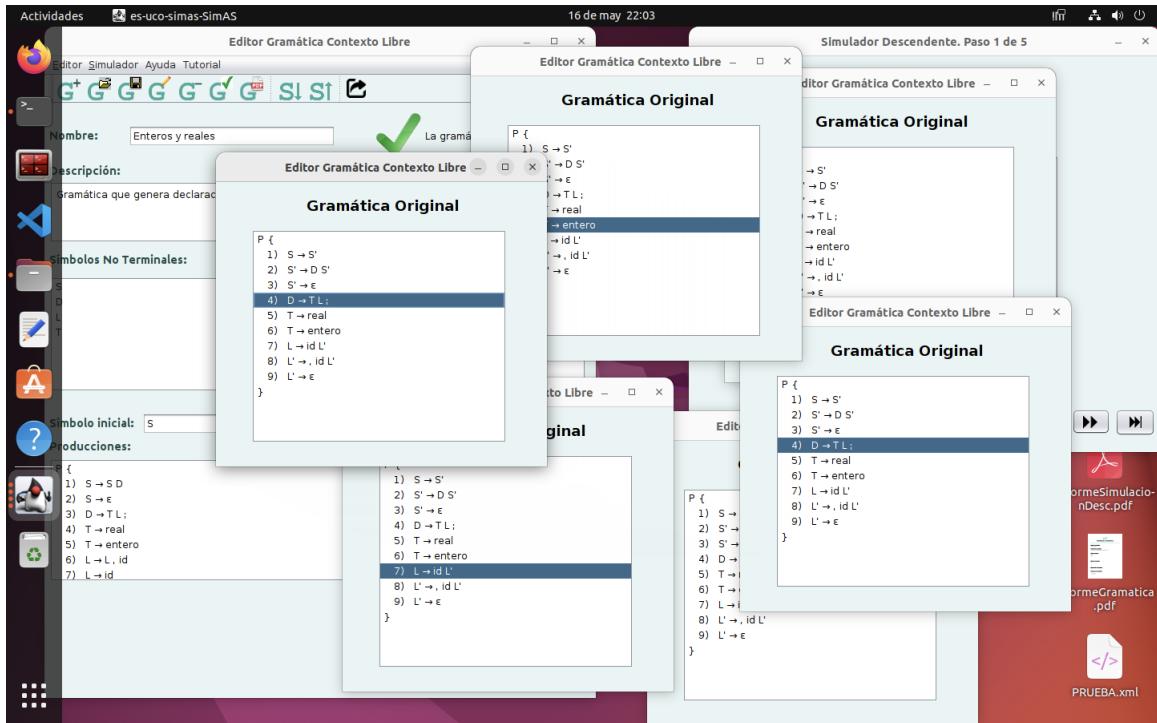


Figura 3.15: Generación repetitiva de la visualización de la gramática

las reglas de error y dificultar su identificación. En la figura 3.16 se observa la posibilidad de añadir cualquier número a las nuevas funciones de error, sin seguir ningún orden.

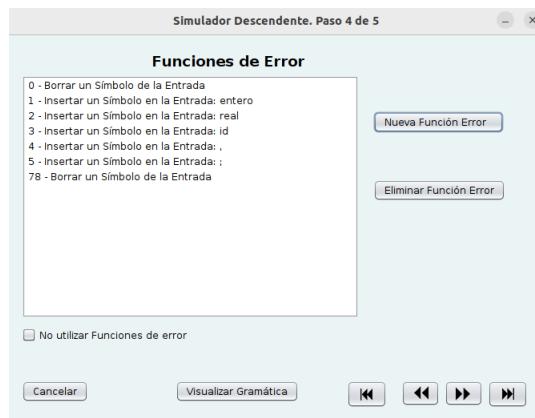


Figura 3.16: Creación de una nueva función de error

Otro problema destacable ocurre al intentar modificar las funciones de error después de haber finalizado y regresado al proceso. En esta situación, la aplicación duplica las filas de los símbolos terminales y modifica los estados de la tabla predictiva, lo que puede causar errores en la definición y gestión de las funciones de error.

La generación incorrecta de informes de simulación descendente y la ausencia de generación de informes tras realizar un análisis de una cadena de entrada también se han identificado como fallos significativos en el funcionamiento de la aplicación en el entorno de Ubuntu 22. Estos problemas afectan negativamente a la capacidad del usuario para

evaluar y analizar los resultados del análisis sintáctico, lo que impacta en la utilidad y fiabilidad de la aplicación en este sistema operativo.

3.4.4. Fallos encontrados en MacOS

Al crear una nueva gramática, el programa no proporciona una opción para regresar al menú principal, lo que dificulta la navegación dentro de la aplicación. Además, al validar una gramática, se abren dos ventanas emergentes, una con la gramática validada y otra que queda congelada, lo que puede resultar confuso para el usuario, observable en la figura 3.17. Por otro lado, al intentar copiar y pegar texto al agregar o modificar símbolos de la gramática, así como al cambiar el nombre de los símbolos, la funcionalidad no está habilitada, lo que limita la capacidad del usuario para editar la gramática de manera eficiente.

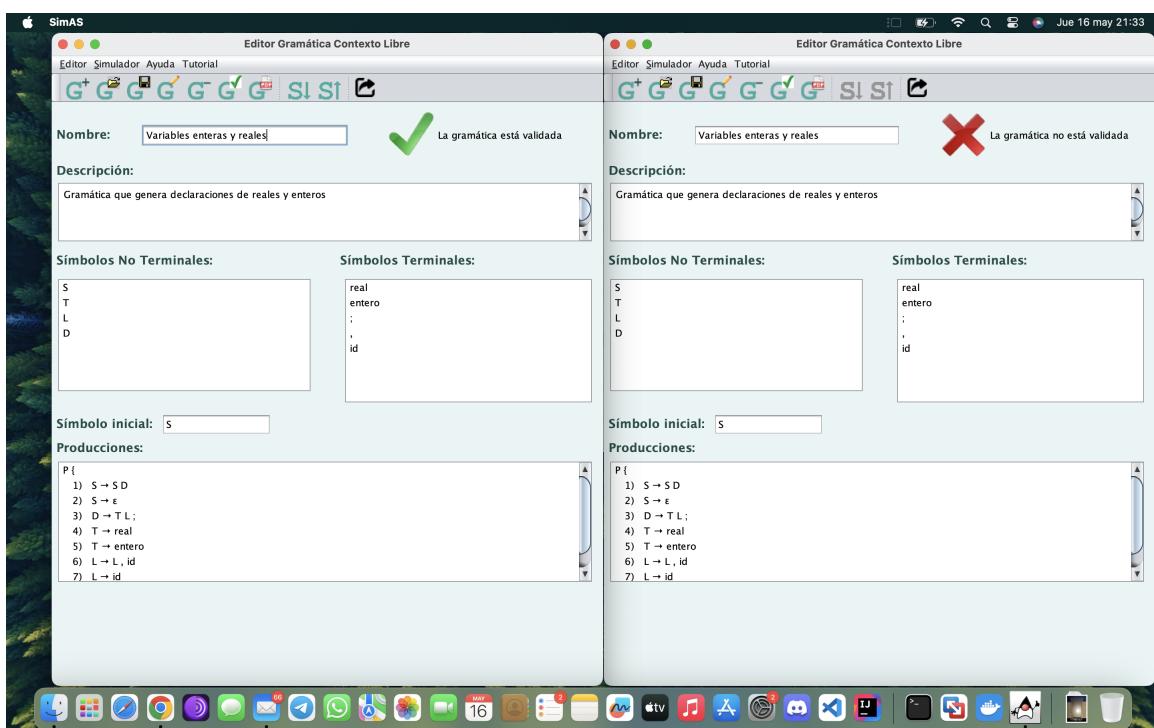


Figura 3.17: Abrir gramática en MacOS.

La aplicación no permite numerar las reglas de la gramática, lo que dificulta la identificación y referencia de las reglas durante el proceso de desarrollo. Asimismo, la imposibilidad de cambiar el orden de las reglas gramaticales puede causar problemas al corregir errores o reorganizar la estructura de la gramática.

En ciertas situaciones, el programa no incluye automáticamente la función de terminar el análisis en la tabla de funciones de error, lo que puede afectar la experiencia del usuario al simular la gramática. Además, al agregar una nueva regla de error, no se asigna automáticamente el siguiente número disponible, lo que puede resultar en la duplicación de números de reglas y generar confusión en la identificación de errores.

El informe generado por la aplicación no se visualiza correctamente en macOS, ya

que al intentar abrirlo aparece un error al cargar el documento PDF. Este problema afecta la capacidad del usuario para revisar y analizar los resultados de la simulación de manera efectiva.

Al intentar modificar las funciones de error en la tabla predictiva, el programa duplica los símbolos no terminales en las filas y cambia los estados de la tabla predictiva, como se puede ver en la imagen 3.18. Este comportamiento inesperado puede causar errores en la interpretación de la tabla y afectar la precisión del análisis sintáctico.



Figura 3.18: Duplicación símbolos no terminales MacOS

Durante la simulación descendente, la aplicación no permite incluir la cadena de entrada, lo que limita la capacidad del usuario para probar diferentes cadenas de entrada y verificar la precisión del análisis sintáctico. Además, al realizar la simulación descendente y agregar la declaración, la ventana emergente no se cierra automáticamente al aceptar la acción, lo que puede resultar en una experiencia de usuario inconsistente y confusa.

3.5. Justificación del trabajo

El desarrollo del presente Trabajo de Fin de Grado está justificado porque puede mejorar la versión anterior de la aplicación SimAS 2.0 [5], que a su vez es una evolución de la primera versión desarrollada anteriormente [27]. Esta justificación se fundamenta en varios aspectos relevantes:

1. **Corrección de problemas detectados:** la detección de algunos problemas y limitaciones en la versión anterior de SimAS 2.0 ha evidenciado la necesidad de realizar mejoras significativas. Estos problemas han obstaculizado el aprendizaje completo de los análisis sintácticos ascendentes y descendentes, lo que motiva la búsqueda de soluciones efectivas.
2. **Mejora del aprendizaje interactivo:** se espera que la mejora y corrección de la aplicación conduzca a un funcionamiento más completo y correcto, lo que facilitará el aprendizaje profundo de los conceptos relacionados con los análisis sintácticos. Este enfoque interactivo no solo beneficia a los estudiantes, sino que también puede ser una herramienta valiosa para los profesores en la enseñanza de la asignatura de Procesadores de Lenguajes.
3. **Impacto en el ámbito educativo y académico:** el desarrollo de una herramienta de aprendizaje interactiva como SimAS 3.0 tiene el potencial de tener un impacto significativo en el ámbito educativo y académico. Proporciona una oportunidad para mejorar la comprensión y el dominio de conceptos complejos en el campo de la informática, especialmente en lo relacionado con el procesamiento de lenguajes.
4. **Continuidad y evolución del proyecto:** al seguir la evolución del proyecto desde la versión original hasta la actual, se establece una continuidad en la mejora y perfeccionamiento de la aplicación. Esta progresión refleja un compromiso constante con la excelencia y la adaptación a las necesidades cambiantes de los usuarios y del entorno educativo.
5. **Facilitación del aprendizaje mediante una interfaz intuitiva:** la implementación de una interfaz más intuitiva y amigable con el usuario tiene como objetivo principal hacer que el proceso de aprendizaje sea más accesible y efectivo para los estudiantes. Esta mejora se alinea con las tendencias actuales en el diseño de herramientas educativas, que buscan maximizar la usabilidad y la experiencia del usuario.
6. **Innovación y avance tecnológico:** la mejora y actualización de SimAS 3.0 representa un avance en la aplicación de nuevas tecnologías y metodologías en el campo de la educación informática. Este enfoque innovador busca aprovechar al máximo las capacidades de las herramientas digitales para mejorar la calidad y efectividad del aprendizaje.
7. **Fomento del desarrollo profesional:** la participación en este proyecto brinda la oportunidad de adquirir y desarrollar habilidades técnicas y profesionales relevantes, como el diseño de software, la programación en Java y el trabajo en equipo. Estas habilidades son altamente valoradas en el mercado laboral y contribuyen al crecimiento profesional del estudiante.
8. **Contribución a la comunidad académica:** al compartir los resultados y el código fuente de SimAS 3.0, se contribuye al conocimiento colectivo y se fomenta la colaboración entre instituciones educativas y profesionales del campo de la informática. Esta contribución puede dar lugar a nuevas investigaciones y desarrollos en el área de los procesadores de lenguajes y la educación informática en general.
9. **Relevancia y aplicabilidad práctica:** la aplicación de SimAS 3.0 no solo se limita al ámbito académico, sino que también tiene aplicaciones prácticas en la industria

y el desarrollo de software. La comprensión profunda de los análisis sintácticos que proporciona la herramienta es fundamental para el diseño y la implementación de compiladores y otros sistemas de procesamiento de lenguajes en la práctica profesional.

En resumen, la realización de este trabajo se justifica por la necesidad de mejorar y perfeccionar una herramienta educativa clave en el ámbito de la informática, con el fin de proporcionar una experiencia de aprendizaje más efectiva y enriquecedora para los estudiantes.

3.5.1. Comparativa entre versiones

La Tabla 3.5 muestra, brevemente, una comparativa de las versiones 1.0 y lo que se pretendió alcanzar con la nueva versión 2.0 de SimAS, además de una comparativa con las características esperadas en la nueva versión, “SimAS 3.0 descendente predictivo”.

Tabla 3.5: Comparación entre versiones de SimAS.

	SimAS 1.0	SimAS 2.0	SimAS 3.0 descendente predictivo
Edición de gramáticas de contexto libre	Sí	Sí	Sí
Análisis sintáctico descendente	Sí	Sí	Sí
Análisis sintáctico ascendente	Sí	Sí	No
Generación de árboles sintácticos	No	Parcial	Sí
Generación de informes	No	Parcial	Sí
Definición de funciones predefinidas de tratamiento de errores	No	Parcial	Sí
Compatibilidad multiplataforma	Parcial	Parcial	Sí

Capítulo 4

Restricciones

4.1. Introducción

Las restricciones técnicas y estratégicas son elementos fundamentales que limitan el diseño y la implementación del sistema. En este capítulo, se analizarán detalladamente los factores que restringen o condicionan el proyecto en diversas áreas. Se pueden identificar dos tipos principales de restricciones:

- **Restricciones iniciales:** estas limitaciones son impuestas por la naturaleza misma del proyecto o por las condiciones ambientales en las que se desarrollará. Son estáticas y no pueden ser modificadas o eliminadas a lo largo del proceso de desarrollo. Por ejemplo, el tiempo del que se dispone para completar el proyecto podría considerarse una restricción inicial, ya que es un factor que no puede ser alterado una vez que se ha establecido un cronograma.
- **Restricciones estratégicas:** se refieren a variables de diseño que ofrecen varias alternativas. Durante el desarrollo del sistema, se elegirán aquellas opciones que se consideren más adecuadas en función de los objetivos y requisitos del proyecto. Por ejemplo, la elección entre diferentes tecnologías de base de datos podría ser una restricción estratégica.

4.2. Factores iniciales

4.2.1. Límites temporales

Los límites temporales son un factor inicial crucial a considerar en el desarrollo del proyecto. Se establece como objetivo la presentación de la nueva versión de la aplicación en septiembre de 2024. Dado el tiempo disponible, se contempla priorizar el perfeccionamiento del análisis sintáctico descendente y la mejora general de la aplicación.

4.2.2. Entorno de software

La nueva versión de la aplicación se basa en la infraestructura existente de SimAS 2.0 [5], desarrollada previamente en Java [19] y Java Swing [20] para la gestión de la interfaz gráfica. Se enfocará en corregir las funcionalidades defectuosas de la versión anterior y en agregar nuevas características para mejorar la experiencia del usuario y la eficiencia del programa.

Se continuará utilizando XML como lenguaje para la importación y exportación de gramáticas. Sin embargo, se realizarán ajustes en las definiciones de los elementos y características implementadas en la versión 2.0 para garantizar una mayor coherencia y modularidad en el manejo de las gramáticas.

4.3. Factores estratégicos

4.3.1. Metodología

En esta subsección se presentarán diversas metodologías de desarrollo de software, cada una con enfoques y características distintivas. La selección de la metodología adecuada para el desarrollo del presente proyecto es crucial para su éxito. A continuación, se describen algunas de las metodologías más comunes:

1. **Ciclo de vida en cascada:** este enfoque sigue una secuencia lineal de fases, donde cada fase se completa antes de pasar a la siguiente [pressman]. Aunque fue ampliamente utilizado en el pasado, actualmente está en desuso debido a su rigidez y dificultad para adaptarse a los cambios en los requisitos del proyecto.
2. **Desarrollo iterativo y incremental:** esta metodología implica el desarrollo y la entrega de funcionalidades en ciclos cortos y repetitivos. Es una metodología ampliamente utilizada en la actualidad, ya que permite una mayor flexibilidad y adaptación a medida que avanza el proyecto [scrum].
3. **Metodología RAD (*Rapid Application Development*):** esta metodología se centra en la rápida construcción de prototipos funcionales del software, con un enfoque en la entrega temprana de funcionalidades y la participación activa del cliente. Aunque ha perdido popularidad en los últimos años, sigue siendo utilizada en algunos proyectos donde el tiempo de desarrollo es crítico y los requisitos del sistema pueden no estar completamente definidos desde el principio [rad].
4. **Metodología basada en UML (*Unified Modeling Language*):** esta metodología se basa en el lenguaje de modelado unificado (UML) para el desarrollo de software. Aunque no es una metodología en sí misma, UML sigue siendo ampliamente utilizado en la industria del software como un marco estructurado y formal para el modelado y diseño de sistemas de software [uml].

Dado que el presente proyecto implica el desarrollo de una aplicación de software, la metodología basada en UML [uml] se considera la más adecuada, porque proporciona

un marco estructurado y formal para el modelado y diseño de software, lo que facilita la comprensión y el análisis de los requisitos del sistema. Además, UML es ampliamente utilizado en la industria del software y cuenta con un amplio conjunto de herramientas y recursos disponibles para su aplicación práctica.

4.3.2. Lenguaje de programación

La versión anterior de la aplicación SimAS 2.0 [5] utilizaba Java [19] para el desarrollo de sus funcionalidades, así como Java Swing [20] para la interfaz gráfica. Además, para la carga y guardado de gramáticas, se empleaba XML [30].

Para esta nueva versión, se seguirán utilizando los mismos lenguajes de programación que en las ediciones anteriores. Java continuará siendo el lenguaje principal para implementar las funcionalidades, mientras que Java Swing se utilizará nuevamente para la creación de la interfaz gráfica. Sin embargo, se introducirá una diferencia significativa en el desarrollo de la interfaz de usuario.

4.3.3. Entorno de desarrollo

Para llevar a cabo el desarrollo del proyecto, se consideraron varios entornos de desarrollo, entre ellos NetBeans [2] y IntelliJ IDEA [13]. Inicialmente se contempló el uso de NetBeans debido a su familiaridad y uso previo en asignaturas del grado. Sin embargo, tras evaluar las opciones disponibles, se tomó la decisión de utilizar IntelliJ IDEA como el entorno de desarrollo principal, complementado con Scene Builder [10] para el diseño de interfaces gráficas.

IntelliJ IDEA ofrece numerosas ventajas que lo hacen especialmente adecuado para este proyecto. Entre ellas se incluyen:

- **Interfaz intuitiva y potente:** IntelliJ IDEA cuenta con una interfaz de usuario intuitiva y altamente personalizable que facilita el desarrollo de aplicaciones Java.
- **Soporte integral para JavaFX y Scene Builder:** IntelliJ IDEA ofrece un soporte completo para el desarrollo de aplicaciones JavaFX, incluida la integración directa con Scene Builder para la creación de interfaces gráficas.
- **Funcionalidades avanzadas de refactorización y depuración:** El IDE proporciona potentes herramientas de refactorización de código y depuración, lo que permite mejorar la calidad del código y detectar y corregir errores de manera eficiente.
- **Amplia variedad de complementos y extensiones:** IntelliJ IDEA cuenta con una amplia gama de complementos y extensiones que permiten personalizar y ampliar sus funcionalidades según las necesidades del proyecto.
- **Integración con sistemas de control de versiones:** este IDE ofrece una integración perfecta con sistemas de control de versiones como Git, lo que facilita la colaboración en equipo y el seguimiento de cambios en el código.

La combinación de IntelliJ IDEA y Scene Builder proporciona un entorno de desarrollo completo y eficiente para la creación de la aplicación SimAS 3.0, permitiendo una experiencia de desarrollo fluida y productiva.

4.3.4. Interfaz gráfica de la aplicación

La interfaz gráfica de la aplicación SimAS 3.0 desempeña un papel fundamental en la experiencia del usuario, ya que proporciona una forma intuitiva y eficiente de interactuar con todas las funcionalidades del sistema. La interfaz ha sido diseñada con el objetivo de ser fácil de usar y estéticamente atractiva, manteniendo al mismo tiempo un alto nivel de funcionalidad y usabilidad.

La principal herramienta utilizada para el diseño de la interfaz gráfica es Scene Builder, que permite crear y modificar las interfaces de usuario de manera visual, utilizando componentes gráficos predefinidos y arrastrándolos y soltándolos en la ventana de diseño. Esto facilita enormemente el proceso de diseño y permite una rápida iteración para ajustar y mejorar la apariencia y la disposición de los elementos de la interfaz.

La interfaz gráfica de SimAS 3.0 se compone de varias secciones principales:

- **Barra de menú:** proporciona acceso a todas las funcionalidades de la aplicación, incluyendo la creación y edición de gramáticas, la ejecución de análisis sintáctico, la importación y exportación de archivos, y la configuración de preferencias.
- **Barra de herramientas:** ofrece acceso rápido a las funciones más utilizadas, como la creación de nuevas gramáticas, la apertura y guardado de archivos, y la ejecución de análisis sintáctico.
- **Editor de gramáticas:** permite al usuario crear, editar y guardar gramáticas de contexto libre de forma intuitiva, proporcionando un entorno de edición con resaltado de sintaxis y sugerencias de autocompletado para facilitar la escritura.
- **Área de visualización de resultados:** muestra los resultados del análisis sintáctico, incluyendo árboles de análisis sintáctico, conjuntos primero y siguiente, y otros datos relevantes para el usuario.
- **Panel de configuración:** Permite al usuario personalizar diferentes aspectos de la aplicación, como la apariencia de la interfaz, las preferencias de análisis sintáctico y las opciones de importación y exportación de archivos.

En resumen, la interfaz gráfica de SimAS 3.0 va a ser diseñada para proporcionar una experiencia de usuario intuitiva y eficiente, facilitando la realización de tareas relacionadas con el análisis sintáctico de gramáticas de contexto libre. Mediante el uso de herramientas como Scene Builder, se pretende crear una interfaz atractiva y funcional que satisfaga las necesidades y expectativas de los usuarios. Véase la sección 6.7 de Requisito de la Interfaz y el capítulo 10 del Diseño de la Interfaz.

4.3.5. Sistema Operativo

El correcto funcionamiento de la aplicación SimAS 3.0 se asegura mediante su ejecución en una máquina virtual Java (JVM) [19], lo que garantiza su compatibilidad y portabilidad en una amplia variedad de sistemas operativos. Este enfoque permite que la aplicación sea ejecutable de manera consistente en plataformas como Windows, macOS y Linux, independientemente de las diferencias en el entorno de ejecución subyacente.

Especificamente, en el caso de macOS Sonoma versión 14.0 [macos], se han llevado a cabo pruebas exhaustivas para asegurar que la aplicación se ejecute sin problemas en este sistema operativo. La JVM actúa como una capa de abstracción que permite que el código Java se ejecute de manera uniforme en diferentes sistemas, lo que garantiza una experiencia de usuario consistente y confiable.

La utilización de una máquina virtual Java también facilita la gestión de recursos y la optimización del rendimiento de la aplicación en el entorno de macOS Sonoma versión 14.0. Se han implementado las mejores prácticas de desarrollo de software para aprovechar al máximo las características específicas de este sistema operativo y garantizar una ejecución eficiente de la aplicación.

En resumen, SimAS 3.0 se ejecutará en una máquina virtual Java para garantizar su compatibilidad y portabilidad en macOS Sonoma versión 14.0, así como en otros sistemas operativos, asegurando que los usuarios puedan acceder a todas sus funcionalidades sin importar la plataforma que utilicen.

4.3.6. Representación de las gramáticas de contexto libre

Para almacenar la información de las gramáticas de contexto libre creadas por el usuario, se seguirá usando el formato XML [30] como en las versiones anteriores. Este formato proporciona una estructura definida que asegura la seguridad y la compatibilidad de los datos entre diferentes sistemas. Esta nueva versión SimAS 3.0 explorará la posibilidad de mejorar la organización de los archivos XML de las gramáticas para facilitar su comprensión y manipulación.

4.3.7. Generación de informes de gramáticas y de simulación

Cada gramática creada con la aplicación podrá ir acompañada de un informe detallado que contenga toda la información pertinente. Del mismo modo, tras realizar una simulación, se ofrecerá la opción de generar informes con los resultados obtenidos, incluyendo todos los diagramas relevantes.

Se identificó un problema en la versión SimAS 2.0 relacionado con la generación inconsistente de informes en algunos sistemas operativos. Se investigará a fondo esta cuestión y se implementarán las correcciones necesarias para garantizar que los informes se generen correctamente en todas las plataformas compatibles.

Además, se planea continuar generando los informes en formato **PDF**, debido a su amplia aceptación y su resistencia a ser alterado, lo que facilitaría su distribución y

aumentaría su accesibilidad. También se estudiará la viabilidad de generar los informes en otros formatos para adaptarse a las necesidades específicas de los usuarios.

4.3.8. Implementación de la ayuda

La implementación de la ayuda del programa es un aspecto crucial para proporcionar una experiencia de usuario satisfactoria. En este sentido, se están considerando dos enfoques distintos para llevar a cabo esta tarea:

Por un lado, se contempla la posibilidad de mantener la misma metodología utilizada en la versión anterior del programa. Esta metodología consiste en emplear una página web para proporcionar la ayuda necesaria. Esta opción tiene la ventaja de ser familiar para los usuarios que hayan utilizado versiones anteriores de la aplicación. Además, la generación de la ayuda en formato HTML [html] facilita su integración con la interfaz de usuario desarrollada en Java [19], lo que permite una navegación fluida y una experiencia coherente para el usuario.

Por otro lado, se está evaluando la opción de implementar la ayuda como parte integral de la interfaz de usuario del programa. Esta alternativa implicaría integrar una sección de ayuda directamente dentro de la ventana principal de la aplicación. Esta sección podría presentarse mediante pestañas, paneles desplegables u otros elementos de navegación, proporcionando así un acceso rápido y sencillo a la información relevante. Este enfoque aprovecharía las capacidades de la interfaz gráfica de usuario para ofrecer una experiencia más intuitiva y cohesiva para el usuario.

La elección entre estas dos opciones dependerá de diversos factores, como la complejidad de la ayuda requerida, las preferencias del usuario y las limitaciones de desarrollo. Ambos enfoques tienen sus ventajas y desventajas, por lo que se llevará a cabo un análisis detallado para determinar cuál de ellos se adapta mejor a las necesidades y objetivos del proyecto.

4.3.9. Implementación del tutorial

El tutorial se mantendrá en formato PDF y se incluirá junto con la aplicación para proporcionar a los usuarios una guía detallada sobre el funcionamiento del análisis sintáctico.

Este tutorial estará diseñado para abordar de manera clara y concisa los conceptos teóricos fundamentales del análisis sintáctico, así como para ofrecer ejercicios prácticos que permitan a los usuarios poner en práctica lo aprendido y afianzar su comprensión. De esta manera, se pretende facilitar el aprendizaje y la familiarización de los usuarios con el uso de la aplicación, brindándoles una herramienta efectiva para mejorar sus habilidades en el análisis sintáctico de gramáticas.

4.3.10. Representación de los árboles sintácticos

Como novedad introducida en SimAS 2.0 [5], se implementó la generación de árboles sintácticos de manera iterativa y paralela al análisis sintáctico. En la versión actual, SimAS 3.0, se busca mejorar esta funcionalidad basándose en la experiencia previa con la herramienta Graphviz [11]. Se ha decidido continuar utilizando Graphviz debido a su facilidad de uso y su capacidad para generar árboles gráficos de manera efectiva. Esta decisión se toma con el objetivo de partir de una base sólida y mejorar los puntos en los que la versión anterior presentaba deficiencias en la representación de árboles sintácticos.

Capítulo 5

Recursos

En este capítulo se expondrán los recursos usados durante la realización de este proyecto. Los recursos se definen como aquellos medios de los que se dispone para abordar el proceso de desarrollo del proyecto.

Existen tres grupos de recursos:

- Recursos humanos.
- Recursos de hardware.
- Recursos de software.

5.1. Recursos humanos

- **Autor**

Antonio Llamas García, estudiante del Grado en Ingeniería Informática, especialidad en Computación.

- **Director**

Dr. Nicolás Luis Fernández García. Profesor Titular de Universidad del área de Ciencia de la Computación e Inteligencia Artificial, perteneciente al Departamento de Informática y Análisis Numérico de la Universidad de Córdoba.

5.2. Recursos de hardware

Durante el desarrollo del proyecto se empleará el equipo informático a disposición del autor, cuya descripción corresponde con:

- Ordenador Portátil Características:
 - Procesador Intel(R) Core(TM) 2,4 GHz i5 de 4 núcleos

- 16 Gbytes de memoria RAM.
- Intel Iris Plus Graphics 655 1536 MB.
- Almacenamiento: 240 Gb SSD.

5.3. Recursos de software

Se empleará Scene Builder [10] junto con JavaFX [21] para, el lugar de diseñar la interfaz manualmente programando, realizar este cometido de una forma mucho más intuitiva. Scene Builder es una herramienta que permite diseñar interfaces gráficas de manera visual para aplicaciones JavaFX. Esta elección se debe al hecho de que Scene Builder trabaja con JavaFX, lo que facilitará una creación más intuitiva y eficiente de la interfaz de usuario, mejorando así el proceso de desarrollo.

Además, se modularizarán las características de los elementos de los ficheros XML de gramáticas, permitiendo una separación más clara y una gestión más eficiente de los datos. Finalmente, se mejorará el uso de Graphviz [11], una biblioteca utilizada en la versión anterior, para permitir la generación de los árboles sintácticos de forma iterativa, mejorando así la visualización y comprensión de los resultados del análisis sintáctico.

5.3.1. Sistema Operativos

- MacOs Sonoma, versión 14.0

5.3.2. Entorno de desarrollo y lenguaje de programación

- IntelliJ IDEA[13]: entorno de desarrollo utilizado para la codificación y pruebas de la aplicación.
- Java [19]: lenguaje de programación multiplataforma.
- Java Swing [20]: biblioteca de Java para el desarrollo de aplicaciones gráficas.

5.3.3. Herramientas de documentación

- L^AT_EX [15]: software de formateo de documentación científica y técnica.
- OverLeaf [26]: *front-end* web para L^AT_EX.

5.3.4. Herramientas para la edición de diagramas

- Draw.io (Diagrams.net) [14]: editor gráfico para el diseño de los diagramas.

Parte II

Análisis

Capítulo 6

Especificación de requisitos

6.1. Introducción

Este capítulo se dedica a la especificación detallada de los requisitos del sistema, abarcando tanto los requisitos *funcionales* como los *no funcionales*, así como los requisitos de *información* y de *interfaz*. Antes de entrar en los detalles específicos de estos requisitos, se proporciona una descripción funcional del simulador, resaltando los módulos identificados previamente. Posteriormente, se identificarán los actores que interactuarán con la aplicación.

6.2. Actores

El software diseñado en este proyecto está destinado a un entorno educativo, dirigido principalmente a *profesores* y *estudiantes*. A pesar de esta diferenciación en los roles, la aplicación no distinguirá funcionalmente entre estos usuarios, ya que ambos tendrán acceso a las mismas funcionalidades. Por lo tanto, cualquier referencia a los *actores* de la aplicación se aplicará de manera uniforme a todos los usuarios finales, independientemente de su rol específico.

6.3. Descripción modular del sistema

El sistema se compone de varios módulos integrados, cada uno con un propósito específico. A continuación, se describen en detalle los módulos principales que conforman la aplicación:

- Editor de gramáticas.
- Simulador gráfico descendente.
- Tutorial.
- Sistema de ayuda.

La Figura 6.1 muestra la descomposición modular del sistema.

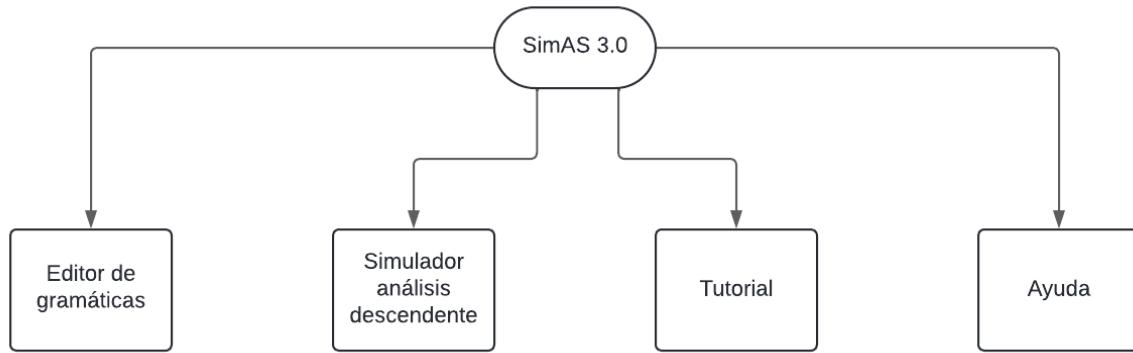


Figura 6.1: Descomposición modular del sistema

6.3.1. Módulo del Editor de Gramáticas

El editor de gramáticas es una herramienta fundamental dentro del sistema, permitiendo a los usuarios crear y modificar gramáticas de contexto libre. Las funcionalidades específicas de este módulo incluyen:

1. **Creación de gramáticas:** los usuarios pueden definir nuevas gramáticas, especificando sus componentes esenciales, como símbolos terminales, no terminales, el símbolo inicial y las reglas gramaticales.
2. **Gestión de vocabulario:** permite a los usuarios añadir y eliminar símbolos del vocabulario de una gramática, gestionando tanto los símbolos terminales como los no terminales, que son necesarios para formar las reglas.
3. **Gestión de reglas gramaticales o producciones:** facilita la introducción, modificación y eliminación de las reglas gramaticales de una gramática, asegurando que no haya duplicados.
4. **Validación de gramáticas:** este proceso asegura que la gramática es válida, verificando que todos los símbolos sean accesibles y generadores:
 - **Accesibilidad:** un símbolo se considera accesible si se puede alcanzar desde el símbolo inicial mediante una serie de derivaciones.
 - **Generatividad:** un símbolo es generativo si puede derivar en una cadena de símbolos terminales o la cadena vacía.

En caso de errores, se informará al usuario y se proporcionarán sugerencias para la corrección.

5. **Almacenamiento de gramáticas:** los usuarios pueden guardar sus gramáticas en archivos para uso futuro, independientemente de si la gramática ha sido completamente validada.

6. **Carga de gramáticas:** permite cargar gramáticas desde archivos existentes, validando que no estén corruptos y notificando al usuario cualquier error durante el proceso.
7. **Consulta de gramáticas:** muestra la estructura completa de la gramática actual, incluyendo todos sus componentes, y permite generar un informe detallado.

6.3.2. Módulo del Simulador Gráfico Descendente Predictivo

Este módulo constituye el núcleo de la aplicación, proporcionando la capacidad de simular el análisis sintáctico descendente predictivo de manera visual e interactiva. A continuación se detalla su funcionamiento.

6.3.2.1. Creación y carga de gramáticas

El primer paso en el proceso de simulación es la creación o carga de una gramática. Los usuarios pueden utilizar el módulo de edición de gramáticas para definir una nueva gramática o cargar una existente desde un archivo. Es fundamental que la gramática haya sido validada previamente para asegurar su correcta estructura y funcionalidad antes de ser utilizada en la simulación.

6.3.2.2. Generación de conjuntos *Primero* y *Siguiente*

Una vez que la gramática está lista, se procede a la generación de los conjuntos Primero y Siguiente. Estos conjuntos son esenciales para la construcción de la tabla predictiva, ya que determinan las posibles reglas gramaticales o producciones que pueden aplicarse durante el análisis de una cadena de entrada.

6.3.2.3. Construcción de la tabla predictiva

Después de obtener los conjuntos Primero y Siguiente, se debe construir la tabla predictiva del análisis sintáctico descendente. Esta tabla es una guía que indica qué regla gramatical debe aplicarse para cada combinación de símbolo de entrada y símbolo no terminal. La tabla predictiva es fundamental para la correcta derivación de la cadena de entrada y facilita la detección y manejo de errores durante el análisis.

6.3.2.4. Implementación de funciones de recuperación de errores

Aunque esta parte es opcional, la implementación de funciones de recuperación de errores es altamente recomendable. Estas funciones se utilizan para manejar las celdas vacías en la tabla predictiva, proporcionando mecanismos para continuar el análisis sintáctico en caso de encontrar un error. Entre las funciones de recuperación de errores se incluyen:

- Eliminar el símbolo actual de la entrada.

- Insertar un nuevo símbolo terminal en la entrada, con funciones específicas para cada símbolo terminal.
- Terminar el análisis si hay un final inesperado de la entrada.

6.3.2.5. Simulación del análisis sintáctico descendente predictivo

Con la tabla predictiva completada, el usuario puede introducir una cadena de entrada para simular el análisis sintáctico descendente predictivo. Durante esta fase, el simulador utilizará la tabla predictiva para derivar la cadena de entrada, gestionando los errores según las funciones de recuperación definidas. El sistema informará al usuario sobre el estado de la cadena de entrada, indicando si es reconocida o si se han producido errores. La aplicación permitirá mostrar la derivación de la cadena de entrada generada por el análisis sintáctico descendente predictivo.

6.3.2.6. Generación del árbol sintáctico descendente

Una característica avanzada del simulador es la capacidad de generar el árbol sintáctico descendente. Esta opción permite a los usuarios visualizar la estructura jerárquica de la derivación de la cadena de entrada en tiempo real, mejorando su comprensión del proceso de análisis sintáctico. El árbol sintáctico se construye de forma paralela a la simulación, proporcionando una representación gráfica de cada paso del análisis.

6.3.2.7. Creación de informes

Finalmente, tras la simulación, se generará un informe detallado que documenta todo el proceso. Este informe, disponible en formatos PDF o HTML, incluirá:

- Información detallada sobre la gramática utilizada.
- Los conjuntos Primero y Siguiiente generados.
- La tabla predictiva construida.
- Las funciones de recuperación de errores definidas.
- La cadena de entrada y su estado (reconocida o no).
- Los errores detectados durante el análisis y cómo fueron manejados.
- La derivación de la cadena de entrada generada por el análisis sintáctico descendente predictivo.
- El árbol sintáctico asociado a la derivación de la cadena de entrada generada por el análisis sintáctico descendente predictivo.

Este informe puede ser almacenado en disco por el usuario, proporcionando una referencia completa y detallada del análisis sintáctico descendente realizado.

6.3.3. Módulo del Tutorial

Este módulo tiene un enfoque educativo, diseñado para proporcionar todos los objetivos pedagógicos del proyecto. Se ubicará de manera accesible desde cualquier otro módulo del programa, permitiendo al usuario consultarla en cualquier momento, incluso mientras interactúa con otras partes de la aplicación.

6.3.3.1. Estructura del tutorial

El tutorial se organizará en las siguientes secciones:

1. **Introducción y fundamentos teóricos:** esta sección cubre los conceptos teóricos fundamentales del análisis sintáctico, con un enfoque en las gramáticas de contexto libre. Proporcionará una base sólida para entender los principios detrás del análisis sintáctico descendente.
2. **Lecciones de análisis sintáctico descendente:** aquí se explicarán los métodos de análisis sintáctico descendente predictivo. Las lecciones estarán estructuradas de manera que el usuario pueda seguir cada paso del proceso de análisis de manera clara y comprensible.
3. **Ejemplos y ejercicios prácticos:** esta parte del tutorial incluirá una serie de ejercicios prácticos y ejemplos resueltos. Los usuarios podrán realizar estos ejercicios y verificar sus respuestas utilizando el simulador. Esta sección está diseñada para reforzar el aprendizaje a través de la práctica y la aplicación de conceptos teóricos.

6.3.4. Módulo de Ayuda

El módulo de *Ayuda* proporciona una guía completa sobre el uso de la aplicación. Está diseñado para ser una referencia detallada que abarca todos los aspectos del funcionamiento del programa.

6.3.4.1. Contenidos de la ayuda

1. **Introducción a la aplicación:** esta sección presentará una visión general de la aplicación, describiendo sus objetivos y las principales funcionalidades que ofrece. Proporcionará una primera impresión de lo que los usuarios pueden esperar del programa.
2. **Requisitos de instalación y procedimiento:** aquí se detallarán los requisitos del sistema necesarios para ejecutar la aplicación, así como las instrucciones paso a paso para su instalación y desinstalación. Se incluirán también soluciones a posibles problemas que puedan surgir durante estos procesos.
3. **Guía de la interfaz de usuario:** esta sección ofrecerá una explicación detallada de los diferentes componentes de la interfaz del simulador. Se proporcionarán descripciones de cada elemento de la interfaz y su funcionalidad, ayudando a los usuarios a navegar y utilizar el programa de manera eficiente.

4. **Descripción de los módulos del programa:** En esta parte, se explicará en profundidad el funcionamiento de cada uno de los módulos de la aplicación. Se incluirán instrucciones específicas sobre cómo utilizar cada módulo, junto con ejemplos prácticos y consejos para maximizar su uso.

6.4. Requisitos funcionales

En esta sección se recogen los requisitos funcionales de la aplicación, los cuales se denotarán como **RF-<acrónimo del módulo>-<número de requisito>**. Los requisitos funcionales se encuentran agrupados según los módulos funcionales del Sistema, que son las siguientes:

- **Editor de gramáticas:** requisitos funcionales del *editor de gramáticas*.
- **Simulador:** requisitos funcionales del *simulador*.
- **Tutorial:** requisitos funcionales del *tutorial*.
- **Ayuda:** requisitos funcionales del *ayuda*.

6.4.1. Requisitos funcionales del Editor de gramáticas

Esta sección detalla todos los requisitos funcionales del *Editor de gramáticas*, listados a continuación:

- **RF-E-1.** El editor de gramáticas permitirá al usuario crear una nueva gramática de contexto libre.
- **RF-E-2.** El editor permitirá la creación y modificación del conjunto de símbolos de la gramática (símbolos terminales y no terminales):
 - **RF-E-2.1.** También admitirá el uso de caracteres especiales, tales como ϵ y \rightarrow , entre otros.
 - **RF-E-2.2.** Será posible definir símbolos personalizados para la construcción de reglas, como, por ejemplo, <asignación>.
 - **RF-E-2.3.** Se gestionarán y notificarán todos los errores relacionados con la creación y edición de símbolos de manera clara y concisa para el usuario.
 - **RF-E-2.4.** Las reglas de la gramática se ordenarán en función del símbolo no terminal de su parte izquierda.
- **RF-E-3.** El editor permitirá crear reglas gramaticales utilizando los símbolos definidos por el usuario. Se verificará que el *antecedente* de cada regla es un *único símbolo no terminal*.
- **RF-E-4.** La inserción de reglas gramaticales estará controlada y se notificará al usuario cualquier error ocurrido. Si una regla es introducida más de una vez, el sistema lo advertirá y solamente permitirá una instancia de la misma.

- **RF-E-5.** El editor permitirá la edición completa de las reglas gramaticales, pudiéndose modificar tanto el antecedente como el consecuente de cada regla. También será posible eliminar reglas de la gramática.
- **RF-E-6.** Por defecto, el símbolo inicial de la gramática será el antecedente de la primera regla gramatical introducida. No obstante, el editor permitirá que el usuario lo modifique posteriormente si lo desea. La gramática debe tener un único símbolo inicial.
- **RF-E-7.** Cada gramática tendrá un nombre y una descripción asignados por el usuario, los cuales podrán ser modificados en cualquier momento.
- **RF-E-8.** La gramática se visualizará en el editor a medida que se construye. Se actualizará con cada nueva regla introducida y podrá ser visualizada en cualquier momento.
- **RF-E-9.** El editor permitirá generar un informe detallado de la gramática actual. Este informe incluirá todos los elementos visualizados y componentes de la gramática, y podrá ser guardado en el disco según la preferencia del usuario.
- **RF-E-10.** El editor permitirá cerrar la gramática en curso, notificando al usuario de la acción. Se ofrecerá la opción de guardar la gramática antes de cerrarla para evitar la pérdida de datos.
- **RF-E-11.** La gramática actual podrá ser guardada en el disco para su uso posterior. Se solicitará al usuario un nombre para el archivo y la ubicación de almacenamiento, además de un nombre para la gramática si no se ha proporcionado anteriormente.
- **RF-E-12.** El editor permitirá abrir una gramática previamente guardada desde un fichero y cargarla en el editor. Se controlarán todos los errores que puedan surgir durante esta operación y se notificarán al usuario si la gramática no puede ser cargada.
- **RF-E-13.** La validación de la gramática comprobará que es adecuada para la simulación. Durante esta validación se revisarán los siguientes aspectos:
 - **RF-E-13.1.** La gramática debe tener un único símbolo inicial (según lo especificado en **RF-E-6**).
 - **RF-E-13.2.** Todos los símbolos deben ser útiles, es decir, accesibles y generadores.
- **RF-E-14.** Cualquier error identificado durante la validación será comunicado al usuario. No se permitirá continuar con la simulación si existen errores en la gramática. Si la validación es exitosa, se informará al usuario mediante un mensaje.
- **RF-E-15.** El editor permitirá transferir gramáticas al simulador para llevar a cabo la simulación del análisis sintáctico descendente predictivo. Solo se podrán transferir aquellas gramáticas que hayan sido validadas con éxito. Esta acción inicializará el simulador y cargará la gramática para su simulación.

6.4.2. Requisitos funcionales del Simulador

- **RF-S-1.** El simulador permitirá aplicar el método de análisis descendente predictivo.
 - **RF-S-1.1.** Se deberá verificar que la gramática no presenta recursividad por la izquierda y está adecuadamente factorizada por la izquierda. En caso contrario, se aplicarán los algoritmos correspondientes. El usuario podrá consultar tanto la gramática original como la transformada.
- **RF-S-2.** El simulador generará los conjuntos Primero y Siguiiente, esenciales para el análisis sintáctico descendente predictivo.
- **RF-S-3.** El simulador construirá la *tabla predictiva* y todos los elementos auxiliares necesarios para el método elegido.
- **RF-S-4.** En la tabla predictiva, el simulador permitirá el uso de funciones de manejo de errores para aplicar el método de nivel de frase de recuperación de errores.
- **RF-S-5.** El simulador notificará cualquier error que ocurra durante la creación de las tablas.
- **RF-S-6.** Durante la simulación, las funciones de error serán invocadas en caso de que se produzca un error.
- **RF-S-7.** La simulación podrá iniciarse una vez que se haya generado la tabla predictiva, independientemente de que esté completada con funciones de error o no.
- **RF-S-8.** El simulador solicitará al usuario una cadena de entrada para verificar si es reconocida por el método de simulación. Esta cadena podrá ser vacía (ϵ). Se controlarán todos los posibles errores de entrada-salida o de simulación.
- **RF-S-9.** Una vez creada la tabla predictiva e introducida la cadena de entrada, la simulación podrá comenzar de forma *continua* o *paso a paso*.
- **RF-S-10.** En la simulación continua, se mostrará el resultado final de la simulación en una tabla que incluirá todos los errores producidos, en su caso. La tabla se generará y se mostrará completa al usuario.
- **RF-S-11.** En la simulación paso a paso, se mostrarán los resultados de la simulación progresivamente, construyendo la tabla de resultados de forma *incremental* a medida que el usuario avance en la simulación.
- **RF-S-12.** Durante la simulación, ya sea automática o paso a paso, el simulador mostrará la construcción del árbol correspondiente al análisis sintáctico descendente.
- **RF-S-13.** El simulador informará al usuario de cualquier error que ocurra durante la simulación. Si existen funciones de manejo de errores asociadas, se utilizarán e informarán al usuario sobre el motivo de su uso.
- **RF-S-14.** Al finalizar, el simulador podrá generar un informe de la simulación en formato PDF o HTML. Este informe incluirá la gramática, su tabla de análisis, las funciones de error, la derivación generada y el árbol sintáctico construido.

- **RF-S-15.** Al finalizar, el simulador podrá mostrar la derivación generada por el análisis sintáctico predictivo.
- **RF-S-16.** Al finalizar, el simulador podrá el árbol sintáctico asociado a la derivación generada por el análisis sintáctico predictivo.
- **RF-S-17.** El simulador permitirá guardar el informe de simulación en el disco duro, indicando la gramática, la tabla predictiva, las funciones de error, la tabla de simulación del análisis sintáctico, la derivación generado y el árbol sintáctico asociado.
- **RF-S-18.** Las simulaciones podrán ser abortadas, permitiendo al usuario cambiar la gramática o el conjunto de funciones de error. Se informará al usuario si desea conservar el trabajo realizado hasta ese momento y guardar los informes de simulación producidos (si los hay).

6.4.3. Requisitos funcionales del Tutorial

- **RF-T-1.** El usuario podrá acceder al tutorial desde cualquier sección de la aplicación.
- **RF-T-2.** El tutorial estará dividido en lecciones que abarcarán todo el procedimiento de operación con los mecanismos del análisis sintáctico descendente predictivo.

6.4.4. Requisitos funcionales de la Ayuda

- **RF-A-1.** La sección de ayuda será accesible desde cualquier parte de la aplicación.
- **RF-A-2.** La ayuda estará organizada en capítulos que cubrirán todos los módulos de la aplicación, proporcionando explicaciones detalladas de todos los procedimientos y utilizando ejemplos para ilustrar su uso.

6.5. Requisitos no funcionales

En esta sección se detallan los requisitos no funcionales del Sistema, identificados como **RNF-<número de requisito>**.

- **RNF-1.** La interfaz del sistema debe ser intuitiva y fácil de usar, centrada en facilitar la comprensión didáctica del análisis sintáctico. Se utilizarán menús fáciles de navegar, botones de acción y gráficos pertinentes a los métodos de análisis sintáctico. Además, deberá manejar errores de manera amigable, proporcionando mensajes claros y soluciones.
- **RNF-2.** La aplicación debe manejar adecuadamente todos los errores que ocurran durante su ejecución (errores de entrada-salida, de ejecución, de simulación, etc.).
- **RNF-3.** El sistema debe proporcionar respuestas a las solicitudes del usuario dentro de un tiempo razonable.

- **RNF-4.** La importación y exportación de gramáticas se hará utilizando el formato XML. Se buscará la máxima modularidad, diferenciando entre el antecedente y el consecuente junto con sus respectivos valores.
- **RNF-5.** Seguridad del sistema. El sistema debe estar protegido y asegurar una descarga segura tanto de informes como de gramáticas.
- **RNF-6.** Escalabilidad del sistema. El sistema debe ser capaz de adaptarse a aumentos o reducciones en sus módulos, integrando nuevos componentes o eliminando los innecesarios sin afectar a la funcionalidad general.
- **RNF-7.** Fiabilidad del sistema. El sistema debe ser confiable y cumplir con los requisitos especificados por el usuario.
- **RNF-8.** Compatibilidad del sistema. El sistema debe ser compatible con los sistemas operativos MacOS, Windows y Linux.

6.6. Requisitos de la información

En esta sección se especifican los requisitos de la información, identificados como **RI-<acrónimo del módulo>-<número de requisito>**. Estos requisitos se agrupan en función de los conceptos identificados durante la especificación de los requisitos del usuario.

Los requisitos de la información se organizan en los siguientes bloques de información:

- **Gramática de contexto libre:** requisitos de información para la *gramática de contexto libre*, que será un sub-módulo auxiliar del módulo *editor de gramáticas*.
- **Editor de gramáticas:** requisitos de información para el módulo *editor de gramáticas*.
- **Simulador:** requisitos de información para el módulo *simulador* del análisis sintáctico descendente predictivo.
- **Ayuda de la aplicación:** requisitos de información para el módulo *ayuda de la aplicación*.
- **Tutorial de la aplicación:** requisitos de información para el módulo *tutorial de la aplicación*.
- **Informes:** requisitos de información para los submódulos *informe de gramática* e *informe de simulación*, pertenecientes al módulo *simulador*.

6.6.1. Gramática de contexto libre

- **RI-G-1.** Una gramática de contexto libre consta de un conjunto de símbolos (terminales y no terminales), un *símbolo inicial*, y una o más reglas gramaticales. Además, la gramática deberá tener un *nombre* y una *descripción* asociada.

- **RI-G-2.** El conjunto de símbolos de la gramática incluirá tanto símbolos *terminales* como *no terminales*. Cada símbolo será representado por una cadena de caracteres y tendrá un código asociado (indicando si es un carácter especial como ϵ).
 - **RI-G-2.1.** Los **símbolos no terminales** estarán representados entre ángulos, como $<\text{asignación}>$ o por una letra mayúscula única, como S .
 - **RI-G-2.2.** Los **símbolos terminales** se representarán en minúsculas (*identificador*, *número*, etc.). También podrán ser símbolos terminales los operadores aritméticos, lógicos o relacionales, y los símbolos como coma, punto, punto y coma, dos puntos, paréntesis, llaves, corchetes, etc.
- **RI-G-3.** El símbolo inicial de la gramática será, por defecto, el antecedente de la primera regla introducida por el usuario, aunque el usuario podrá seleccionar cualquier símbolo *no terminal* como símbolo inicial.
- **RI-G-4.** Una **regla gramatical** o **producción** se compone de dos partes: el *antecedente* y el *consecuente*.
 - **RI-G-4.1.** El **antecedente** de una regla gramatical debe ser un *símbolo no terminal*.
 - **RI-G-4.2.** El **consecuente** de una regla gramatical puede ser una secuencia de símbolos gramaticales (tanto *terminales* como *no terminales*), incluyendo la posibilidad de estar vacío (ϵ). No hay limitaciones en cuanto al número de símbolos en el consecuente.
 - **RI-G-4.3.** Las flechas (\rightarrow) se utilizan para separar el antecedente y el consecuente de una regla.
- **RI-G-5.** En una gramática de contexto libre, todos los símbolos no terminales deben aparecer en el antecedente de, al menos, una regla gramatical.
- **RI-G-6.** La palabra vacía se representa con ϵ . No puede haber otro símbolo que signifique lo mismo que la palabra vacía, aunque se pueden utilizar otros símbolos especiales como α , β , etc.
- **RI-G-7.** No puede haber dos símbolos (ya sean *terminales* o *no terminales*) idénticos.

6.6.2. Editor de gramáticas

- **RI-E-1.** El editor permite manejar gramáticas de contexto libre, facilitando las operaciones descritas en la sección 6.4.1 de Requisitos funcionales del Editor de gramáticas. La información necesaria sobre las gramáticas se encuentra en la sección anterior. El editor soporta el uso de múltiples gramáticas de contexto libre simultáneamente.
- **RI-E-2.** El informe de gramática se generará en formato PDF o HTML, incluyendo todos los detalles relevantes de los componentes de la gramática.

6.6.3. Simulador

- **RI-S-1.** El simulador trabaja con gramáticas de contexto libre, permitiendo cargar solo una gramática a la vez.
- **RI-S-2.** El simulador incluye los siguientes componentes: *cadena de entrada, modo de operación y generación de árbol sintáctico*.
- **RI-S-3.** Será necesario calcular los conjuntos Primero y Siguiente.
- **RI-S-4.** La cadena de entrada estará compuesta por símbolos *terminales* que deben pertenecer a la gramática. Además, se podrá incluir el símbolo vacío ϵ .
 - **RI-S-4.1.** La cadena puede ser **vacía**, conteniendo únicamente el símbolo vacío (ϵ).
 - **RI-S-4.2.** La cadena solamente puede **contener** símbolos terminales de la gramática.
- **RI-S-5.** El simulador puede operar en modo *continuo* o *paso a paso*.
- **RI-S-6.** El simulador construye la tabla predictiva basándose en la gramática y el método de análisis descendente predictivo.
- **RI-S-7.** La tabla predictiva puede tener asignada una o más funciones de manejo de errores.
- **RI-S-8.** El simulador genera la tabla de análisis usando la *tabla predictiva* y la *cadena a reconocer*.
 - La primera columna indicará el contenido de la pila del análisis sintáctico descendente predictivo. En la primera fila de la primera columna, aparecerá el símbolo $\$$ y el símbolo inicial para indicar el comienzo del análisis.
 - La segunda columna indicará el contenido de la cadena de entrada que se va a analizar. Deberá terminar con el símbolo $\$$.
 - Indicará las acciones análisis sintáctico descendente predictivo: emparejar, aplicar una regla gramatical, usar una función de error o aceptar.
- **RI-S-9.** Se puede mostrar simultáneamente la derivación de la cadena de entrada generada por el análisis sintáctico descendente predictivo. Se usará una ventana separada para mejorar la comprensión del usuario.
- **RI-S-10.** Se puede generar simultáneamente el árbol sintáctico generado por la derivación de la cadena de entrada. Se usará una ventana separada para mejorar la comprensión del usuario.
- **RI-S-11.** El informe de simulación se basará en la información de la tabla de análisis o la tabla predictiva, y mostrará el resultado de la simulación y cualquier error encontrado. El informe se almacenará en formato PDF.

6.6.4. Ayuda de la aplicación

- **RI-A-1.** La ayuda de la aplicación se dividirá en capítulos.
- **RI-A-2.** Cada capítulo tendrá un menú de navegación para desplazarse dentro del capítulo o volver al índice de la ayuda.
- **RI-A-3.** Cada capítulo se almacenará en un archivo en formato *HTML*.

6.6.5. Tutorial de la aplicación

- **RI-T-1.** El tutorial de la aplicación consistirá en varias lecciones.
- **RI-T-2.** Cada lección incluirá explicaciones, ejemplos y ejercicios sobre los conceptos de los métodos de análisis sintáctico. Tendrá un menú para navegar por el contenido o volver al índice del tutorial.
- **RI-T-3.** El tutorial se almacenará en formato PDF.

6.6.6. Informes

- **RI-INF-1.** El informe de gramática incluirá información sobre los componentes de la gramática (símbolos terminales y no terminales, símbolo inicial y reglas gramaticales o producciones). El formato del informe será PDF o HTML.
- **RI-INF-2.** El informe de simulación contendrá detalles de la gramática a simular, el método de análisis sintáctico utilizado, los conjuntos Primero y Siguiente, la tabla predictiva y las funciones de error, la tabla de análisis generada y el resultado del análisis. También se podrá incluir la derivación generada y el árbol sintáctico correspondiente. El informe se generará en formato PDF o HTML.

6.7. Requisitos de la interfaz

Los requisitos de la interfaz detallan el formato de interacción de la aplicación con su entorno.

En esta sección se presentan los requisitos de la interfaz, que se identificarán como **RINT-<número requisito>**.

- **RINT-1.** La interfaz del sistema debe ser adaptable y ajustar su diseño al tamaño de la pantalla del dispositivo en uso, ya sea PC, tablet o smartphone.
- **RINT-2.** El sistema solo mostrará al usuario las opciones y acciones que son relevantes en cada momento específico.
- **RINT-3.** Los nombres de los distintos elementos de la interfaz serán claros y explicativos.

- **RINT-4.** El sistema permitirá la navegación entre los distintos subsistemas siguiendo un formato de pestañas. Estas pestañas se generarán en cada momento en función de las necesidades del subsistema con el que se esté trabajando.
- **RINT-5.** El sistema proporcionará mensajes de error informativos para asegurar que no haya errores en el análisis.
- **RINT-6.** El sistema utilizará ventanas emergentes para mostrar información adicional, evitando que el usuario tenga que retroceder para revisarla.

Capítulo 7

Análisis funcional

7.1. Introducción

Este capítulo ofrece una visión general del sistema, comenzando con una introducción preliminar, seguida de la identificación de los actores involucrados. A continuación, se detallarán los casos de uso mediante un diagrama contextual general, para después desglosar y afinar cada caso de uso que compone la aplicación. Finalmente, este capítulo concluirá con la validación de los casos de uso en relación con los requisitos funcionales previamente descritos.

Un caso de uso describe una secuencia de acciones realizadas por el sistema en respuesta a las interacciones de un actor, especificando el comportamiento del sistema o de una parte del mismo.

Un diagrama de casos de uso representa visualmente un conjunto de casos de uso, las relaciones entre ellos y los actores involucrados. Este diagrama se utiliza para modelar la vista estática de un sistema, permitiendo visualizar su comportamiento externo y, de esta manera, identificar qué debe hacer el sistema independientemente de cómo se implementará.

Los casos de uso del sistema se especificarán según los siguientes aspectos:

- **Nombre:** denominación completa y codificación del diagrama de casos de uso.
- **Actor:** usuario participante en el diagrama.
- **Descripción:** breve explicación del diagrama en su conjunto.
- **Casos de uso:** lista y breve descripción de los casos de uso presentes en el diagrama.
- **Flujo de eventos principal:** enumeración de los pasos que debe seguir el actor para lograr el objetivo del caso de uso.
- **Flujo de eventos alternativo:** pasos a seguir en caso de que el actor no pueda alcanzar el objetivo principal del caso de uso.

7.2. Actores del Sistema

Como se mencionó en el capítulo anterior, el sistema cuenta con un único tipo de actor final, que incluye tanto a *profesores* como a *alumnos* que utilizan el software para realizar análisis de gramáticas de contexto libre.

Por lo tanto, la especificación de los casos de uso que se presenta a continuación se centra exclusivamente en el actor del sistema, denominado **Usuario**.

7.3. Especificación de casos de uso

En esta sección, se procederá a especificar cada caso de uso detalladamente, siguiendo el esquema previamente mencionado. Se iniciará con la presentación de un diagrama de casos de uso de contexto, que proporcionará una visión general de la interacción entre el usuario y el sistema. Luego, cada caso de uso será refinado individualmente para ofrecer una comprensión profunda de las funcionalidades y comportamientos esperados del sistema.

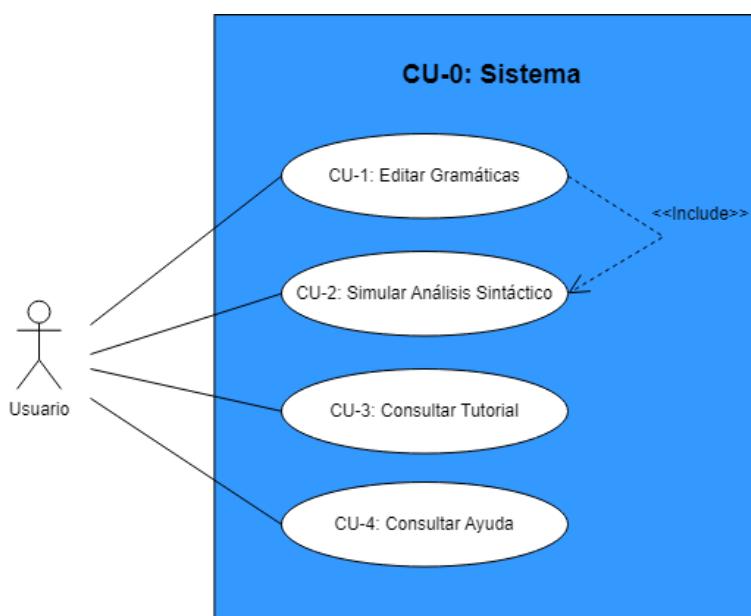


Figura 7.1: Diagrama de contexto del Sistema

7.3.1. Diagrama de contexto: CU-0. Sistema

La figura 7.1 presenta el diagrama de contexto del sistema, en el cual se identifican los cuatro casos de uso principales que conforman la aplicación. Estos casos de uso son:

- **CU-1. Editar gramática:** corresponde al módulo para la *edición de gramáticas*.
- **CU-2. Simular análisis sintáctico descendente predictivo:** se refiere al módulo que *simula el analizador sintáctico descendente predictivo utilizando gramáticas de contexto libre*.

contexto libre.

- **CU-3. Consultar tutorial:** relacionado con el módulo del *tutorial*.
- **CU-4. Consultar ayuda:** está relacionado con el módulo de la *ayuda*.

La especificación del caso de uso CU-0. Sistema se presenta en la tabla 7.1.

Tabla 7.1: Caso de uso: CU-0. Sistema

Nombre	CU-0. Sistema (diagrama de contexto). Nivel: 0
Descripción	Proporciona al usuario una visión general de todos los subsistemas que componen la aplicación, registrando todas las interacciones posibles del usuario con estos.
Actores	Usuario
Casos de uso	<ol style="list-style-type: none"> 1. CU-1. Editar Gramáticas: este subsistema se encarga de la edición de gramáticas de contexto libre. 2. CU-2. Simular análisis sintáctico descendente predictivo: este subsistema se dedica a la simulación del analizador sintáctico descendente predictivo utilizando gramáticas de contexto libre. 3. CU-3. Consultar Tutorial: proporciona acceso al tutorial de la aplicación. 4. CU-4. Consultar Ayuda: permite acceder a la ayuda de la aplicación.
Continúa en la página siguiente	

Tabla 7.1 – continua de la página anterior

Flujo de eventos principal	<ol style="list-style-type: none"> 1. El usuario inicia sesión en el Sistema. 2. Se presentan todas las opciones disponibles al usuario. 3. El usuario selecciona una opción del Sistema. 4. El Sistema procesa la selección del usuario. 5. Se proporciona retroalimentación al usuario sobre la selección realizada. 6. El usuario interactúa con el subsistema seleccionado (por ejemplo, edición de gramáticas, simulación, consulta de tutoriales o ayuda). 7. El usuario puede guardar su progreso y retornar al menú principal en cualquier momento.
Flujo de eventos alternativo	<ol style="list-style-type: none"> 1. Ocurre un error durante la inicialización del Sistema. 2. Se notifica al usuario sobre el error. 3. El Sistema sugiere posibles soluciones o pasos a seguir para resolver el problema. 4. Si el error persiste, el Sistema guarda cualquier progreso no guardado previamente y cierra la aplicación de manera segura. 5. El usuario es redirigido a una página de soporte técnico para obtener asistencia adicional.

A continuación, se procederá a detallar todos los casos de uso que conforman las funcionalidades del simulador. Este análisis detallado se realizará de manera sistemática, abordando cada caso de uso general identificado en el diagrama de contexto (figura 7.1): CU-1, CU-2, CU-3 y CU-4.

7.3.2. CU-1. Editar Gramáticas

Este caso de uso representa la edición de gramáticas de contexto libre en el sistema.

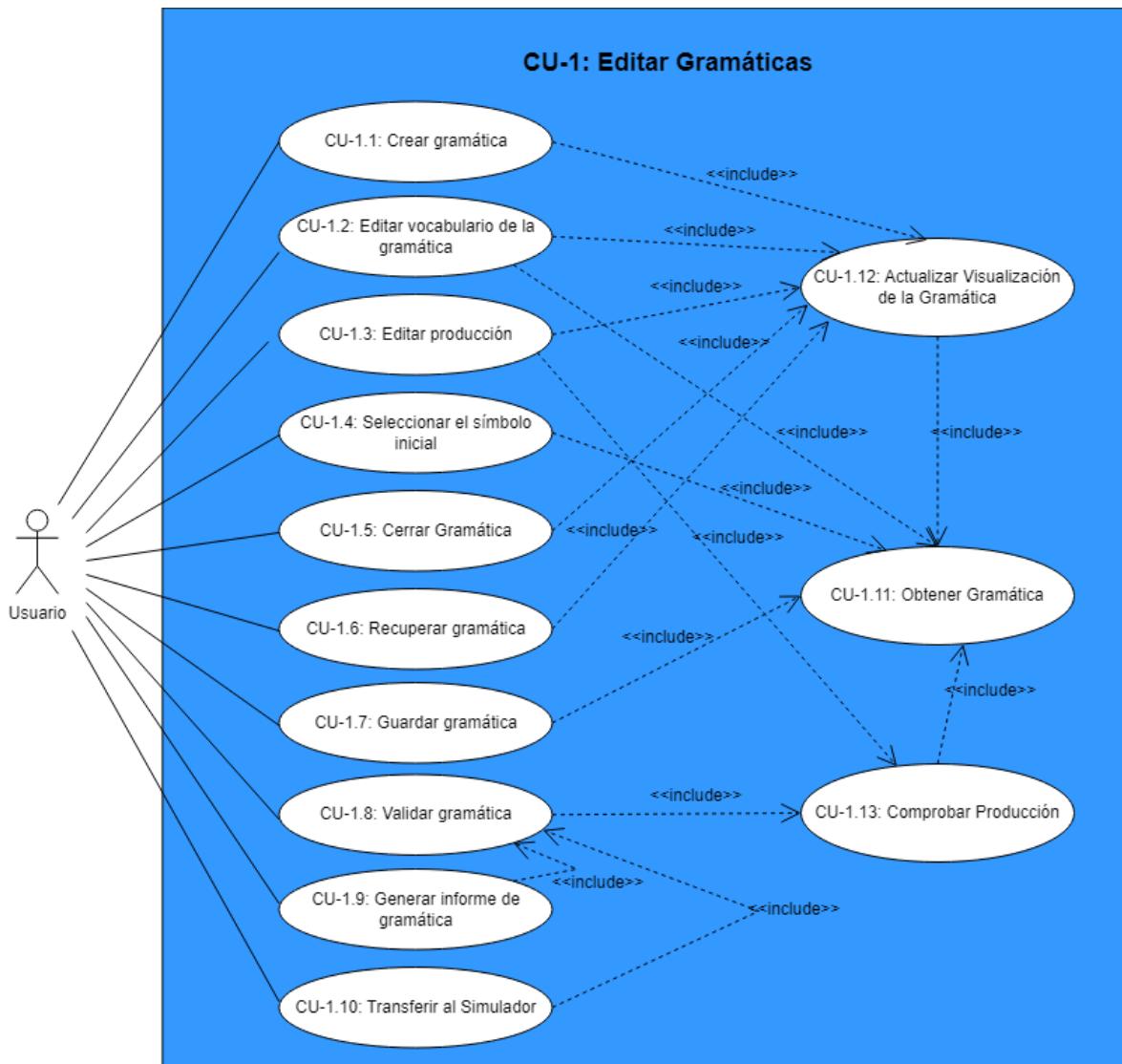


Figura 7.2: CU-1: Editar Gramáticas

La figura 7.2 muestra la representación gráfica de este caso de uso, que está formado por los siguientes sub-casos de uso:

- **CU-1.1. Crear gramática.**
- **CU-1.2. Editar vocabulario de la gramática.**
- **CU-1.3. Editar producción.**
- **CU-1.4. Seleccionar el símbolo inicial.**
- **CU-1.5. Cerrar gramática.**
- **CU-1.6. Recuperar gramática.**
- **CU-1.7. Guardar gramática.**

- CU-1.8. Validar gramática.
- CU-1.9. Generar informe de gramática.
- CU-1.10. Transferir al simulador.
- CU-1.11. Obtener gramática.
- CU-1.12. Actualizar visualización de la gramática.
- CU-1.13. Comprobar producción.

Tabla 7.2: Caso de uso: CU-1. Edición de gramáticas

Nombre	CU-1. Editar Gramáticas Nivel: 1
Descripción	Permite al usuario la creación y edición de gramáticas de contexto libre.
Actores	Usuario

Casos de uso	<ol style="list-style-type: none"> 1. CU-1.1. Crear gramática: permite <i>crear</i> una gramática a partir de una producción inicial. 2. CU-1.2. Editar vocabulario de la gramática: permite <i>editar</i> el vocabulario de símbolos (V_N, V_T) de la gramática. 3. CU-1.3. Editar producción: permite la <i>edición</i> de producciones de la gramática. 4. CU-1.4. Seleccionar el símbolo inicial: permite la <i>selección</i> del símbolo inicial de la gramática. 5. CU-1.5. Cerrar gramática: <i>cierra</i> una gramática abierta en el editor. 6. CU-1.6. Recuperar gramática: <i>carga</i> una gramática almacenada en un fichero. 7. CU-1.7. Guardar gramática: <i>almacena</i> una gramática en un fichero. 8. CU-1.8. Validar gramática: <i>valida</i> una gramática de contexto libre. 9. CU-1.9. Generar informe: <i>genera</i> un informe de una gramática de contexto libre. 10. CU-1.10. Transferir al simulador: <i>transfiere</i> una gramática de contexto libre al simulador para simular el análisis sintáctico. 11. CU-1.11. Obtener gramática: <i>obtiene</i> el vocabulario de símbolos (V_N, V_T) así como las producciones de la gramática. 12. CU-1.12. Actualizar visualización de la gramática: <i>actualiza</i> la visualización de la gramática en la interfaz del simulador para permitir al usuario visualizarla a medida que realiza modificaciones en la misma. 13. CU-1.13. Comprobar producción: <i>comprueba</i> si una regla gramatical o producción es correcta o no.
---------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Flujo de eventos principal	<ol style="list-style-type: none"> 1. El usuario accede al Sistema. 2. Se presentan todas las opciones disponibles al usuario. 3. El usuario selecciona una de las opciones del Sistema. 4. El Sistema ejecuta la acción correspondiente a la opción seleccionada. 5. El Sistema proporciona retroalimentación al usuario sobre la acción realizada. 6. El usuario puede guardar su progreso y retornar al menú principal en cualquier momento.
Flujo de eventos alternativo	<ol style="list-style-type: none"> 1. Ocurre un error durante la inicialización del Sistema. 2. Se notifica al usuario sobre el error. 3. El Sistema sugiere posibles soluciones o pasos a seguir para resolver el problema. 4. Si el error persiste, el Sistema guarda cualquier progreso no guardado previamente y cierra la aplicación de manera segura. 5. El usuario es redirigido a una página de soporte técnico para obtener asistencia adicional.
Flujo de eventos alternativo adicional	<p>La acción representada por el caso de uso CU-1.12 se lleva a cabo de forma continua, sin que lo solicite el usuario. Además, esta acción se ejecuta cada vez que el usuario modifica la gramática utilizando alguno de los casos de uso que implican este sub-caso de uso.</p>

En las siguientes secciones, se refinará cada uno de los sub-casos de uso que componen el caso de uso CU-1.

7.3.2.1. CU-1.1. Crear gramática

Este caso de uso describe el proceso de creación de una gramática de contexto libre. La figura 7.3 muestra la representación gráfica del caso de uso, mientras que la tabla 7.3

presenta su descripción tabular.

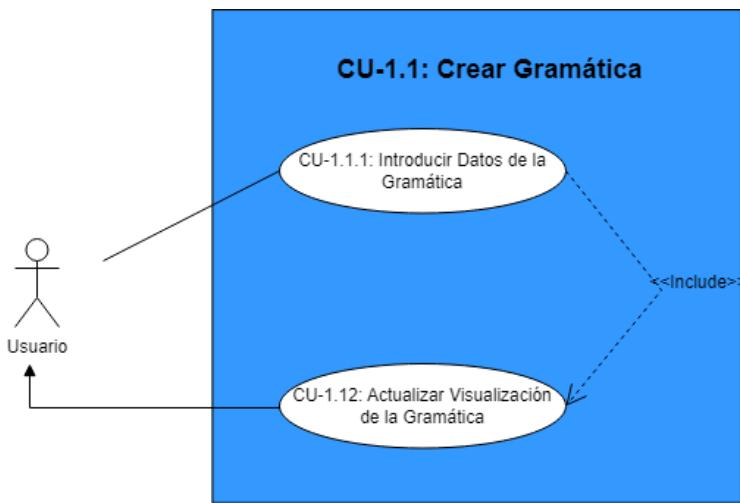


Figura 7.3: CU-1.1: Crear gramática

Tabla 7.3: Caso de uso: CU-1.1 Crear gramática

Nombre	CU-1.1 Crear gramática. Nivel: 2
Descripción	Permite al usuario crear gramáticas de contexto libre.
Actores	Usuario
Casos de uso	<ol style="list-style-type: none"> 1. CU-1.1.1 Introducir datos de la gramática: permite que el usuario <i>introduzca</i> los datos de una gramática de contexto libre. 2. CU-1.1.2 Actualizar visualización de la gramática: <i>actualiza</i> la visualización de la gramática en la interfaz del simulador.
Flujo de eventos principal	<ol style="list-style-type: none"> 1. El usuario introduce el nombre de la gramática y una descripción de la misma. 2. Se crea la gramática y se actualiza la visualización en la interfaz. 3. El usuario recibe una confirmación de la creación exitosa de la gramática.
continúa en la página siguiente	

Tabla 7.3 – continúa de la página anterior

Flujo de eventos alternativo	<p>1. Si en el paso 2 se detecta algún error durante la actualización de la visualización:</p> <ul style="list-style-type: none"> a) Se informa al usuario del error producido. b) El sistema proporciona posibles soluciones o pasos a seguir.
-------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

El caso de uso *CU-1.1.1 Introducir datos de la gramática* no requiere un refinamiento adicional debido a su simplicidad. Este caso de uso se encarga de solicitar al usuario el nombre y la descripción de la gramática, excluyendo símbolos y producciones, para posteriormente incorporarlos en la definición de la gramática.

7.3.2.2. CU-1.2. Editar vocabulario de la gramática

Este caso de uso describe el proceso de edición del vocabulario de la gramática, incluyendo tanto los símbolos terminales como los no terminales. La figura 7.4 muestra la representación gráfica del caso de uso, mientras que la tabla 7.4 presenta su descripción tabular.

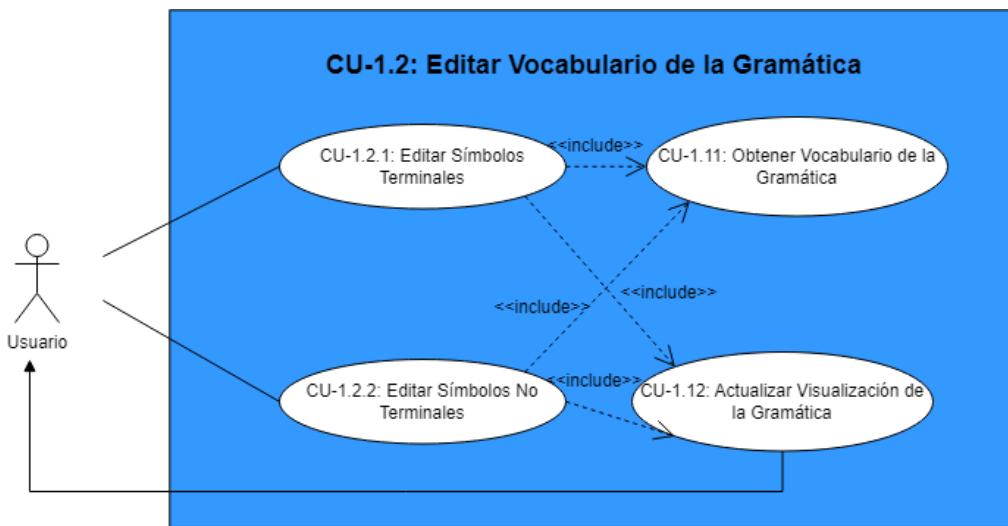


Figura 7.4: CU-1.2: Editar vocabulario de la gramática

Tabla 7.4: Caso de uso: CU-1.2 Editar vocabulario de la gramática

Nombre	CU-1.2 Editar vocabulario de la gramática.
	Nivel: 2

continúa en la página siguiente

Tabla 7.4 – continúa de la página anterior

Descripción	Permite al usuario editar el vocabulario (V_N, V_T) de la gramática.
Actores	Usuario
Casos de uso	<ol style="list-style-type: none"> 1. CU-1.2.1. Editar símbolos terminales: permite que el usuario <i>introduzca o modifique</i> el conjunto de símbolos terminales de la gramática. 2. CU-1.2.2. Editar símbolos no terminales: permite que el usuario <i>introduzca o modifique</i> el conjunto de símbolos no terminales de la gramática. 3. CU-1.11. Obtener gramática: <i>obtiene</i> el vocabulario de símbolos (V_N, V_T) así como las producciones de la gramática. 4. CU-1.12. Actualizar visualización de la gramática: <i>actualiza</i> la visualización de la gramática en la interfaz del simulador.
Flujo de eventos principal	<ol style="list-style-type: none"> 1. El usuario elige entre alguna de las opciones disponibles: <ol style="list-style-type: none"> a) Introducir el conjunto de símbolos terminales de la gramática y se comprueba si son correctos. b) Introducir el conjunto de símbolos no terminales de la gramática y se comprueba si son correctos.
Flujo de eventos alternativo	<p>Si durante la opción o función 1, se detecta que alguno de los símbolos terminales introducido no es correcto:</p> <ol style="list-style-type: none"> 1. Se informa al usuario del error producido. 2. Se pide al usuario que reintroduzca el símbolo terminal que produjo el error.
Flujo de eventos alternativo	<p>Si durante la opción o función 2, se detecta que alguno de los símbolos no terminales introducido no es correcto:</p> <ol style="list-style-type: none"> 1. Se informa al usuario del error producido. 2. Se pide al usuario que reintroduzca el símbolo no terminal que produjo el error.

Los casos de uso *CU-1.2.1 Editar símbolos terminales* y *CU-1.2.2 Editar símbolos no terminales* no se refinará más, puesto que son casos de uso bastante sencillos y que pueden ser fácilmente traducidos a una implementación posterior sin necesidad de ser refinados.

El caso de uso CU-1.2.1 mostrará al usuario una ventana para *introducir* o *modificar* (en el caso de que ya los haya introducido) todos los símbolos terminales de la gramática, mientras que el caso de uso CU-1.2.2 permitirá modificar los no terminales.

7.3.2.3. CU-1.3. Editar producción

Este caso de uso representa la edición de una producción de una gramática. En la figura 7.5 se muestra la representación gráfica del caso de uso y en la tabla 7.5 se muestra su representación tabular.

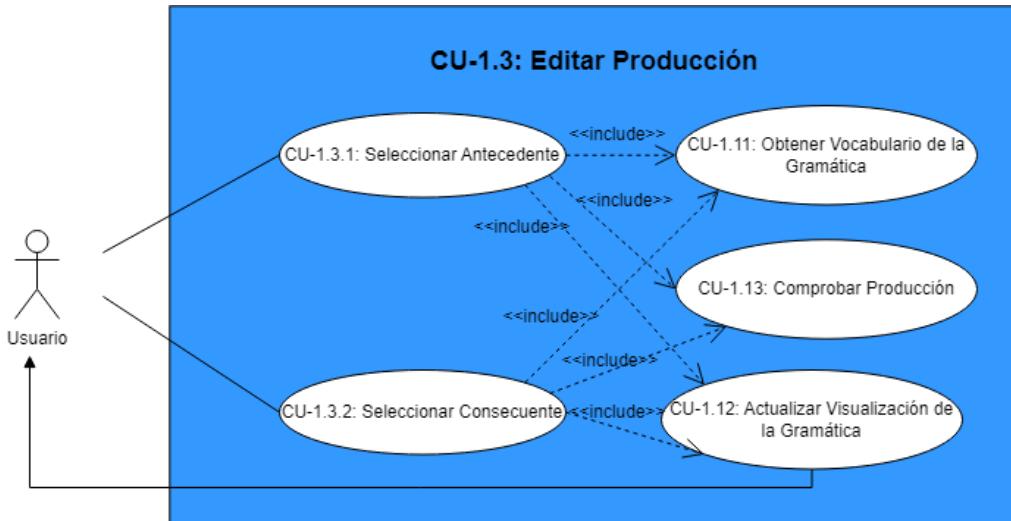


Figura 7.5: CU-1.3: Editar producción

Tabla 7.5: Caso de uso: CU-1.3: Editar producción

Nombre	CU-1.3 Editar producción. Nivel: 2
Descripción	Permite al usuario editar una producción de la gramática de contexto libre.
Actores	Usuario
continúa en la página siguiente	

Tabla 7.5 – continúa de la página anterior

Casos de uso	<ol style="list-style-type: none"> 1. CU-1.3.1 Seleccionar antecedente: permite que el usuario <i>modifique</i> el antecedente de una producción. 2. CU-1.3.2 Seleccionar consecuente: permite que el usuario <i>modifique</i> el consecuente de una producción. 3. CU-1.11. Obtener gramática: <i>obtiene</i> el vocabulario de símbolos (V_N, V_T) así como las producciones de la gramática. 4. CU-1.12. Actualizar visualización de la gramática: <i>actualiza</i> la visualización de la gramática en la interfaz del simulador. 5. CU-1.13 Comprobar producción: <i>comprueba</i> si una producción es correcta.
Flujo de eventos principal	<ol style="list-style-type: none"> 1. El usuario <i>selecciona</i> el antecedente o el consecuente (o ambos) de la producción que desea crear o modificar (en caso de que en CU-1 el usuario hubiera seleccionado alguna producción que ya existía). 2. El usuario <i>modifica</i> el antecedente o el consecuente (o ambos). 3. Se <i>comprueba</i> si la producción modificada es correcta.
Flujo de eventos alternativo	<p>Si no existe ningún símbolo en el vocabulario de la gramática (o solamente existe uno de los conjuntos de símbolos: terminales o no terminales) o bien alguno de los símbolos posee un error, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario del evento y se le pide que introduzca o modifique el conjunto de símbolos de la gramática.
Flujo de eventos alternativo	<p>Si se produce algún fallo durante el paso 2, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario del error producido. 2. Se <i>deshacen</i> los cambios realizados por el usuario en la producción.

continúa en la página siguiente

Tabla 7.5 – continúa de la página anterior

Flujo de eventos alternativo	<p>Si se comprueba la producción resultante y esta no es correcta, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario del error (diciendo además si está en el antecedente, en el consecuente o en ambos) y el motivo del mismo. 2. Se <i>pide</i> al usuario que corrija el error.
-------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Los casos de uso *CU-1.3.1 Seleccionar antecedente* y *CU-1.3.2 Seleccionar consecuente* no se refinará más, puesto que son casos de uso bastante sencillos y que pueden ser fácilmente traducidos a una implementación posterior sin necesidad de ser refinados.

El caso de uso CU-1.3.1 mostrará un diálogo al usuario que le permita seleccionar el antecedente de la producción de entre los símbolos no terminales que introdujo (mediante el caso de uso 1.2).

El caso de uso CU-1.3.2, por su parte, permitirá al usuario modificar el consecuente de la producción (realizando acciones análogas a la del caso de uso 1.3.1).

7.3.2.4. CU-1.4. Seleccionar el símbolo inicial

Este caso de uso representa la selección del símbolo inicial de la gramática. En la figura 7.6 se muestra la representación gráfica del caso de uso y en la tabla 7.6 se muestra su representación tabular.

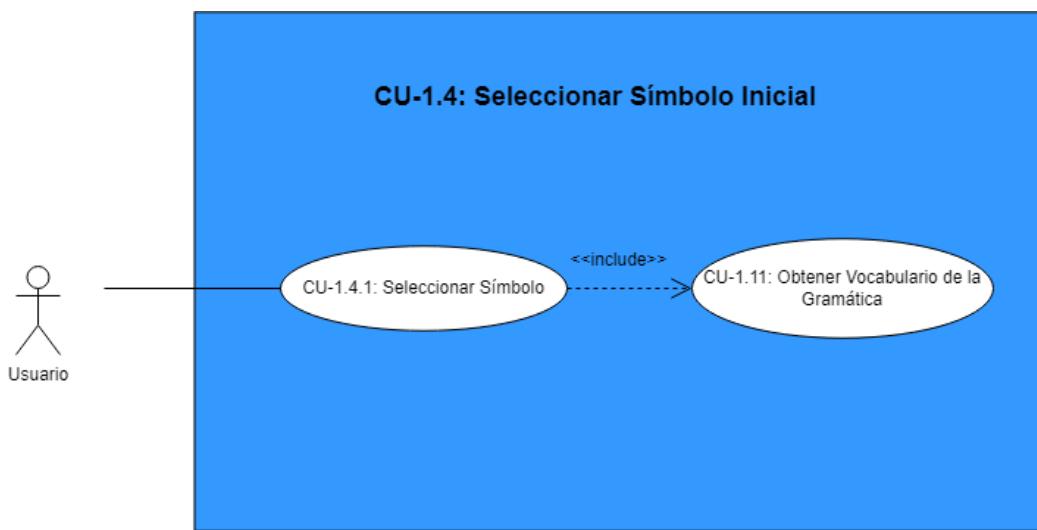


Figura 7.6: CU-1.4: Seleccionar el símbolo inicial

Tabla 7.6: Caso de uso: CU-1.4 Seleccionar el símbolo inicial

Nombre	CU-1.4 Seleccionar el símbolo inicial. Nivel: 2
Descripción	Permite al usuario seleccionar el símbolo inicial de la gramática de contexto libre.
Actores	Usuario
Casos de uso	<ol style="list-style-type: none"> 1. CU-1.4.1 Seleccionar símbolo: permite que el usuario <i>seleccione</i> un símbolo inicial de entre el conjunto de símbolos no terminales de la gramática. 2. CU-1.11. Obtener gramática: <i>obtiene</i> el vocabulario de símbolos (V_N, V_T) así como las producciones de la gramática.
Flujo de eventos principal	<ol style="list-style-type: none"> 1. Se <i>obtienen</i> los símbolos no terminales de la gramática. 2. Se <i>muestra</i> al usuario un diálogo para seleccionar el símbolo inicial de la gramática. 3. El usuario <i>selecciona</i> un símbolo para que sea el inicial. 4. Se <i>comprueba</i> la producción en la que el símbolo aparece como antecedente. 5. Se <i>asigna</i> dicho símbolo como símbolo inicial.
Flujo de eventos alternativo	<p>Si se produce algún fallo durante el paso 3, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario del error producido. 2. Se <i>termina</i> la acción de seleccionar símbolo inicial.
continúa en la página siguiente	

Tabla 7.6 – continúa de la página anterior

Flujo de eventos alternativo	Si se produce algún fallo durante el paso 4, se procede como sigue: <ol style="list-style-type: none">1. Se <i>informa</i> al usuario de que la producción en la que el símbolo aparece en el antecedente no es correcta.2. Se <i>permite</i> al usuario modificar dicha producción.3. Se <i>reanuda</i> el paso 4, permitiendo al usuario continuar con la asignación de símbolo inicial.
Flujo de eventos alternativo	Si se producen errores en el paso 5, se procede como sigue: <ol style="list-style-type: none">1. Se <i>informa</i> al usuario del error.2. Se <i>reintenta</i> asignar el símbolo, abortando la acción actual en caso de que siga siendo imposible dicha asignación.

El caso de uso *CU-1.4.1 Seleccionar símbolo* no se refinará más, puesto que se trata de un caso de uso bastante sencillo y que puede ser fácilmente traducido a una implementación posterior sin necesidad de ser refinados. Este caso de uso simplemente se encargará de mostrar, una vez obtenidos los símbolos no terminales de la gramática, un diálogo que permita seleccionar el símbolo inicial (de entre todos los símbolos no terminales que posee la gramática).

7.3.2.5. CU-1.5. Cerrar gramática

Este caso de uso se encarga de cerrar la gramática seleccionada por el usuario (con una posterior actualización de la vista de la gramática).

Este caso de uso no se refinará más, puesto que el proceso de cierre de la gramática es inmediato y evidente.

7.3.2.6. CU-1.6. Recuperar gramática

Este caso de uso representa la recuperación de una gramática desde un fichero. En la figura 7.7 se muestra la representación gráfica del caso de uso y en la tabla 7.7 se muestra su representación tabular.

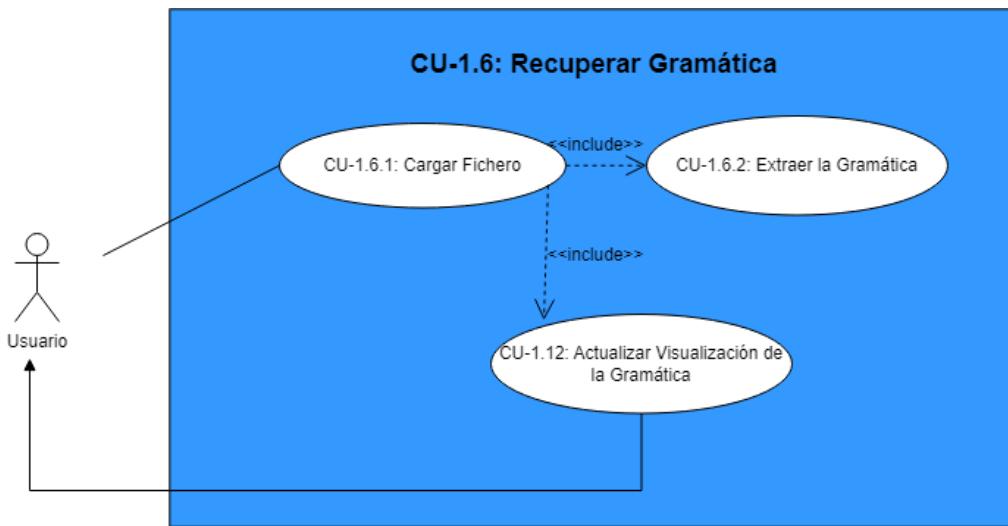


Figura 7.7: CU-1.6: Recuperar gramática

Tabla 7.7: Caso de uso: CU-1.6 Recuperar gramática

Nombre	CU-1.6 Recuperar gramática. Nivel: 2
Descripción	Permite al usuario recuperar una gramática de un fichero.
Actores	Usuario
Casos de uso	<ol style="list-style-type: none"> 1. CU-1.6.1 Cargar fichero: <i>carga</i> un fichero que contiene una gramática. 2. CU-1.6.2 Extraer la gramática: <i>extrae</i> la gramática contenida en el fichero recuperado. 3. CU-1.12. Actualizar visualización de la gramática: <i>actualiza</i> la visualización de la gramática en la interfaz del simulador.
Flujo de eventos principal	<ol style="list-style-type: none"> 1. El usuario <i>selecciona</i> el fichero que contiene la gramática que desea cargar. 2. Se <i>extrae</i> la gramática del fichero. 3. Se <i>carga</i> la gramática en el editor (creando toda la estructura de la gramática y actualizando la vista de la gramática pertinente).
continúa en la página siguiente	

Tabla 7.7 – continúa de la página anterior

Flujo de eventos alternativo	<p>Si se produce algún fallo durante el paso 1, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario de que no ha podido seleccionar ningún fichero. 2. Se <i>termina</i> la acción de recuperar gramática.
Flujo de eventos alternativo	<p>Si se produce algún fallo durante el paso 2, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario de que la gramática no pudo ser extraída o cargada en el simulador, debido a posibles problemas en la estructura del archivo. 2. Se <i>termina</i> la acción de recuperar gramática.
Flujo de eventos alternativo	<p>Si se produce algún fallo durante el paso 3, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario de que la gramática ha podido ser extraída pero no se puede cargar en el editor. 2. Se <i>termina</i> la acción de recuperar gramática.

Los casos de uso *CU-1.6.1 Cargar fichero* y *CU-1.6.2 Extraer gramática* no se refinará más, puesto que son casos de uso bastante sencillos y que pueden ser fácilmente traducidos a una implementación posterior sin necesidad de ser refinados.

7.3.2.7. CU-1.7. Guardar gramática

Este caso de uso representa el almacenamiento de una gramática. En la figura 7.8 se muestra la representación gráfica del caso de uso y en la tabla 7.8 se muestra su representación tabular.

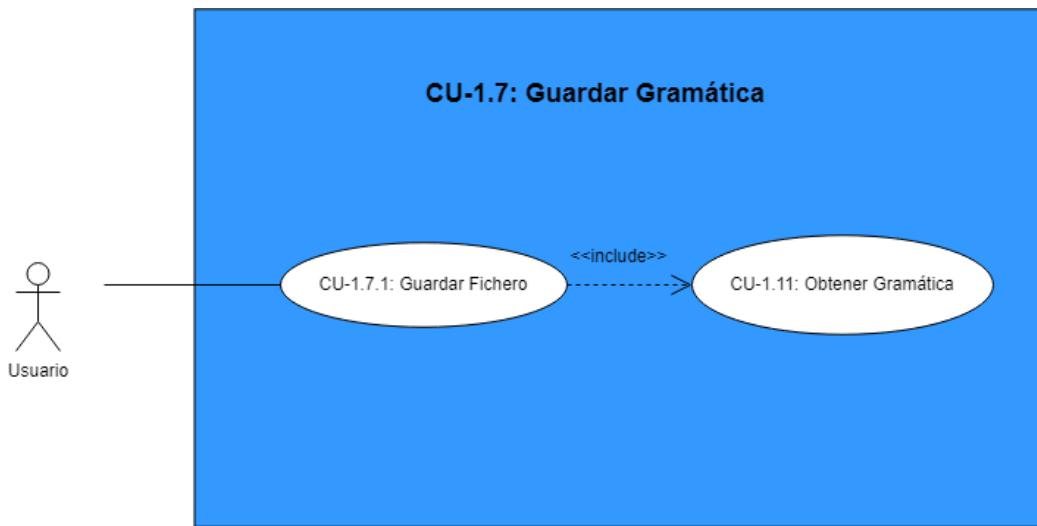


Figura 7.8: CU-1.7: Guardar gramática

Tabla 7.8: Caso de uso: CU-1.7 Guardar gramática

Nombre	CU-1.7 Guardar gramática. Nivel: 2
Descripción	Permite al usuario guardar una gramática de un fichero.
Actores	Usuario
Casos de uso	<ol style="list-style-type: none"> 1. CU-1.7.1 Guardar fichero: <i>guarda</i> en un fichero la gramática creada. 2. CU-1.11. Obtener gramática: <i>obtiene</i> el vocabulario de símbolos (V_N, V_T) así como las producciones de la gramática.
Flujo de eventos principal	<ol style="list-style-type: none"> 1. El usuario <i>selecciona</i> el fichero en el que desea guardar la gramática. 2. Se <i>guarda</i> la gramática en un fichero (con el formato especificado en el capítulo anterior).
Flujo de eventos alternativo	<p>Si se produce algún fallo durante el paso 1, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario de que no ha podido seleccionar crear el fichero. 2. Se <i>termina</i> la acción de guardar gramática.
Flujo de eventos alternativo	<p>Si se produce algún fallo durante el paso 2, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario de que la gramática no pudo ser guardada. 2. Se <i>termina</i> la acción de guardar gramática.

El caso de uso *CU-1.7.1 Guardar fichero* no se refinará más, puesto que se trata de un proceso simple que puede ser fácilmente traducido a una implementación posterior sin necesidad de ser refinado más.

7.3.2.8. CU-1.8. Validar gramática

Este caso de uso representa la validación de una gramática. En la figura 7.9 se muestra la representación gráfica del caso de uso y en la tabla 7.9 se muestra su representación tabular.

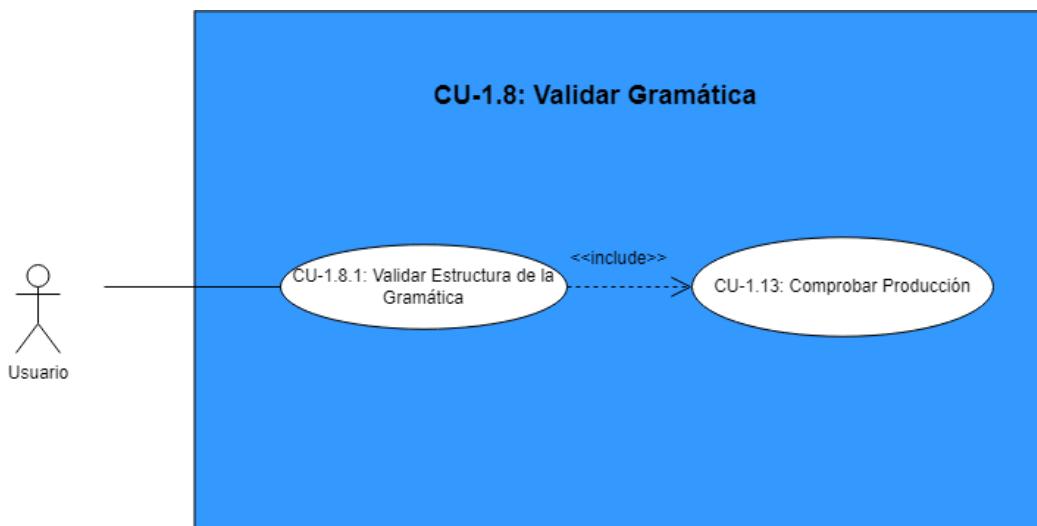


Figura 7.9: CU-1.8: Validar gramática

Tabla 7.9: Caso de uso: CU-1.8 Validar gramática

Nombre	CU-1.8 Validar gramática. Nivel: 2
Descripción	Permite al usuario validar una gramática de contexto libre.
Actores	Usuario
Casos de uso	<ol style="list-style-type: none"> 1. CU-1.8.1 Validar estructura de la gramática: <i>valida</i> la estructura de la gramática, formada por el conjunto de producciones, símbolos (<i>terminales</i> y <i>no terminales</i>) y símbolo inicial. 2. CU-1.12 Comprobar producción: <i>comprueba</i> si una producción es correcta.
Flujo de eventos principal	<ol style="list-style-type: none"> 1. Se <i>obtiene</i> todo el vocabulario de la gramática a ser validada. 2. Se <i>comprueban</i> todas las producciones que componen la gramática. 3. Se <i>informa</i> al usuario acerca del resultado del la validación.
Flujo de eventos alternativo	<p>Si se produce algún fallo durante el paso 1, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario de que no ha podido obtener el vocabulario de la gramática. 2. Se <i>termina</i> la acción de validar gramática.
Flujo de eventos alternativo	<p>Si se produce algún fallo durante el paso 2, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario de que no se han podido comprobar las producciones de la gramática. 2. Se <i>termina</i> la acción de validar gramática.
Flujo de eventos excepcional	<p>Si durante la ejecución del paso 2, alguna producción comprobada posee errores (esto excluye cualquier error de funcionamiento, descrito en el flujo alternativo anterior), se procede como sigue:</p> <ol style="list-style-type: none"> 1. El <i>resultado</i> de la validación será insatisfactorio.

El caso de uso *CU-1.8.1 Validar estructura de la gramática* no se refinará más,

puesto que es un proceso simple que puede ser fácilmente traducido a una implementación posterior sin necesidad de ser refinado más.

7.3.2.9. CU-1.9. Generar informe de gramática

Este caso de uso representa la generación de un informe de una gramática. En la figura 7.10 se muestra la representación gráfica del caso de uso y en la tabla 7.10 se muestra su representación tabular.

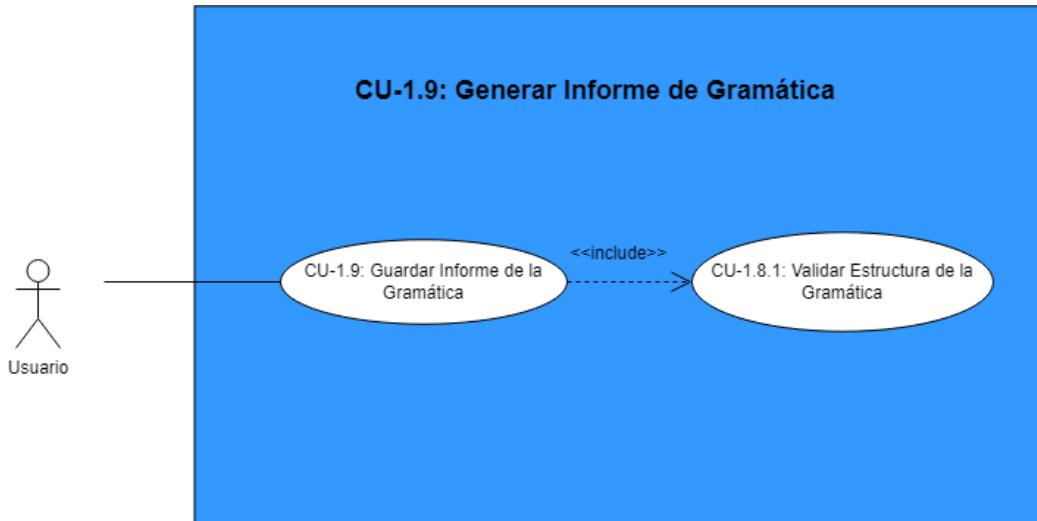


Figura 7.10: CU-1.9: Generar informe de gramática

Tabla 7.10: Caso de uso: CU-1.9 Generar informe de gramática

Nombre	CU-1.9 Generar informe de gramática. Nivel: 2
Descripción	Permite al usuario generar un informe de una gramática.
Actores	Usuario
Casos de uso	<ol style="list-style-type: none"> 1. CU-1.9.1 Guardar informe de la gramática: <i>guarda</i> un fichero que contiene el informe de la gramática. 2. CU-1.8.1 Validar estructura de la gramática: <i>valida</i> la estructura de la gramática, formada por el conjunto de producciones y símbolos (<i>terminales</i> y <i>no terminales</i>).

continúa en la página siguiente

Tabla 7.10 – continúa de la página anterior

Flujo de eventos principal	<ol style="list-style-type: none"> 1. Se realiza una <i>validación</i> de la estructura de la gramática, para así comprobar si la gramática es correcta. 2. Se <i>genera</i> un informe de la gramática, según lo especificado en <i>RI-INF-1</i>.
Flujo de eventos alternativo	<p>Si se produce algún fallo durante el paso 1, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario que no ha podido generar el informe de la gramática. 2. Se <i>termina</i> la acción de generar informe de gramática.
Flujo de eventos alternativo	<p>Si se produce algún fallo durante el paso 2, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario de que el informe no ha podido ser guardado. 2. Se <i>termina</i> la acción de generar informe de gramática.
Flujo de eventos excepcional	<p>Si durante la ejecución del paso 1, se detecta que la gramática no es válida, puesto que posee errores (esto excluye cualquier error de funcionamiento, descrito en el flujo alternativo anterior), se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>aborta</i> la generación del informe. 2. Se <i>informa</i> al usuario de que la gramática contiene errores.

El caso de uso *CU-1.9.1 Guardar informe de la gramática* no se refinará más, puesto que es un **proceso simple** que puede ser fácilmente traducido a una implementación posterior sin necesidad de ser refinado.

7.3.2.10. CU-1.10. Transferir al simulador

Este caso de uso representa la transferencia de una gramática al simulador. En la figura 7.11 se muestra la representación gráfica del caso de uso y en la tabla 7.11 se muestra su representación tabular.

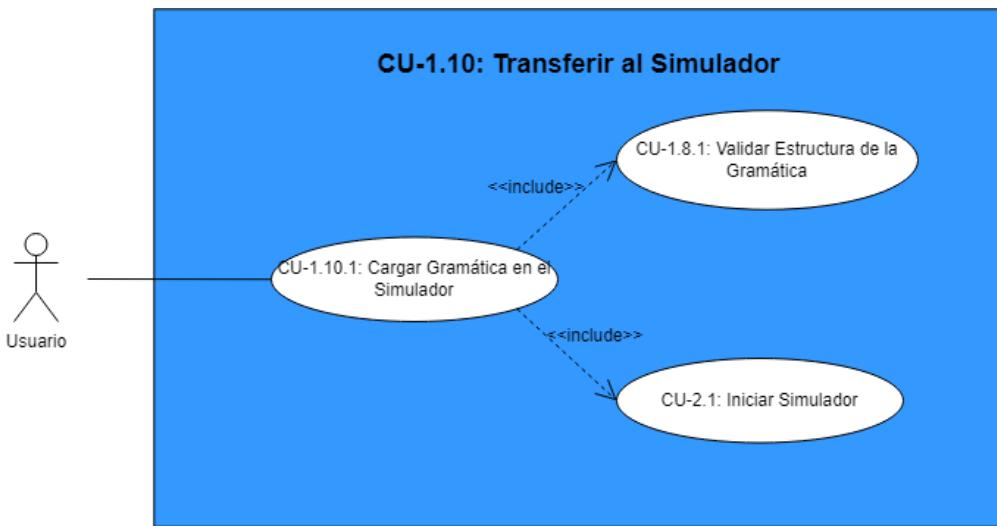


Figura 7.11: CU-1.10: Transferir al simulador

Tabla 7.11: Caso de uso: CU-1.10 Transferir al simulador

Nombre	CU-1.10 Transferir al simulador. Nivel: 2
Descripción	Permite al usuario transferir una gramática al simulador para simular un método de análisis sintáctico.
Actores	Usuario
Casos de uso	<ol style="list-style-type: none"> 1. CU-1.10.1 Cargar gramática en el simulador: <i>carga</i> la gramática en el simulador. 2. CU-1.8.1 Validar estructura de la gramática: <i>valida</i> la estructura de la gramática, formada por el conjunto de producciones y símbolos (<i>terminales</i> y <i>no terminales</i>). 3. CU-2.1. Iniciar el simulador: <i>inicializa</i> el simulador y transfiere la gramática, o simplemente carga la gramática en el simulador (en el caso de que este ya se encuentre inicializado previamente).
continúa en la página siguiente	

Tabla 7.11 – continúa de la página anterior

Flujo de eventos principal	<ol style="list-style-type: none"> 1. Se <i>realiza</i> una validación de la estructura de la gramática, para así comprobar si la gramática es correcta. 2. Se <i>transfiere</i> la gramática al simulador y se inicializa este (en caso de que no se haya iniciado anteriormente al transferir otra gramática)
Flujo de eventos alternativo	<p>Si se produce algún fallo durante el paso 1, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario de que no ha podido validar la gramática. 2. Se <i>termina</i> la acción de transferir al simulador.
Flujo de eventos alternativo	<p>Si se produce algún fallo durante el paso 2, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario de que no se ha podido transferir la gramática. 2. Se <i>termina</i> la acción de transferir al simulador.
Flujo de eventos excepcional	<p>Si durante la ejecución del paso 1, se detecta que la gramática no es válida, puesto que posee errores (esto excluye cualquier error de funcionamiento, descrito en el flujo alternativo anterior), se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>aborta</i> la transferencia de la gramática. 2. Se <i>informa</i> al usuario de que la gramática contiene errores (no es una gramática válida).

El caso de uso *CU-1.10.1 Cargar gramática en el simulador* no se refinará más, puesto que se trata de un proceso simple, que puede ser fácilmente traducido a una implementación posterior sin necesidad de ser refinado.

7.3.2.11. CU-1.11. Obtener gramática

Este caso de uso se encarga de obtener el vocabulario de la gramática (V_N, V_T), así como todo el conjunto de producciones y el símbolo inicial.

Se trata de un proceso bastante sencillo, pero que es utilizado en los demás casos de uso cada vez que se desea obtener la estructura de una gramática. Este caso de uso será

directamente traducido a una implementación sin ser preciso realizar un refinamiento más exhaustivo.

7.3.2.12. CU-1.12. Actualizar visualización

Este caso de uso representa la actualización de la visualización de una gramática. Cabe destacar que este caso de uso no es invocado directamente por el usuario, sino que el propio Sistema actualiza la vista de la gramática automáticamente, a medida que esta sufre alguna modificación (a partir de aquellos casos de uso que incluyen el comportamiento de este caso de uso). En la figura 7.12 se muestra la representación gráfica del caso de uso y en la tabla 7.12 se muestra su representación tabular.

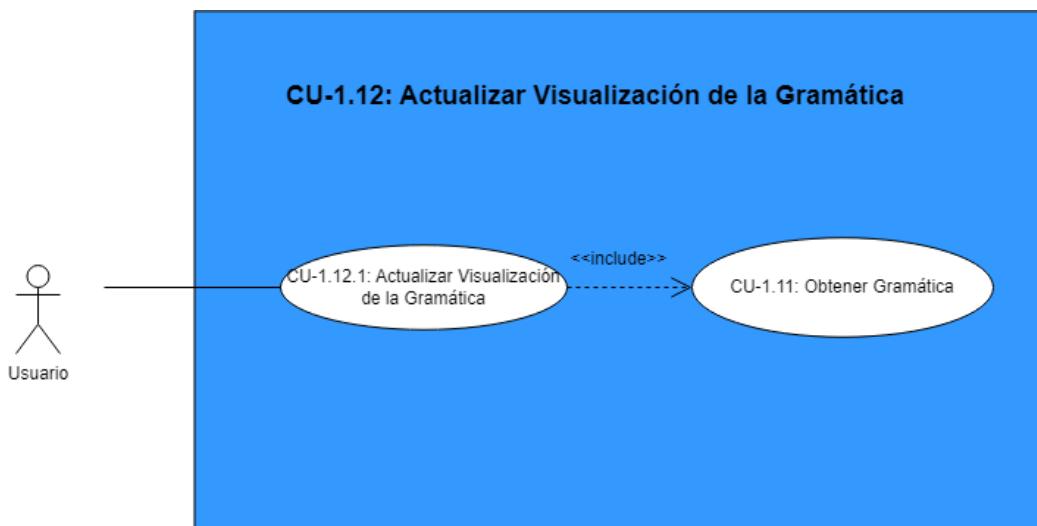


Figura 7.12: CU-1.12: Actualizar visualización de la gramática

Tabla 7.12: Caso de uso: CU-1.12 Actualizar visualización de la gramática

Nombre	CU-1.12 Actualizar visualización de la gramática. Nivel: 2
Descripción	Permite al usuario visualizar la gramática a medida que la va construyendo.
Actores	Usuario
continúa en la página siguiente	

Tabla 7.12 – continúa de la página anterior

Casos de uso	<ol style="list-style-type: none"> 1. CU-1.12.1 Actualizar visualización de la gramática: <i>actualiza</i> la visualización de la gramática en el editor, a medida que esta sufre alguna modificación en su estructura (producciones y símbolos). 2. CU-1.11. gramática: <i>obtiene</i> el vocabulario de símbolos (V_N, V_T) así como las producciones de la gramática.
Flujo de eventos principal	<ol style="list-style-type: none"> 1. Se <i>obtiene</i> el vocabulario de la gramática. 2. Se <i>actualiza</i> la vista de la gramática en el editor de gramáticas.
Flujo de eventos alternativo	<p>Si se produce algún fallo durante el paso 1, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario de que no ha podido obtener el vocabulario de la gramática y que existen errores que impiden visualizarla. 2. Se <i>aborta</i> la actualización de la vista de la gramática.
Flujo de eventos alternativo	<p>Si se produce algún fallo durante el paso 2, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario de que no ha podido actualizar la vista de la gramática. 2. Se <i>aborta</i> la actualización de la vista de la gramática.

El casos de uso *CU-1.12.1 Actualizar visualización de la gramática* no se refinará más, puesto que se trata de un proceso simple, que puede ser fácilmente traducido a una implementación posterior sin necesidad de ser refinado.

7.3.2.13. CU-1.13. Comprobar producción

Este caso de uso se encargará de comprobar si la estructura de una regla gramatical o producción es correcta.

No es necesario refinar de forma profunda este caso de uso ya que es bastante simple y no depende de ningún caso de uso de nivel inferior, por lo que se puede traducir fácilmente sin un mayor refinamiento.

7.3.3. CU-2. Simular análisis sintáctico descendente predictivo

Este caso de uso representa la simulación del análisis sintáctico descendente predictivo con una gramática de contexto libre. En la figura 7.13 se muestra la representación gráfica de este caso de uso, que está formada por los siguientes sub-casos de uso:

- **CU-2.1. Simular Análisis Descendente Predictivo.**
- **CU-2.2. Buscar Gramática.**
- **CU-2.3. Completar Tabla con Funciones de Error.**
- **CU-2.4. Analizar Cadena.**
- **CU-2.5. Seleccionar Velocidad Simulación.**
- **CU-2.6. Construir Árbol Sintáctico.**
- **CU-2.7. Mostrar la derivación.**
- **CU-2.8. Generar Informe Simulación.**

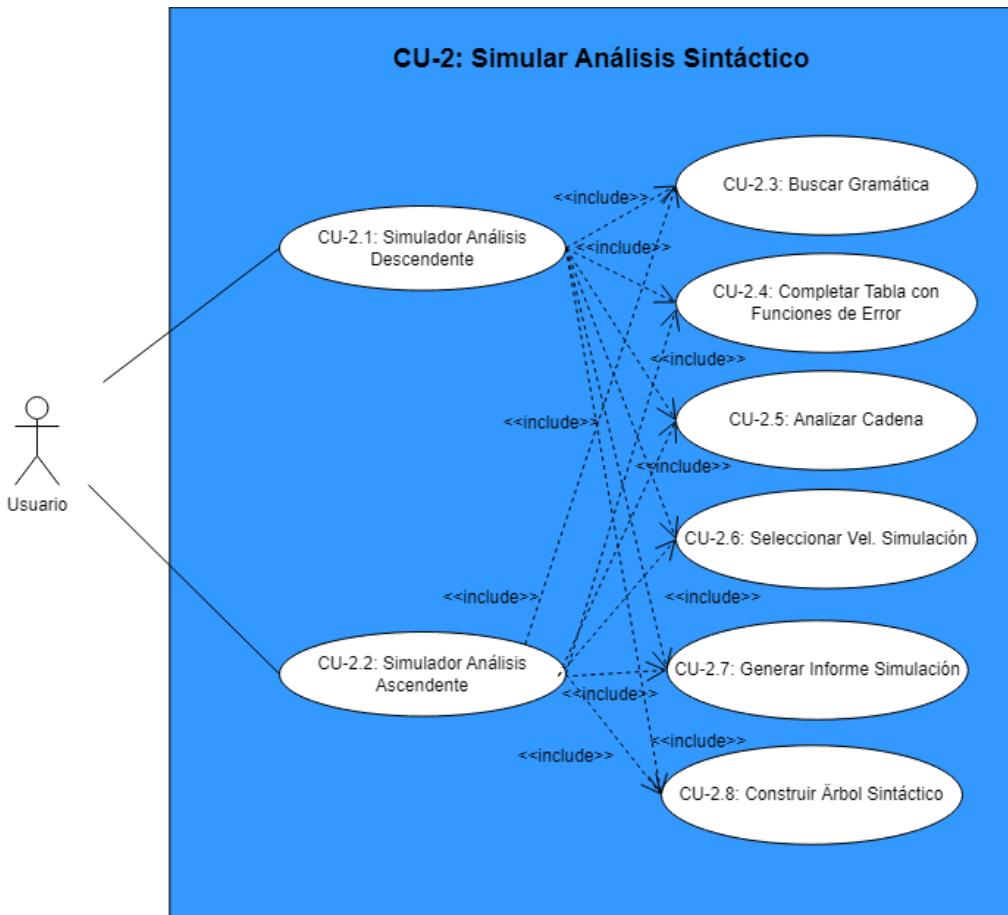


Figura 7.13: CU-2 Simular análisis sintáctico predictivo

La especificación del caso de uso se muestra en la tabla 7.13.

Tabla 7.13: Caso de uso: CU-2 Simular análisis sintáctico

Nombre	CU-2. Simular análisis sintáctico. Nivel: 1
Descripción	Permite al usuario realizar la simulación de un analizador sintáctico usando una gramática.
Actores	Usuario
Casos de uso	<ol style="list-style-type: none"> 1. CU-2.1. Simular Análisis Descendente Predictivo: lleva a cabo el análisis sintáctico descendente predictivo de una gramática de contexto libre. 2. CU-2.2. Buscar gramática: realiza la búsqueda en el sistema de una gramática de contexto libre guardada con anterioridad.
Flujo de eventos principal	<ol style="list-style-type: none"> 1. Se <i>selecciona</i> la gramática que se desea simular. 2. Se <i>realiza</i> el análisis sintáctico descendente predictivo.
Flujo de eventos alternativo	<p>Si se produce un fallo durante el paso 1, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario que se ha podido cargar la gramática seleccionada. 2. Se <i>aborta</i> la selección de la gramática y no se podrá llevar a cabo la simulación de dicha gramática. 3. Se redirecciona a la pantalla de simulación.

A continuación, se refinará cada uno de los casos de uso que componen el caso de uso CU-2.

7.3.3.1. CU-2.1. Simular Análisis Descendente Predictivo

Este caso de uso representa la simulación del análisis sintáctico descendente predictivo de una gramática de contexto libre. En la figura 7.14 se muestra la representación gráfica del caso de uso y en la tabla 7.14 se muestra su representación tabular. Los subcasos de uso que lo conforman son:

- **CU-2.1.1 Gestionar Tabla Predictiva.**

- **CU-2.1.2 Gestionar Conjunto Primero y Siguiente.**

Este caso de uso CU-2.1 también interactúa con los siguientes casos de uso:

- **CU-2.2. Buscar Gramática.**
- **CU-2.3. Completar Tabla con Funciones de Error.**
- **CU-2.4. Analizar Cadena.**
- **CU-2.5. Seleccionar Velocidad Simulación.**
- **CU-2.6. Generar Informe Simulación.**
- **CU-2.7. Construir Árbol Sintáctico.**

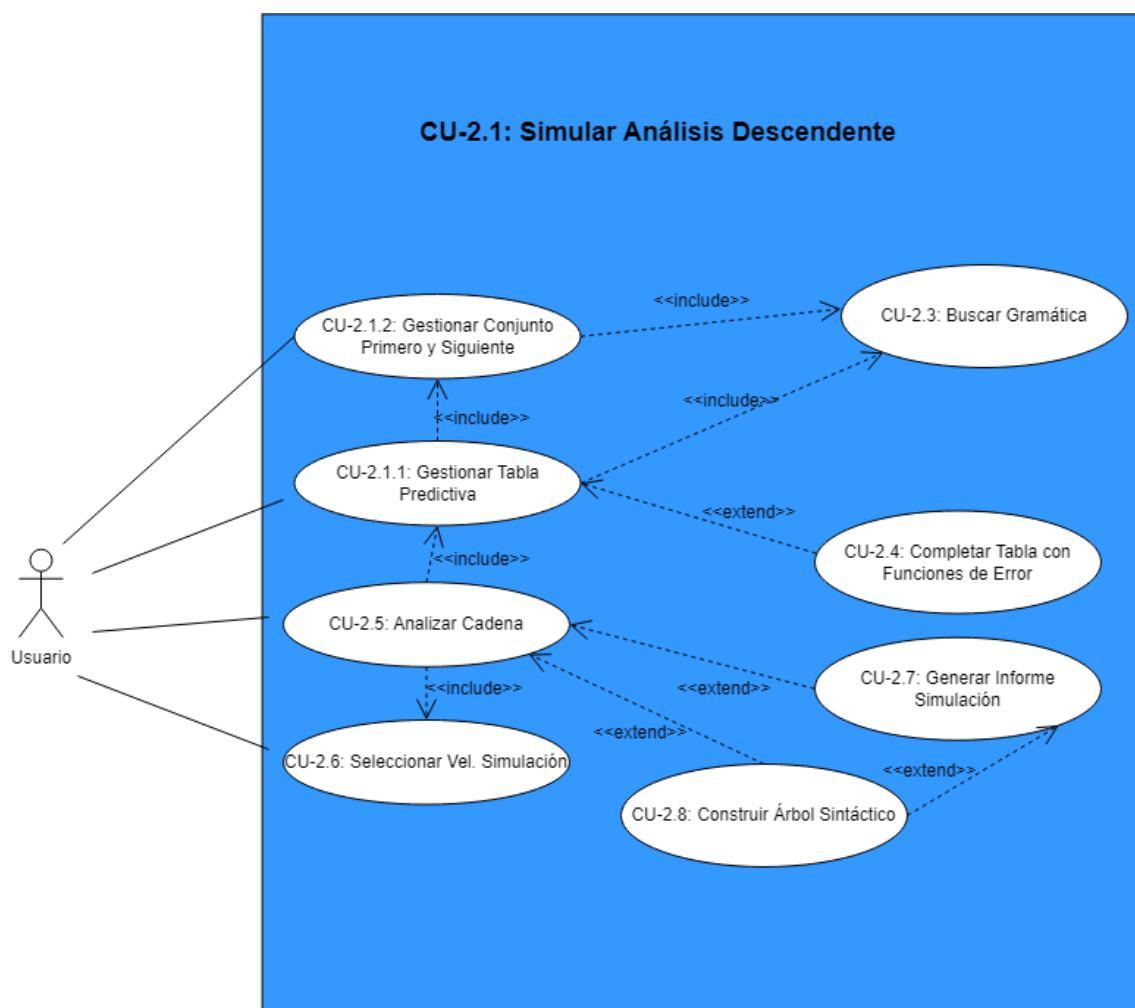


Figura 7.14: CU-2.1 Simular Análisis descendente

Tabla 7.14: Caso de uso: CU-2.1 Simular Análisis descendente predictivo

Nombre	CU-2.1. Simular Análisis descendente. Nivel: 2
Descripción	Permite al usuario realizar la simulación del análisis sintáctico descendente de una gramática de contexto libre.
Actores	Usuario
Casos de uso	<p>1. CU-2.1.1. Gestionar tabla predictiva: <i>gestionar</i> la tabla predictiva necesaria para llevar a cabo la simulación.</p> <p>2. CU-2.3. Buscar gramática: realiza la búsqueda en el sistema de una gramática de contexto libre guardada con anterioridad.</p> <p>3. CU-2.4. Completar tabla con funciones de error: si el usuario lo desea, puede <i>completar</i> la tabla predictiva con funciones para el tratamiento de los errores que se puedan producir.</p> <p>4. CU-2.1.2. Gestionar conjunto primero y siguiente: <i>gestionar</i> el conjunto primero y siguiente de una gramática de contexto libre, necesarios para crear la tabla predictiva.</p> <p>5. CU-2.5. Analizar cadena: <i>realiza</i> el análisis sintáctico descendente predictivo de una cadena.</p> <p>6. CU-2.6. Seleccionar velocidad de simulación: <i>selecciona</i> la velocidad a la que se llevará a cabo la simulación: paso a paso o continua.</p> <p>7. CU-2.7. Generar informe de la simulación: una vez terminado el proceso de simulación, el usuario puede obtener un informe con todos los detalles.</p> <p>8. CU-2.8. Construir Árbol Sintáctico: se construirá el árbol sintáctico de la derivación correspondiente.</p>

Continúa en la página siguiente

Tabla 7.14 – continúa de la página anterior

Flujo de eventos principal	<ol style="list-style-type: none"> 1. Se <i>selecciona</i> la gramática que se desea simular. 2. Se <i>construye</i> el conjunto primero y siguiente. 3. Se <i>crea</i> la tabla predictiva con los datos de los pasos anteriores. 4. Si se desea, se puede <i>completar</i> la tabla predictiva con funciones de tratamiento de errores. 5. Una vez que se tiene todo lo anterior, se procede a analizar una cadena con el simulador. 6. Se <i>elige</i> la velocidad de simulación: paso a paso o continua. 7. Se <i>escoge</i> si se desea construir de forma paralela a la derivación el árbol sintáctico. 8. Se <i>genera</i> el informe de la simulación que se ha llevado a cabo en los pasos anteriores.
Flujo de eventos alternativo	<p>Si se produce un fallo durante el paso 1, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario que no se ha podido cargar la gramática seleccionada. 2. Se <i>aborta</i> la selección de la gramática y no se podrá llevar a cabo la simulación. 3. Se redirecciona a la pantalla de simulación.
Flujo de eventos alternativo	<p>Si se produce un fallo durante el paso 3, se procede como sigue:</p> <ol style="list-style-type: none"> 1. Se <i>informa</i> al usuario que se ha podido crear la tabla predictiva. 2. Se <i>aborta</i> la creación de la tabla predictiva y no se podrá llevar a cabo la simulación hasta que todos los datos sean correctos. 3. Se redirecciona a la pantalla de simulación.

Continúa en la página siguiente

Tabla 7.14 – continúa de la página anterior

Flujo de eventos alternativo	Si se produce un fallo durante el paso 5, se procede como sigue: <ol style="list-style-type: none">1. Se <i>informa</i> al usuario que no se ha podido analizar la cadena introducida.2. Se <i>aborta</i> el análisis y no se podrá llevar a cabo hasta que todos los datos sean correctos.3. Se redirecciona a la pantalla de simulación.
Flujo de eventos alternativo	Si se produce un fallo durante el paso 7, se procede como sigue: <ol style="list-style-type: none">1. Se <i>informa</i> al usuario que no se ha podido generar el informe de la simulación.2. Se <i>aborta</i> la generación del informe y no se podrá obtener el informe.3. Se redirecciona a la pantalla de simulación.

El caso de uso *CU-2.1.1* no se refinará más, puesto que se trata de un proceso simple, que puede ser fácilmente traducido a una implementación posterior sin necesidad de ser refinado.

7.3.3.2. CU-2.1.2. Gestionar conjunto Primero y Siguiente

Este caso de uso es un proceso simple que consiste en crear los conjuntos primero y siguiente de la gramática, necesarios para crear las tablas de análisis en el análisis sintáctico descendente predictivo. Por tanto, este proceso no será refinado de forma más exhaustiva.

7.3.3.3. CU-2.2. Buscar gramática

Este caso de uso es un proceso simple que consiste en seleccionar una gramática de entre las que se encuentran cargadas en el simulador. Por tanto, este proceso no será refinado de forma más exhaustiva.

7.3.3.4. CU-2.3. Completar tabla con funciones de error

Este caso de uso es un proceso simple en el cual, una vez construida la tabla del análisis sintáctico, se completa con la funciones de error. Por tanto, este proceso no será refinado de forma más exhaustiva.

7.3.3.5. CU-2.4. Analizar cadena

Este caso de uso es un proceso simple en el que se lleva a cabo el análisis sintáctico descendente de una cadena. Por tanto, este proceso no será refinado de forma más exhaustiva.

7.3.3.6. CU-2.5. Selecciona velocidad de simulación

Este caso de uso es un proceso simple que consiste en seleccionar entre los dos tipos de velocidad de simulación: paso a paso o continuo. Por tanto, este proceso no será refinado de forma más exhaustiva.

7.3.3.7. CU-2.6. Generar informe de simulación

Este caso de uso es un proceso simple que consiste en generar un informe con todos los datos del análisis sintáctico creados durante la simulación, el cual podrá ver y descargar el usuario. Por tanto, este proceso no será refinado de forma más exhaustiva.

7.3.3.8. CU-2.7. Construir Árbol Sintáctico Descendente

Este caso de uso es un proceso simple que consiste en generar la construcción del árbol sintáctico descendente correspondiente a la derivación que se está realizando en dicho momento. Esta generación del árbol puede ser tanto instantánea como paso a paso y se hará de forma paralela a la propia derivación.

7.3.4. CU-3. Consultar Tutorial

Este caso de uso representa la consulta del tutorial de la aplicación. En la figura 7.15 se muestra la representación gráfica de este caso de uso, que está formada por los siguientes sub-casos de uso:

- CU-3.1. Consultar una lección.
- CU-3.2. Obtener recurso del tutorial.
- CU-3.3. Navegar por el recurso seleccionado.

En este caso de uso se muestran flechas que se dirige desde un caso de uso al usuario. Esto representa el hecho de que el usuario visualiza el tutorial por pantalla.

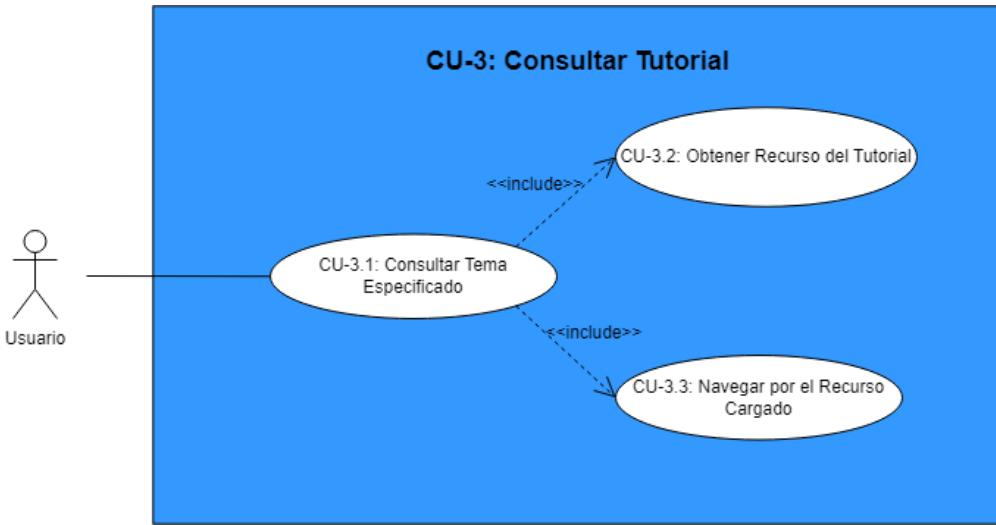


Figura 7.15: CU-3: Consultar Tutorial

La especificación del caso de uso se muestra en la tabla 7.15.

Tabla 7.15: Caso de uso: CU-3. Consultar Tutorial

Nombre	CU-3. Consultar Tutorial. Nivel: 1
Descripción	Permite al usuario consultar el tutorial de la aplicación.
Actores	Usuario
Casos de uso	<ol style="list-style-type: none"> 1. CU-3.1. Consultar una lección: permite <i>consultar</i> una lección del tutorial. 2. CU-3.2. Obtener recurso del tutorial: permite <i>obtener</i> una lección del tutorial. 3. CU-3.3. Navegar por el recurso seleccionado: permite <i>navegar</i> por una lección del tutorial.
Flujo de eventos principal	<ol style="list-style-type: none"> 1. Se <i>obtiene</i> un recurso del tutorial (especificado por el usuario). 2. El usuario <i>navega</i> por el recurso. 3. El usuario <i>accede</i> a otro recurso a través del enlace (para lo que se vuelve al paso 1), o sigue navegando por el mismo.
Continúa en la página siguiente	

Tabla 7.15 – continúa de la página anterior

Flujo de eventos alternativo	Si se produce algún error al <i>cargar</i> o <i>navegar</i> por un recurso del tutorial, se informa al usuario y se restablece el recurso previo (cerrando el tutorial en caso de que se produzca un fallo).
-------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Nótese que lo que aquí se llama recurso del tutorial, es realmente una *lección* del tutorial. El hecho de denominarlo recurso implica que el fichero en cuestión (un recurso del simulador) es buscado y cargado, y, por tanto, contiene una lección del tutorial.

Este caso de uso no es necesario refinarlo exhaustivamente ya que es un proceso sencillo de traducir e implementar, por lo que se realizará sin mayores complicaciones.

7.3.5. CU-4. Consultar Ayuda

Este caso de uso representa la consulta de la ayuda de la aplicación. En la figura 7.16 se muestra la representación gráfica de este caso de uso, que está formada por los siguientes sub-casos de uso:

- **CU-4.1. Consultar un capítulo.**
- **CU-4.2. Obtener recurso de la ayuda.**
- **CU-4.3. Navegar por el recurso seleccionado.**

En este caso de uso se muestran flechas que se dirige desde un caso de uso al usuario. Esto representa el hecho de que el usuario visualiza la ayuda por pantalla.

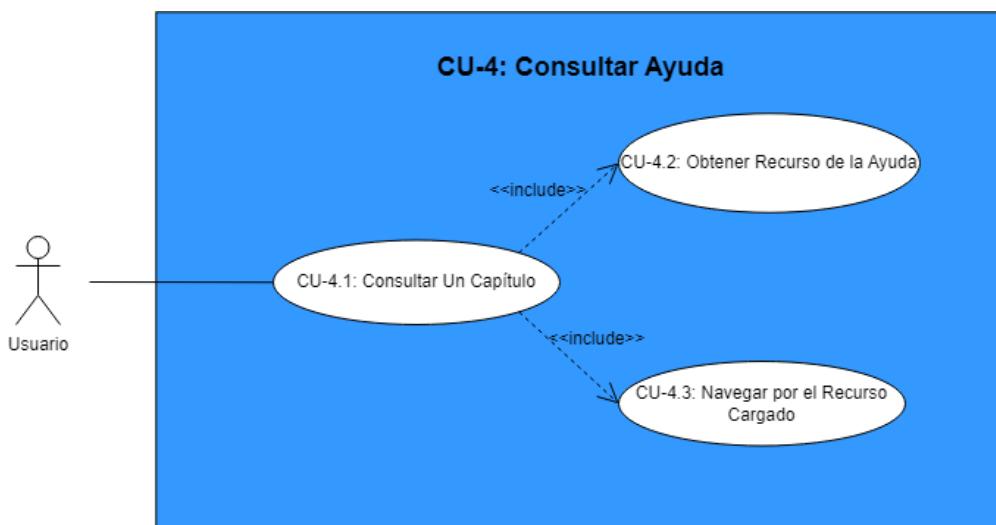


Figura 7.16: CU-4: Consultar Ayuda

La especificación del caso de uso se muestra en la tabla 7.16.

Tabla 7.16: Caso de uso: CU-4. Consultar Ayuda

Nombre	CU-4. Consulta de la ayuda. Nivel: 1
Descripción	Permite al usuario consultar la ayuda de la aplicación.
Actores	Usuario
Casos de uso	<ol style="list-style-type: none">1. CU-4.1. Consultar un capítulo: permite <i>consultar</i> un capítulo de la ayuda.2. CU-4.2. Obtener recurso de la ayuda: permite <i>obtener</i> un capítulo de la ayuda.3. CU-4.3. Navegar por el recurso seleccionado: permite <i>navegar</i> por un capítulo de la ayuda.
Flujo de eventos principal	<ol style="list-style-type: none">1. Se <i>obtiene</i> un recurso de la ayuda (especificado por el usuario).2. El usuario <i>navega</i> por el recurso.3. El usuario <i>accede</i> a otro recurso a través del enlace (para lo que se vuelve al paso 1), o sigue navegando por el mismo.
Flujo de eventos alternativo	Si se produce algún error al <i>cargar</i> o <i>navegar</i> por un recurso de la ayuda, se informa al usuario y se restablece el recurso previo (cerrando la ayuda en caso de que se produzca un fallo).

Nótese que lo que aquí se llama *recurso* de la ayuda (al igual que sucedió con el tutorial), es realmente un *capítulo* de la ayuda. El hecho de nombrarlo *recurso* implica que el fichero en cuestión (un recurso del simulador) es buscado y cargado, y por tanto, contiene un capítulo de la ayuda.

Al igual que en el caso de uso anterior, no es necesario un refinamiento más profundo, ya que es una tarea sencilla de implementar y que no supone una mayor complejidad.

7.4. Validación de los casos de uso

Para asegurar que los casos de uso cumplen con los requisitos funcionales establecidos en el capítulo anterior, se llevará a cabo una validación exhaustiva. Esta validación garantizará que los casos de uso se corresponden adecuadamente con los requisitos del sistema y funcionan de manera esperada en distintos escenarios.

7.4.1. Validación del Módulo de Edición de Gramática

La primera validación que se realizará será del módulo de edición respecto a los requisitos funcionales del sistema. El resultado de esta validación se recoge en la tabla 7.17.

Tabla 7.17: Matriz de correspondencia RF-Casos de uso

		Casos de uso												
		CU-1.1	CU-1.2	CU-1.3	CU-1.4	CU-1.5	CU-1.6	CU-1.7	CU-1.8	CU-1.9	CU-1.10	CU-1.11	CU-1.12	CU-1.13
Requisitos funcionales	RF-E-1	X												
	RF-E-2	X	X											
	RF-E-3			X										
	RF-E-4			X										
	RF-E-5			X										
	RF-E-6				X									
	RF-E-7							X						
	RF-E-8											X		
	RF-E-9									X				
	RF-E-10					X								
	RF-E-11							X						
	RF-E-12						X						X	
	RF-E-13								X					
	RF-E-14								X					X
	RF-E-15										X			

7.4.2. Validación del Módulo de Simulación de Gramática

La segunda validación que se realizará será del módulo de simulación respecto a los requisitos funcionales del sistema. El resultado de esta validación se recoge en la tabla 7.18.

Tabla 7.18: Matriz de correspondencia RF-Casos de uso

		Casos de uso						
		CU-2.1	CU-2.2	CU-2.3	CU-2.4	CU-2.5	CU-2.6	CU-2.7
Requisitos funcionales	RF-S-1	X	X					
	RF-S-2	X	X					
	RF-S-3				X			
	RF-S-4				X			
	RF-S-5				X			
	RF-S-6				X			
	RF-S-7					X		
	RF-S-8					X		
	RF-S-9						X	
	RF-S-10						X	
	RF-S-11						X	
	RF-S-12							X
	RF-S-13				X			
	RF-S-14							X
	RF-S-15							X
	RF-S-16			X				X
	RF-S-17							X
	RF-S-18							X

7.4.3. Validación del Módulo del Tutorial

La matriz de validación de este Módulo se recoge en la tabla 7.19.

Tabla 7.19: Matriz de correspondencia RF-CU, Módulo tutorial

		Casos de uso		
		CU-3.1	CU-3.2	CU-3.3
R. funcionales	RF-T-1	X		
	RF-T-2		X	X
	RF-T-3		X	

7.4.4. Validación del Módulo de la Ayuda

La matriz de validación de este Módulo se recoge en la tabla 7.20.

7.4. VALIDACIÓN DE LOS CASOS DE USO

Tabla 7.20: Matriz de correspondencia RF-CU, Módulo ayuda

		Casos de uso		
		CU-4.1	CU-4.2	CU-4.3
R. funcionales	RF-A-1	X		
	RF-A-2		X	X
	RF-A-3		X	

Parte III

Diseño

Capítulo 8

Diseño de paquetes

8.1. Introducción

Este capítulo presenta la arquitectura de paquetes de SimAS 3.0, analizando cómo se organiza el sistema en módulos funcionales que agrupan clases por cercanía semántica y responsabilidad. La arquitectura modular implementada sigue principios de diseño orientado a objetos, facilitando el mantenimiento, la reutilización de código y la evolución del sistema.

8.1.1. Arquitectura general del sistema

SimAS 3.0 está estructurado siguiendo una arquitectura *modular por capas* con separación clara de responsabilidades [16]:

- **Capa de Presentación** (*bienvenida, vistas*): gestiona la interfaz de usuario y la interacción con el usuario final.
- **Capa de Lógica de Negocio** (*editor, simulador*): implementa la funcionalidad principal de edición y simulación.
- **Capa de Modelo de Datos** (*gramatica*): representa las estructuras de datos fundamentales del dominio.
- **Capa de Servicios Transversales** (*utils*): proporciona funcionalidades comunes y utilitarias.
- **Capa de Recursos** (*resources, centroayuda*): contiene recursos estáticos y documentación.

8.1.2. Principios de diseño aplicados

La arquitectura de paquetes de SimAS 3.0 se fundamenta en los siguientes principios de diseño:

1. **Separación de responsabilidades:** cada paquete tiene una responsabilidad única y bien definida.
2. **Acoplamiento bajo:** las dependencias entre paquetes están claramente definidas y minimizadas.
3. **Cohesión alta:** las clases dentro de cada paquete comparten una fuerte relación funcional.
4. **Abstracción:** cada paquete expone una interfaz clara y oculta su implementación interna.
5. **Reutilización:** los paquetes están diseñados para ser reutilizables en diferentes contextos.

8.1.3. Métricas generales del sistema

Con el conocimiento detallado de las 39 clases documentadas en el Capítulo 9, podemos proporcionar métricas precisas sobre la complejidad del sistema:

- **Total de clases Java:** 39 clases distribuidas en 6 paquetes principales
- **Paquete más complejo:** *simulador* (13 clases - 33 % del total)
- **Paquete más simple:** *bienvenida* (2 clases - 5 % del total)
- **Relaciones de dependencia:** 15 relaciones principales entre paquetes
- **Patrones de diseño utilizados:** MVC [8, 22], Strategy [7, 25], Factory [28, 18], Observer [6, 23], Singleton [3, 24]

8.2. Especificación detallada de los paquetes

SimAS 3.0 se organiza en 8 paquetes principales que encapsulan diferentes aspectos funcionales del sistema. A continuación se detalla cada paquete con sus clases específicas y responsabilidades.

8.2.1. Paquete bienvenida

El paquete **bienvenida** implementa la *capa de presentación inicial* del sistema, actuando como punto de entrada principal de SimAS 3.0. Este paquete es fundamental para la experiencia del usuario, proporcionando una interfaz intuitiva y profesional.

8.2.1.1. Responsabilidades principales

- Gestión completa del ciclo de vida inicial de la aplicación
- Navegación centralizada hacia módulos funcionales
- Coordinación entre componentes de la interfaz principal
- Gestión de estados globales de la aplicación

8.2.1.2. Clases del paquete

- **Bienvenida.java** (extends Application): clase principal que implementa el patrón Singleton para garantizar una única instancia de bienvenida. Gestiona la pantalla inicial, configuración regional, y transición hacia el menú principal. Implementa ActualizableTextos para soporte de internacionalización.
- **MenuPrincipal.java** (extends Application): controlador principal del menú de navegación. Coordina el acceso a todas las funcionalidades del sistema (editor, simulador, ayuda, configuración). Gestiona la creación y configuración de ventanas secundarias, implementando el patrón Facade para simplificar el acceso a módulos complejos.

8.2.1.3. Características técnicas

- **Complejidad:** 2 clases (5 % del total del sistema).
- **Patrones implementados:** Singleton, Facade, MVC (Vista-Controlador).
- **Dependencias:** utils (internacionalización), resources (iconos), vistas (FXML).
- **Responsabilidad:** Punto de entrada único y navegación centralizada.

8.2.1.4. Flujo de ejecución

El flujo típico de este paquete sigue el patrón: Bienvenida → Configuración regional → MenuPrincipal → Navegación modular.

Este paquete establece la base para la experiencia del usuario, asegurando una transición fluida desde el lanzamiento de la aplicación hasta los módulos especializados de edición y simulación.

8.2.2. Paquete editor

El paquete **editor** implementa el *núcleo funcional de edición de gramáticas* de Si-mAS 3.0, proporcionando una interfaz completa para la creación, modificación y gestión de gramáticas de contexto libre. Este paquete representa el componente más complejo del sistema, con 10 clases que implementan un asistente paso a paso altamente sofisticado.

8.2.2.1. Arquitectura y diseño

El paquete sigue una arquitectura *modular jerárquica* organizada en tres niveles:

1. **Nivel de Control Principal:** editor y EditorWindow (gestión global).
2. **Nivel de Coordinación:** PanelCreacionGramatica (orquestación del asistente).
3. **Nivel de Especialización:** paneles específicos por funcionalidad.

8.2.2.2. Patrones de diseño implementados

- **MVC (Model-View-Controller):** separación clara entre modelo (*gramatica*), vista (FXML) y controlador (clases del editor).
- **Strategy:** los diferentes pasos del asistente implementan estrategias específicas.
- **Observer:** comunicación reactiva entre paneles y el panel padre.
- **Factory:** creación dinámica de paneles según el paso del asistente.
- **Composite:** PanelCreacionGramatica como contenedor de paneles especializados.

8.2.2.3. Clases del paquete por funcionalidad

Control principal

- **Editor.java** (extends VBox, implements ActualizableTextos): controlador principal del editor. Gestiona la gramática activa, coordina todos los paneles del asistente, y maneja la persistencia de datos. Implementa el patrón Mediator para coordinar la comunicación entre paneles.
- **EditorWindow.java:** gestor de la ventana física del editor. Maneja la creación del Stage, configuración de dimensiones, y gestión del ciclo de vida de la ventana.

Asistente de creación

- **PanelCreacionGramatica.java** (extends BorderPane, implements ActualizableTextos): orquestador principal del asistente. Implementa el patrón State para gestionar la transición entre pasos, valida la consistencia de la gramática, y coordina la navegación entre paneles.
- **PanelCreacionGramaticaPaso1.java** (extends VBox, implements ActualizableTextos): primer paso - configuración básica (nombre, descripción).
- **PanelCreacionGramaticaPaso2.java** (extends VBox, implements ActualizableTextos): segundo paso - gestión de símbolos terminales y no terminales.

- **PanelCreacionGramaticaPaso3.java** (extends VBox, implements ActualizableTextos): tercer paso - definición de producciones.
- **PanelCreacionGramaticaPaso4.java** (extends VBox, implements ActualizableTextos): cuarto paso - validación y finalización.

Paneles especializados

- **PanelProducciones.java** (extends VBox, implements ActualizableTextos): panel especializado para la gestión detallada de reglas de producción. Implementa el patrón Command para operaciones CRUD sobre producciones.
- **PanelSimbolosNoTerminales.java** (extends VBox, implements ActualizableTextos): panel para gestión de símbolos no terminales con validación automática y sugerencias predefinidas.
- **PanelSimbolosTerminales.java** (extends VBox, implements ActualizableTextos): panel para gestión de símbolos terminales con soporte para expresiones regulares y validación de sintaxis.

8.2.2.4. Características técnicas

- **Complejidad:** 10 clases (26 % del total del sistema).
- **Patrones principales:** MVC, Strategy, Observer, Mediator, Factory.
- **Interfaz de usuario:** 7 archivos FXML dedicados.
- **Internacionalización:** Soporte completo en 6 idiomas.
- **Validación:** Validación en tiempo real y feedback contextual.

8.2.2.5. Dependencias y relaciones

- **Dependencias principales:**
 - *gramatica*: modelo de datos fundamental (uso intensivo).
 - *utils*: internacionalización y utilidades de UI.
 - *vistas*: definiciones FXML de la interfaz.
 - *resources*: iconos y recursos gráficos.
- **Relaciones con otros paquetes:**
 - **simulador**: comparte conceptos de gramática y validación.
 - **bienvenida**: integración con menú principal.
 - **centroayuda**: contexto de ayuda integrado.

8.2.2.6. Flujo de trabajo del asistente

El asistente implementa un **patrón wizard secuencial** con 4 pasos principales:

1. **Configuración inicial**: nombre, descripción y preparación.
2. **Definición de símbolos**: terminales y no terminales con validación.
3. **Producciones**: reglas de derivación con verificación sintáctica.
4. **Finalización**: validación global y generación de tabla predictiva.

Este paquete representa el corazón de la funcionalidad de SimAS 3.0, proporcionando una experiencia de usuario intuitiva para la creación de gramáticas complejas mediante un asistente paso a paso altamente sofisticado.

8.2.3. Paquete simulador

El paquete **simulador** implementa el *motor de análisis sintáctico descendente predictivo* más complejo y sofisticado de SimAS 3.0. Con 13 clases especializadas, este paquete representa el 33 % de toda la funcionalidad del sistema, implementando algoritmos avanzados de análisis sintáctico con manejo completo de errores.

8.2.3.1. Arquitectura y diseño

El paquete sigue una arquitectura *por capas funcionales* organizada en tres niveles principales:

1. **Nivel de Control y Coordinación**: PanelSimuladorDesc (orquestación global)
2. **Nivel de Especialización por Paso**: PanelNuevaSimDescPaso1-6 (lógica específica de cada fase)
3. **Nivel de Utilidades y Soporte**: componentes auxiliares y controladores especializados

8.2.3.2. Algoritmo implementado

El simulador implementa el *algoritmo completo de análisis sintáctico descendente predictivo* siguiendo estos pasos:

1. **Eliminación de recursividad**: transformación de gramáticas recursivas.
2. **Factorización**: eliminación de ambigüedades por factorización.
3. **Cálculo de conjuntos**: Primero y Siguiente para cada símbolo.

4. **Construcción de tabla predictiva:** matriz de análisis sintáctico.
5. **Configuración de errores:** funciones de recuperación de errores.
6. **Simulación interactiva:** análisis paso a paso con visualización.

8.2.3.3. Patrones de diseño implementados

- **Template Method:** los pasos del asistente siguen un patrón común con especializaciones.
- **State:** transición controlada entre los 6 pasos del asistente.
- **Strategy:** diferentes estrategias de análisis según la configuración.
- **Observer:** actualización reactiva de la interfaz durante la simulación.
- **Command:** operaciones de simulación como comandos reversibles.
- **Bridge:** separación entre lógica de análisis y presentación.

8.2.3.4. Clases del paquete por funcionalidad

Coordinación principal

- **PanelSimuladorDesc.java:** orquestador principal del simulador. Implementa el patrón Mediator para coordinar los 6 pasos del asistente, gestionar el estado global de la simulación, y proporcionar persistencia de la tabla predictiva extendida.

Pasos del asistente

- **PanelNuevaSimDescPaso1.java** (implements PanelNuevaSimDescPaso, ActualizableTextos): paso 1 - visualización y transformación de la gramática original. Aplica eliminación de recursividad y factorización automática.
- **PanelNuevaSimDescPaso2.java** (implements PanelNuevaSimDescPaso, ActualizableTextos): paso 2 - cálculo y visualización de conjuntos Primero y Siguiente. Implementa algoritmos iterativos para el cálculo de estos conjuntos fundamentales.
- **PanelNuevaSimDescPaso3.java** (implements PanelNuevaSimDescPaso, ActualizableTextos): paso 3 - construcción de la tabla predictiva básica. Genera la matriz de análisis sintáctico a partir de los conjuntos calculados.
- **PanelNuevaSimDescPaso4.java** (implements PanelNuevaSimDescPaso, ActualizableTextos): paso 4 - gestión de funciones de error. Permite definir estrategias de recuperación de errores personalizadas.
- **PanelNuevaSimDescPaso5.java** (implements PanelNuevaSimDescPaso, ActualizableTextos): paso 5 - configuración de tabla predictiva extendida. Permite editar manualmente la tabla para añadir funciones de error específicas.

- **PanelNuevaSimDescPaso6.java** (extends BorderPane, implements PanelNuevaSimDescPaso, ActualizableTextos): paso 6 - configuración final y preparación para simulación. Valida la completitud de la configuración y genera informes.

Componentes auxiliares

- **PanelNuevaSimDescPaso.java**: interfaz común para todos los pasos del asistente. Define el contrato estándar para la navegación y gestión de estado.
- **PanelGramaticaOriginal.java** (extends VBox, implements ActualizableTextos): panel especializado para mostrar la gramática original en una pestaña separada con formato optimizado.
- **NuevaFuncionError.java** (implements ActualizableTextos): panel modal para la creación y edición de funciones de error. Implementa validación automática y sugerencias inteligentes.
- **EditorCadenaEntradaController.java**: controlador especializado para la edición de cadenas de entrada con soporte para símbolos terminales predefinidos y validación sintáctica.

Simulación final

- **SimulacionFinal.java** (extends BorderPane, implements ActualizableTextos): motor de simulación interactiva. Implementa el algoritmo de análisis sintáctico descendente con pila, visualización paso a paso, navegación bidireccional, y generación de informes detallados.
- **PanelSimulacion.java** (extends VBox): panel básico para simulación con visualización de pila, entrada, y resultados. Proporciona una interfaz simplificada para demostraciones.

8.2.3.5. Características técnicas avanzadas

- **Complejidad algorítmica**: implementa algoritmos $O(n^3)$ para cálculo de conjuntos y $O(n^2)$ para construcción de tabla.
- **Manejo de errores**: sistema completo de recuperación de errores con 7 tipos de funciones predefinidas.
- **Persistencia**: mantiene el estado de la tabla predictiva extendida entre sesiones.
- **Visualización**: tres tipos de vistas: básica, avanzada, y con árbol sintáctico.
- **Internacionalización**: soporte completo en 6 idiomas con terminología técnica precisa.
- **Navegación**: sistema de navegación bidireccional (adelante/atrás) durante la simulación.

8.2.3.6. Dependencias críticas

- **Dependencia fundamental:** *gramatica* - utiliza todas las clases del modelo de datos.
- **Dependencias técnicas:**
 - *utils*: TabManager, SecondaryWindow, internacionalización.
 - *vistas*: 6 archivos FXML dedicados al asistente.
 - *resources*: iconos específicos para simulación.

8.2.3.7. Flujo de análisis sintáctico

El paquete implementa el *ciclo completo de análisis sintáctico*:

1. **Preparación:** eliminación de recursividad y factorización.
2. **Análisis léxico:** cálculo de conjuntos Primero y Siguiente.
3. **Construcción:** generación de tabla predictiva.
4. **Configuración:** definición de estrategias de error.
5. **Ejecución:** análisis descendente con pila.
6. **Recuperación:** aplicación de funciones de error cuando es necesario.

Este paquete representa el componente más sofisticado de SimAS 3.0, implementando algoritmos de análisis sintáctico a nivel académico con una interfaz de usuario que facilita el entendimiento de conceptos complejos del análisis de lenguajes formales.

8.2.4. Paquete **gramatica**

El paquete **gramatica** implementa el *modelo de datos fundamental* de SimAS 3.0, representando todas las estructuras matemáticas y algorítmicas necesarias para el análisis de lenguajes formales. Con 8 clases especializadas, este paquete establece la base teórica sobre la cual se construye toda la funcionalidad del sistema.

8.2.4.1. Arquitectura del modelo de datos

El paquete sigue una arquitectura *jerárquica por composición* organizada en tres niveles conceptuales:

1. **Nivel de abstracción máxima:** Gramatica (representación completa).
2. **Nivel de componentes:** Produccion, Simbolo y sus especializaciones.
3. **Nivel de algoritmos:** TablaPredictiva y componentes asociados.

8.2.4.2. Patrones de diseño implementados

- **Composite:** estructura jerárquica Gramatica → Produccion → Simbolo.
- **Factory:** creación especializada de símbolos terminales y no terminales.
- **Strategy:** diferentes algoritmos para construcción de tabla predictiva.
- **Template Method:** algoritmos comunes para cálculo de conjuntos.
- **Observer:** notificación de cambios en la estructura de la gramática.

8.2.4.3. Clases del paquete por funcionalidad

Representación de gramáticas

- **Gramatica.java:** clase central del modelo. Implementa el patrón Singleton para gestión de instancia única, coordina todas las operaciones sobre la gramática, y proporciona métodos para transformación automática (eliminación de recursividad, factorización). Gestiona la persistencia y validación global de la gramática.

Estructura de producciones

- **Produccion.java:** representa una regla de producción completa. Implementa el patrón Composite con Antecedente y Consecuente, maneja numeración automática de producciones, y proporciona métodos de validación sintáctica.
- **Antecedente.java:** modelo matemático del lado izquierdo de una producción. Gestiona un único símbolo no terminal con métodos de comparación e igualdad.
- **Consecuente.java:** modelo matemático del lado derecho de una producción. Maneja una lista ordenada de símbolos con operaciones de manipulación eficientes.

Jerarquía de símbolos

- **Simbolo.java:** clase abstracta base que define la interfaz común para todos los símbolos. Implementa comparación, hashing, y métodos de representación.
- **Terminal.java** (extends Simbolo): representa símbolos terminales con métodos específicos para análisis léxico y comparación de tipos.
- **NoTerminal.java** (extends Simbolo): representa símbolos no terminales con gestión de conjuntos Primero y Siguiiente, algoritmos de cálculo iterativo, y métodos de comparación específicos.

Algoritmos de análisis

- **TablaPredictiva.java**: implementa la tabla predictiva básica. Gestiona la matriz de análisis sintáctico, coordina el cálculo de conjuntos, y proporciona métodos de consulta eficientes. Implementa el patrón Strategy para diferentes algoritmos de construcción.
- **FilaTablaPredictiva.java**: representa una fila completa de la tabla predictiva. Gestiona el mapeo entre símbolos y producciones, soporta funciones de error, y proporciona métodos de manipulación de celdas.
- **TablaPredictivaPaso5.java** (extends TablaPredictiva): extensión especializada para el manejo de funciones de error. Implementa algoritmos avanzados de recuperación de errores y gestión de conflictos en la tabla.
- **FuncionError.java**: modelo de funciones de recuperación de errores. Define 7 tipos de acciones (insertar, borrar, modificar) con parámetros configurables y métodos de ejecución automática.

8.2.4.4. Algoritmos implementados

El paquete implementa algoritmos fundamentales de la teoría de lenguajes formales:

- **Eliminación de recursividad**: algoritmo iterativo para transformación de gramáticas.
- **Factorización**: algoritmo de detección y resolución de ambigüedades.
- **Cálculo de conjuntos Primero**: algoritmo recursivo con memorización.
- **Cálculo de conjuntos Siguiente**: algoritmo iterativo con convergencia.
- **Construcción de tabla predictiva**: algoritmo matricial $O(n^2)$.
- **Resolución de conflictos**: estrategias automáticas para entradas múltiples.

8.2.4.5. Características técnicas

- **Complejidad**: 8 clases (21 % del total del sistema).
- **Patrones principales**: Composite, Factory, Strategy, Template Method.
- **Estructuras de datos**: Listas, mapas y matrices optimizadas.
- **Algoritmos**: Complejidad desde $O(n)$ hasta $O(n^3)$ según la operación.
- **Persistencia**: Soporte completo para serialización y carga de gramáticas.
- **Validación**: Validación automática en tiempo real de consistencia gramatical.

8.2.4.6. Relaciones críticas con el sistema

- **Dependencia universal:** todos los paquetes principales (*editor, simulador*) dependen críticamente de este paquete.
- **Interfaz pública:** expone una API completa para manipulación de gramáticas.
- **Extensibilidad:** diseño que permite añadir nuevos tipos de símbolos y algoritmos.
- **Consistencia:** garantiza la integridad matemática de todas las operaciones.

8.2.4.7. Flujo de transformación gramatical

El paquete gestiona el *ciclo completo de transformación de gramáticas*:

1. **Entrada:** gramática en formato texto o cargada desde archivo.
2. **Análisis:** parsing y construcción de estructura de objetos.
3. **Transformación:** eliminación de recursividad y factorización automática.
4. **Análisis:** cálculo de conjuntos Primero y Siguiente.
5. **Construcción:** generación de tabla predictiva.
6. **Validación:** verificación de completitud y corrección.
7. **Exportación:** serialización para uso en simulador.

Este paquete representa la *base teórica sólida* de SimAS 3.0, implementando con precisión matemática los conceptos fundamentales del análisis sintáctico descendente predictivo y proporcionando una API robusta para su manipulación algorítmica.

8.2.5. Paquete centroayuda

El paquete **centroayuda** implementa el *sistema de documentación y soporte al usuario* de SimAS 3.0. Este paquete proporciona una experiencia de ayuda integral que combina documentación técnica, tutoriales interactivos, y referencias teóricas del análisis sintáctico.

8.2.5.1. Arquitectura del sistema de ayuda

El paquete sigue una arquitectura *modular por contenido* organizada en tres componentes principales:

1. **Componente de interfaz:** AcercaDe.java para información del sistema.
2. **Componente de contenido:** documentación HTML y PDF estructurada.
3. **Componente multimedia:** imágenes, diagramas y recursos visuales.

8.2.5.2. Contenido del sistema de ayuda

- **AcercaDe.java:** ventana modal que muestra información del sistema (versión, autores, institución). Implementa el patrón Singleton para garantizar una única instancia y proporciona información dinámica sobre la configuración actual.
- **ayuda.html:** archivo principal de documentación que estructura toda la ayuda de la aplicación. Implementa navegación jerárquica con secciones para introducción, tutoriales, y referencia técnica.
- **SimAS.html:** documentación específica de la aplicación con guías detalladas de uso, ejemplos prácticos, y solución de problemas comunes.
- **Tema_1.pdf a Tema_5.pdf:** documentos PDF especializados que cubren aspectos teóricos del análisis sintáctico:
 - Tema 1: Conceptos básicos de gramáticas formales.
 - Tema 2: Análisis sintáctico descendente.
 - Tema 3: Tablas predictivas y algoritmos.
 - Tema 4: Funciones de error y recuperación.
 - Tema 5: Casos prácticos avanzados.
- **imagenes/**: directorio multimedia con 90 archivos de imagen organizados por categorías:
 - Diagramas de flujo de algoritmos.
 - Ejemplos visuales de gramáticas.
 - Capturas de pantalla de la interfaz.
 - Ilustraciones de conceptos teóricos.
 - Gráficos de árboles sintácticos.

8.2.5.3. Características técnicas

- **Estructura jerárquica:** navegación por secciones con índices detallados.
- **Multimedia integrada:** combinación de texto, imágenes y diagramas.
- **Acceso contextual:** ayuda específica según el contexto de uso.
- **Internacionalización:** soporte para documentación en múltiples idiomas.
- **Actualización:** sistema de versionado para mantener documentación actualizada.

8.2.5.4. Relaciones con el sistema

- **Integración con módulos principales:** acceso desde *editor* y *simulador*.
- **Contexto inteligente:** ayuda específica según la operación en curso.
- **Complementariedad:** trabaja junto con el sistema de internacionalización.

Este paquete proporciona una *experiencia de aprendizaje integral* que facilita la comprensión tanto de la herramienta como de los conceptos teóricos del análisis sintáctico, siendo esencial para usuarios principiantes y avanzados.

8.2.6. Paquete resources

El paquete **resources** implementa el *sistema de recursos visuales centralizados* de SimAS 3.0, proporcionando todos los elementos gráficos necesarios para crear una interfaz de usuario coherente, profesional y accesible. Este paquete garantiza la consistencia visual en toda la aplicación.

8.2.6.1. Arquitectura de recursos

El paquete sigue una organización *jerárquica por funcionalidad* con categorización clara de recursos:

1. **Recursos funcionales:** iconos de acciones y navegación.
2. **Recursos de dominio:** elementos específicos del análisis sintáctico.
3. **Recursos de internacionalización:** banderas de idiomas.
4. **Recursos de identidad:** logotipos e imagen corporativa.

8.2.6.2. Categorización detallada de recursos

Iconos de acción (funcionales)

- **Operaciones CRUD:** abrir.png, guardar.png, nuevo.png, eliminar.png, editar.png.
- **Controles de diálogo:** aceptar.png, cancelar.png, aplicar.png, cerrar.png.
- **Navegación:** anterior.png, siguiente.png, inicio.png, fin.png.
- **Sistema:** configuracion.png, ayuda.png, informacion.png.

Iconos de navegación

- **Direccionales:** flecha-izquierda.png, flecha-derecha.png, flecha-arriba.png, flecha-abajo.png.
- **Asistentes:** paso-anterior.png, paso-siguiente.png, finalizar.png.
- **Jerarquía:** expandir.png, contraer.png, nivel-superior.png.

Iconos específicos de simulación

- **Análisis sintáctico:** pila.png, entrada.png, produccion.png, derivacion.png.
- **Estados:** exito.png, error.png, advertencia.png, procesamiento.png.
- **Funciones:** prediccion.png, matching.png, reduccion.png.

Recursos de internacionalización

- **Banderas nacionales:** españa.png, reino-unido.png, francia.png, alemania.png, portugal.png, japon.png.
- **Indicadores lingüísticos:** soporte visual para 6 idiomas disponibles.

Elementos de identidad corporativa

- **Logo principal:** simas-logo.png (versión completa).
- **Icono de aplicación:** simas-icon.png (versión reducida para barras de título).
- **Universidad:** uco-logo.png (logotipo institucional).

Recursos simbólicos del dominio

- **Símbolos gramaticales:** terminal.png, no-terminal.png, epsilon.png, fin-cadena.png.
- **Elementos matemáticos:** produccion.png, antecedente.png, consecuente.png.
- **Construcciones teóricas:** automata.png, arbol.png, grafo.png.

8.2.6.3. Directorio icons/ especializado

El subdirectorio **icons/** contiene 24 iconos adicionales organizados por especialización:

- **Iconos de validación:** 6 iconos para estados de validación (válido, inválido, pendiente, etc.).

- **Iconos de tipos:** 8 iconos para diferentes tipos de elementos (texto, número, booleano, etc.).
- **Iconos de estado:** 10 iconos para estados del sistema (cargando, procesando, completado, etc.).

8.2.6.4. Características técnicas

- **Formato estandarizado:** todos los iconos en formato PNG con transparencias.
- **Resolución optimizada:** tamaños estándar (16x16, 24x24, 32x32 píxeles).
- **Paleta de colores:** esquema consistente con la interfaz general.
- **Accesibilidad:** contraste adecuado y significado claro.
- **Escalabilidad:** diseño vectorial base con exportación a múltiples resoluciones.

8.2.6.5. Sistema de carga y gestión

- **Carga diferida:** recursos cargados bajo demanda para optimizar memoria.
- **Cache inteligente:** sistema de caché para recursos frecuentemente utilizados.
- **Gestión de errores:** manejo robusto de recursos faltantes o corruptos.
- **Internacionalización:** asociación automática de recursos según idioma.

8.2.6.6. Integración con el sistema

- **Dependencia universal:** utilizado por todos los paquetes con interfaz gráfica.
- **Consistencia garantizada:** framework centralizado para acceso a recursos.
- **Mantenibilidad:** actualización centralizada de recursos visuales.
- **Extensibilidad:** fácil adición de nuevos recursos siguiendo convenciones.

Este paquete establece los *cimientos visuales* de SimAS 3.0, garantizando una experiencia de usuario coherente y profesional mediante un sistema de recursos gráficos bien organizado y eficientemente gestionado.

8.2.7. Paquete utils

El paquete **utils** implementa el *framework de servicios transversales* más sofisticado de SimAS 3.0, proporcionando 6 clases especializadas que ofrecen funcionalidades críticas de internacionalización, gestión de interfaz y control de navegación. Este paquete representa el 15 % de la funcionalidad total del sistema.

8.2.7.1. Arquitectura de servicios transversales

El paquete sigue una organización *por capas de abstracción* que separa claramente las responsabilidades:

1. **Capa de internacionalización:** ActualizableTextos, LanguageItem, LanguageListCell.
2. **Capa de gestión de interfaz:** SecondaryWindow, TabManager, TabPaneMonitor.
3. **Capa de recursos:** archivos de propiedades lingüísticas.

8.2.7.2. Patrones de diseño implementados

- **Observer:** sistema de notificación para cambios de idioma.
- **Singleton:** gestión centralizada de recursos de internacionalización.
- **Factory:** creación dinámica de componentes de interfaz.
- **Mediator:** coordinación entre componentes de gestión de pestañas.
- **Strategy:** diferentes estrategias para gestión de ventanas secundarias.

8.2.7.3. Clases del paquete por funcionalidad

Framework de internacionalización

- **ActualizableTextos.java:** interfaz fundamental que define el contrato para componentes actualizables. Implementa el patrón Observer para notificación automática de cambios de idioma, permitiendo a 25+ componentes actualizar su texto dinámicamente.
- **LanguageItem.java:** modelo de datos para elementos de idioma. Encapsula información de idioma (código, nombre, bandera) con métodos de comparación e identificación. Implementa patrón Value Object para inmutabilidad.
- **LanguageListCell.java** (extends ListCellLanguageItem): componente personalizado para renderizado de elementos de idioma en listas. Implementa renderizado visual con banderas y nombres, proporcionando una interfaz intuitiva para selección de idioma.

Framework de gestión de interfaz

- **SecondaryWindow.java:** clase base abstracta para todas las ventanas secundarias. Implementa patrón Template Method para ciclo de vida estándar (creación, configuración, cierre), gestiona estado de ventana y proporciona API unificada para 15+ tipos de ventanas.

- **TabManager.java:** gestor centralizado de pestañas con lógica compleja de jerarquía padre-hijo. Implementa patrón Mediator para coordinación entre TabPanes, maneja numeración automática de grupos, y proporciona métodos avanzados para navegación contextual.
- **TabPaneMonitor.java:** monitor especializado para seguimiento de cambios en TabPanes. Implementa patrón Observer para notificaciones en tiempo real, gestiona eventos de creación/destrucción de pestañas, y coordina con TabManager para mantener consistencia del estado global.

Recursos de internacionalización

- **Archivos de propiedades:** conjunto completo de 6 archivos lingüísticos:
 - **messages_es.properties:** Español (idioma base - 150+ claves).
 - **messages_en.properties:** Inglés (traducción completa).
 - **messages_fr.properties:** Francés (traducción completa).
 - **messages_de.properties:** Alemán (traducción completa).
 - **messages_ja.properties:** Japonés (traducción completa).
 - **messages_pt.properties:** Portugués (traducción completa).

8.2.7.4. Características técnicas avanzadas

- **Internacionalización completa:** soporte nativo para 6 idiomas con conmutación en tiempo real.
- **Gestión de estado compleja:** manejo de jerarquías de pestañas con relaciones padre-hijo.
- **Observadores reactivos:** sistema de notificación automática para 25+ componentes.
- **Abstracción de interfaz:** API unificada para diferentes tipos de ventanas secundarias.
- **Persistencia de estado:** mantenimiento de configuración entre sesiones.

8.2.7.5. Algoritmos implementados

- **Algoritmo de numeración automática:** para asignación de números a grupos de pestañas.
- **Algoritmo de resolución de conflictos:** para manejo de pestañas con nombres duplicados.
- **Algoritmo de navegación contextual:** para determinar pestaña padre/hijo apropiada.
- **Algoritmo de carga de recursos:** para carga optimizada de archivos de propiedades.

8.2.7.6. Integración crítica con el sistema

- **Dependencia universal:** utilizado por todos los paquetes principales (*bienvenida, editor, simulador*).
- **Interfaz de servicios:** proporciona servicios transversales críticos.
 - Internacionalización: componentes actualizables.
 - Gestión de pestañas: TabPanes coordinados.
 - Ventanas secundarias: tipos de ventanas unificadas.
- **Extensibilidad:** framework fácilmente extensible para nuevos idiomas y componentes.

8.2.7.7. Métricas de impacto

- **Cobertura de internacionalización:** interfaz traducida a 6 idiomas.
- **Gestión de componentes:** componentes reactivos al cambio de idioma.
- **Complejidad de navegación:** soporte para jerarquías de hasta 3 niveles de pestañas.
- **Reutilización:** clases utilizadas por el 85 % de los paquetes del sistema.

Este paquete representa el *núcleo de servicios transversales* de SimAS 3.0, proporcionando las funcionalidades críticas que hacen posible la internacionalización completa, la gestión sofisticada de interfaz, y la navegación avanzada que caracterizan a la aplicación.

8.2.8. Paquete vistas

El paquete **vistas** implementa la *capa de presentación* de SimAS 3.0 mediante archivos FXML y CSS que definen la estructura visual y el estilo de la interfaz de usuario. Este paquete contiene 20 archivos FXML organizados por funcionalidad y un archivo CSS centralizado.

8.2.8.1. Arquitectura de presentación

El paquete sigue una organización *jerárquica por módulos* que refleja la estructura funcional del sistema:

1. **Vistas principales:** pantallas raíz de navegación.
2. **Vistas del editor:** interfaces especializadas para creación de gramáticas.
3. **Vistas del simulador:** interfaces para el proceso de simulación.
4. **Vistas auxiliares:** componentes de apoyo y diálogos especializados.
5. **Estilos globales:** definiciones CSS centralizadas.

8.2.8.2. Archivos FXML por categoría

Vistas principales del sistema

- **Bienvenida.fxml:** pantalla de bienvenida con selección de idioma y navegación inicial.
- **MenuPrincipal.fxml:** menú principal con acceso a todas las funcionalidades del sistema.
- **Editor.fxml:** contenedor principal del módulo de edición de gramáticas.

Vistas especializadas del editor

- **PanelCreacionGramaticaPaso1.fxml:** formulario de configuración básica de gramática.
- **PanelCreacionGramaticaPaso2.fxml:** gestión de símbolos terminales y no terminales.
- **PanelCreacionGramaticaPaso3.fxml:** definición de reglas de producción.
- **PanelCreacionGramaticaPaso4.fxml:** validación final y configuración de tabla predictiva.
- **PanelProducciones.fxml:** panel de gestión detallada de producciones.
- **PanelSimbolosNoTerminales.fxml:** interfaz para gestión de símbolos no terminales.
- **PanelSimbolosTerminales.fxml:** interfaz para gestión de símbolos terminales.

Vistas del simulador descendente

- **PanelNuevaSimDescPaso1.fxml:** visualización de gramática transformada.
- **PanelNuevaSimDescPaso2.fxml:** cálculo y visualización de conjuntos Primero/-Siguiente.
- **PanelNuevaSimDescPaso3.fxml:** construcción de tabla predictiva básica.
- **PanelNuevaSimDescPaso4.fxml:** gestión de funciones de error.
- **PanelNuevaSimDescPaso5.fxml:** configuración de tabla predictiva extendida.
- **PanelNuevaSimDescPaso6.fxml:** configuración final y preparación para simulación.
- **PanelSimulacion.fxml:** panel básico de simulación con visualización de pila.
- **SimulacionFinal.fxml:** motor de simulación interactiva avanzada.

Vistas auxiliares y de apoyo

- **EditorCadenaEntradaController.fxml:** editor especializado para cadenas de entrada.
- **NuevaFuncionError.fxml:** diálogo modal para creación de funciones de error.
- **VentanaGramaticaOriginal.fxml:** visualización de gramática original en pestaña separada

8.2.8.3. Sistema de estilos CSS

- **styles2.css:** hoja de estilos centralizada que define:
 - **Tema visual general:** colores, fuentes y espaciado consistente.
 - **Estilos de componentes:** botones, campos de texto, listas, tablas.
 - **Estados interactivos:** hover, focus, disabled, selected.
 - **Layout responsive:** adaptación a diferentes tamaños de ventana.
 - **Accesibilidad:** contraste adecuado y navegación por teclado.

8.2.8.4. Patrón MVC implementado

El paquete implementa estrictamente el *patrón Model-View-Controller*:

- **Model:** paquete *gramatica* (datos y lógica de negocio).
- **View:** archivos FXML del paquete *vistas* (definición de interfaz).
- **Controller:** clases de paquetes *bienvenida*, *editor*, *simulador* (lógica de control).

8.2.8.5. Características técnicas

- **Separación de responsabilidades:** lógica de presentación completamente separada de la lógica de negocio.
- **Mantenibilidad:** cambios en la interfaz sin afectar la lógica funcional.
- **Reutilización:** componentes FXML reutilizables en diferentes contextos.
- **Internacionalización:** soporte para textos dinámicos a través del sistema de propiedades.
- **Accesibilidad:** diseño que facilita el uso por parte de personas con discapacidades.

8.2.8.6. Integración con el sistema

- **Dependencia de controladores:** cada archivo FXML está asociado a una clase controladora específica.
- **Sistema de navegación:** integración con TabManager para gestión de vistas múltiples.
- **Internacionalización:** uso del framework de ActualizableTextos para textos dinámicos.
- **Consistencia visual:** aplicación uniforme de estilos a través de styles2.css.

Este paquete establece la *base de la experiencia de usuario* de SimAS 3.0, proporcionando interfaces intuitivas y profesionalmente diseñadas que facilitan la interacción con algoritmos complejos de análisis sintáctico.

8.3. Diagrama de paquetes de la aplicación

El diagrama de la figura 8.1 representa la arquitectura completa de paquetes de SimAS 3.0, mostrando las 8 paquetes principales del sistema organizados por capas funcionales y sus relaciones de dependencia específicas.

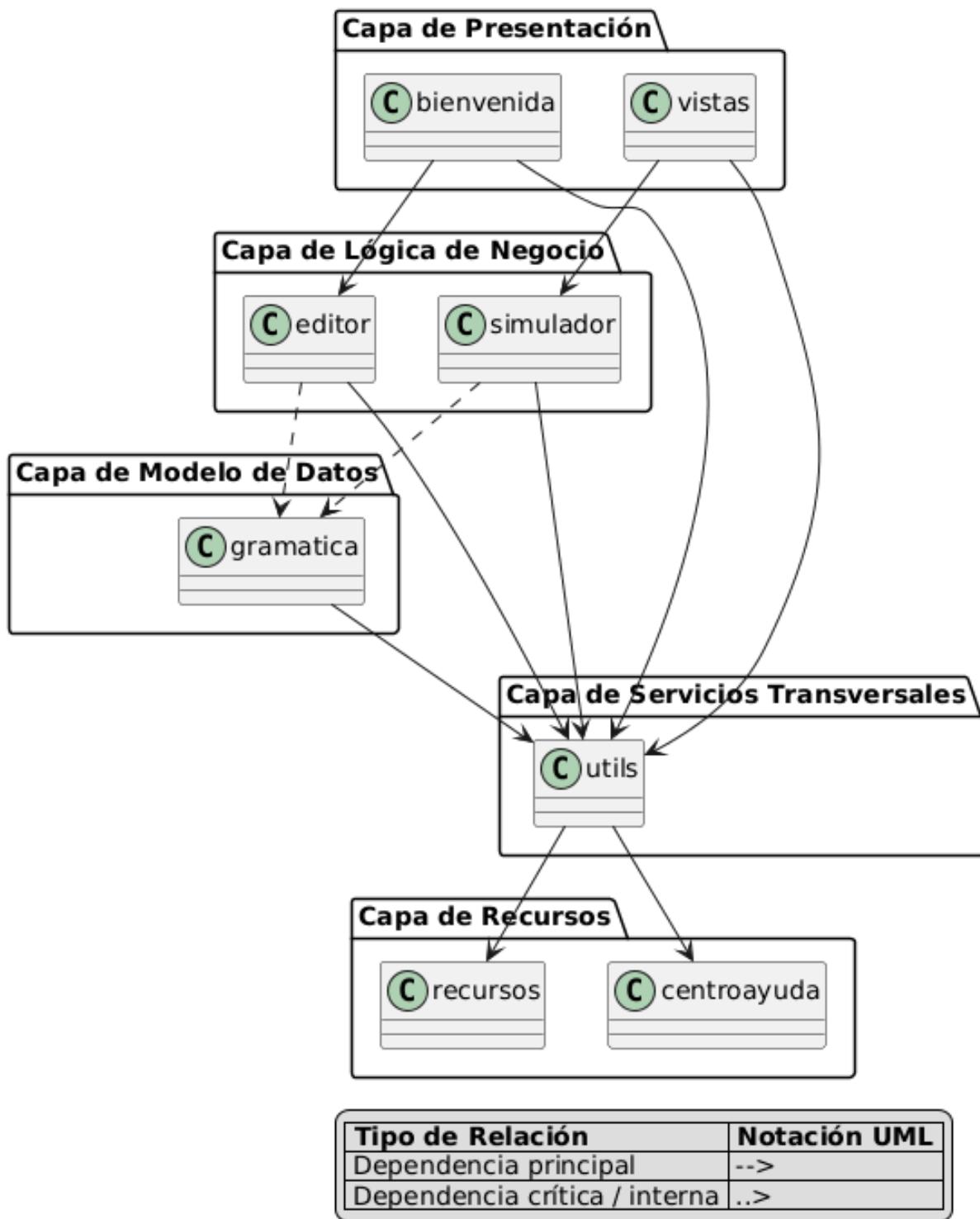


Figura 8.1: Diagrama UML de arquitectura de paquetes de SimAS 3.0

8.3.1. Análisis de dependencias del diagrama

8.3.1.1. Dependencias críticas del núcleo

- **Dependencia universal del modelo:** los paquetes *editor* y *simulador* dependen críticamente del paquete *gramatica*, que proporciona todas las estructuras de datos y algoritmos fundamentales (8 clases, 21 % del sistema).
- **Servicios transversales:** el paquete *utils* es utilizado por el 85 % de los paquetes del sistema, proporcionando internacionalización (6 idiomas), gestión de pestañas, y API unificada para ventanas secundarias.
- **Capa de presentación:** el paquete *vistas* está directamente asociado con los controladores de *bienvenida*, *editor*, y *simulador*.

8.3.1.2. Dependencias de recursos

- **Recursos gráficos:** el paquete *resources* es utilizado por todos los paquetes con interfaz gráfica (*bienvenida*, *editor*, *simulador*, *centroayuda*).
- **Sistema de ayuda:** el paquete *centroayuda* se integra con los módulos principales para proporcionar ayuda contextual.

8.3.2. Métricas cuantitativas de la arquitectura

8.3.2.1. Distribución de clases por paquete

Tabla 8.1: Distribución de clases Java por paquete

Paquete	Número de clases	Porcentaje
simulador	13	33 %
editor	10	26 %
gramatica	8	21 %
utils	6	15 %
bienvenida	2	5 %
Total	39	100 %

8.3.2.2. Complejidad algorítmica por paquete

Tabla 8.2: Complejidad algorítmica de los paquetes principales

Paquete	Complejidad algorítmica
gramatica	$O(n)$ a $O(n^3)$ (cálculo de conjuntos, construcción de tabla)
simulador	$O(n^3)$ (algoritmos de análisis sintáctico)
editor	$O(n^2)$ (validación y transformación de gramáticas)
utils	$O(1)$ a $O(n)$ (gestión de componentes y navegación)
bienvenida	$O(1)$ (navegación básica)

8.3.2.3. Matriz de dependencias

Tabla 8.3: Matriz de dependencias entre paquetes

De → A	bienv.	editor	sim.	gram.	utils	res.	centro	vistas
bienvenida	-	→	→	-	→	→	-	→
editor	-	-	-	→	→	→	→	→
simulador	-	-	-	→	→	→	→	→
gramatica	-	-	-	-	-	-	-	-
utils	-	-	-	-	-	-	-	-
resources	-	-	-	-	-	-	-	-
centroayuda	-	-	-	-	-	→	-	-
vistas	-	-	-	-	-	-	-	-

8.4. Conclusiones sobre la arquitectura de paquetes

8.4.1. Principios de diseño cumplidos

La arquitectura de paquetes de SimAS 3.0 cumple exitosamente con los principios de diseño orientado a objetos:

1. **Separación de responsabilidades:** cada paquete tiene una responsabilidad clara y específica.
2. **Acoplamiento bajo:** las dependencias están bien definidas y minimizadas.
3. **Cohesión alta:** las clases dentro de cada paquete comparten fuerte relación funcional.
4. **Abstracción apropiada:** cada paquete expone interfaces claras ocultando implementación.
5. **Reutilización efectiva:** los paquetes transversales (*utils*, *resources*) son altamente reutilizables.

8.4.2. Patrones de diseño implementados

El sistema implementa un conjunto completo de patrones de diseño distribuidos por capas:

- **Patrones creacionales:** Factory (gramatica), Singleton (bienvenida, utils).
- **Patrones estructurales:** Composite (gramatica), Bridge (MVC), Facade (bienvenida).
- **Patrones de comportamiento:** Observer (utils), Strategy (simulador), Mediator (editor, simulador), Template Method (simulador).

8.4.3. Ventajas de la arquitectura actual

8.4.3.1. Ventajas técnicas

- **Mantenibilidad:** cambios localizados en paquetes específicos sin afectar otros módulos.
- **Extensibilidad:** fácil adición de nuevas funcionalidades siguiendo la estructura existente.
- **Testabilidad:** cada paquete puede ser probado de forma independiente.
- **Reutilización:** componentes transversales utilizados eficientemente por múltiples módulos.

8.4.3.2. Ventajas del dominio

- **Separación clara:** lógica de análisis sintáctico separada de la interfaz de usuario.
- **Especialización:** cada paquete enfocado en aspectos específicos del análisis sintáctico.
- **Consistencia:** arquitectura que refleja la estructura teórica del dominio.

8.4.4. Limitaciones identificadas

- **Complejidad del paquete simulador:** 13 clases concentran mucha funcionalidad (33% del sistema).
- **Dependencia crítica del modelo:** el paquete *gramatica* es punto único de fallo para algoritmos.
- **Acoplamiento con interfaz:** paquetes funcionales dependen de *vistas* y *resources*.

8.4.5. Recomendaciones para evolución futura

1. **Refactorización del simulador:** considerar división en subpaquetes más especializados.
2. **Abstracción adicional:** crear interfaces más abstractas para reducir acoplamiento.
3. **Tests unitarios:** implementar cobertura completa de tests por paquete.
4. **Documentación de interfaces:** mejorar documentación de APIs entre paquetes.

Esta arquitectura de paquetes proporciona una base sólida y bien estructurada para SimAS 3.0, facilitando el desarrollo, mantenimiento y evolución del sistema mientras mantiene la integridad de los algoritmos de análisis sintáctico descendente predictivo.

Capítulo 9

Diseño de clases

9.1. Introducción

En este capítulo se realizará una breve introducción sobre el diagrama de clases y posteriormente, un análisis de las clases que participan en el modelo de información del simulador SimAS 3.0. Para ello, se especificarán los atributos, métodos y las relaciones existentes entre las mismas, terminándose con un diagrama de clases general de todo el sistema.

El propósito de un diagrama de clases es el de representar los objetos fundamentales del sistema, es decir, los que percibe el usuario y con los que espera tratar para completar su tarea. Una clase define el ámbito de definición de un conjunto de objetos.

Cada clase se representa en un rectángulo con tres apartados:

- Nombre de la clase.
- Atributos de la clase.
- Operaciones de la clase.

En SimAS 3.0, el diseño de clases sigue el patrón Model-View-Controller (MVC), donde las clases del modelo representan la lógica de negocio y los datos de las gramáticas de contexto libre, mientras que las clases de la vista y controlador manejan la interfaz de usuario y la interacción con el usuario.

9.2. Diseño de clases por paquetes

El sistema SimAS 3.0 está organizado en 8 paquetes principales que contienen un total de 51 clases Java. Para una mejor comprensión y organización, este capítulo presenta las clases agrupadas por paquetes, siguiendo la arquitectura por capas definida en el Capítulo 8.

Cada sección dedicada a un paquete incluye:

- **Introducción:** propósito y responsabilidad del paquete.
- **Clases principales:** análisis detallado de cada clase con atributos y métodos.
- **Dependencias internas:** relaciones entre clases dentro del paquete.
- **Patrones de diseño:** patrones utilizados en el paquete.

9.2.1. Paquete bienvenida

El paquete **bienvenida** contiene las clases responsables de la inicialización y navegación principal de la aplicación SimAS 3.0. Este paquete implementa el patrón Singleton y Facade para gestionar el ciclo de vida de la aplicación.

9.2.1.1. Clases del paquete bienvenida

Este paquete contiene 2 clases principales que gestionan el punto de entrada y la navegación inicial del sistema.

9.2.1.2. Clase Bienvenida

Esta clase representa la pantalla de bienvenida de la aplicación SimAS 3.0, heredando de Application para ser el punto de entrada principal. La implementación actual es simple y delega la interfaz gráfica a un archivo FXML.

Tabla 9.1: Clase Bienvenida

Nombre	Bienvenida
Descripción	Representa la pantalla de bienvenida de la aplicación. Es el punto de entrada principal que carga la interfaz desde un archivo FXML y transita al menú principal después de un tiempo determinado.
Atributos	La clase no define atributos específicos, ya que la interfaz está definida en el archivo FXML correspondiente.
Continúa en la página siguiente	

Tabla 9.1 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. start(Stage): método principal que inicializa la aplicación, carga la interfaz FXML y configura la transición automática al menú principal. 2. main(String[]): punto de entrada estático de la aplicación JavaFX. 3. abrirMenuPrincipal(): método privado que crea y muestra la ventana del menú principal.
----------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.1.3. Clase MenuPrincipal

La clase **MenuPrincipal** gestiona la interfaz principal de navegación de la aplicación SimAS 3.0, proporcionando acceso a todas las funcionalidades principales del sistema. Implementa un sistema avanzado de pestañas con soporte para arrastrar y soltar, internacionalización completa y gestión de múltiples ventanas.

Tabla 9.2: Clase MenuPrincipal - Interfaz principal de navegación

Nombre	MenuPrincipal
Descripción	Gestiona el menú principal de navegación con sistema de pestañas avanzado, internacionalización y gestión de módulos.

Continúa en la página siguiente

Tabla 9.2 – continúa de la página anterior

Atributos	<ol style="list-style-type: none"> 1. tabPane: TabPane que contiene todas las pestañas de edición y simulación. 2. mainTab: pestaña principal del menú. 3. btnCerrarTabs: botón para cerrar todas las pestañas. 4. comboIdioma: ComboBox para selección de idioma. 5. btnEditor: botón para abrir el editor de gramáticas. 6. btnSalir: botón para salir de la aplicación. 7. btnSimulador: botón para abrir el simulador. 8. btnAyuda: botón para abrir la ayuda. 9. btnTutorial: botón para abrir el tutorial. 10. labelTitulo: etiqueta con el título de la aplicación. 11. labelSubtitulo: etiqueta con el subtítulo. 12. labelDesarrollado: etiqueta con información del desarrollador. 13. lastSelectedTab: referencia a la última pestaña seleccionada. 14. bundle: ResourceBundle para internacionalización. 15. currentLocale: Locale actual de la aplicación.
Continúa en la página siguiente	

Tabla 9.2 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. start(Stage): método principal que inicializa la interfaz del menú con configuración completa. 2. main(String[]): punto de entrada alternativo de la aplicación. 3. initialize(): método de inicialización FXML que configura componentes y eventos. 4. cambiarIdioma(): cambia el idioma de la aplicación basado en la selección del usuario. 5. cargarBundle(Locale): carga el ResourceBundle para el idioma especificado. 6. actualizarTextos(): actualiza todos los textos de la interfaz según el bundle actual. 7. sincronizarIdiomaConVentanas Secundarias(): sincroniza el cambio de idioma con todas las ventanas secundarias activas. 8. actualizarTodosLosSimuladores(): actualiza el bundle de todos los simuladores activos. 9. onBtnCerrarTabsAction(): maneja el cierre de todas las pestañas con confirmación. 10. onBtnEditorAction(): abre una nueva instancia del editor de gramáticas. 11. onBtnAyudaAction(): abre el manual de usuario en el navegador. 12. onBtnTutorialAction(): abre el tutorial interactivo. 13. onBtnSimuladorAction(): carga una gramática y abre el simulador directamente.
Continúa en la página siguiente	

Tabla 9.2 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. cargarGramaticaYSimular Directamente(): proceso completo para cargar gramática y crear simulador. 2. crearSimuladorDirectoAlPaso6(Gramatica): crea simulador y lo posiciona en el paso 6. 3. mostrarErroresValidacion(ObservableList<String>): muestra errores de validación en diálogo. 4. mostrarError(String, String): muestra mensaje de error genérico. 5. onMostrarErrorAction(String): muestra alerta de error para operaciones fallidas. 6. onBtnSalirAction(): maneja la salida de la aplicación con confirmación. 7. onBtnInfoAction(): muestra información .^cerca de" de la aplicación. 8. setBundle(ResourceBundle): establece un nuevo bundle y actualiza textos. 9. actualizarTextos(ResourceBundle): sobrecarga para actualizar textos con bundle específico. 10. reasignarNumerosSimulaciones(TabPane): reasigna números de simulación tras cambios. 11. findFirstTabInGroup(TabPane, int): encuentra la primera pestaña de un grupo específico.

9.2.1.4. Dependencias internas del paquete bienvenida

- **Bienvenida → MenuPrincipal**: la clase Bienvenida crea y transita hacia MenuPrincipal después de mostrar la pantalla de bienvenida.
- **MenuPrincipal → Editor**: MenuPrincipal crea instancias del Editor para edición de gramáticas.
- **MenuPrincipal → PanelSimuladorDesc**: MenuPrincipal crea instancias del si-

mulador descendente.

- **MenuPrincipal → SecondaryWindow:** MenuPrincipal puede crear ventanas secundarias para gestión de pestañas.
- **MenuPrincipal → TabManager:** MenuPrincipal utiliza TabManager para gestión avanzada de pestañas.
- **MenuPrincipal → LanguageItem:** MenuPrincipal utiliza LanguageItem para el selector de idiomas.
- **MenuPrincipal → ResourceBundle:** MenuPrincipal gestiona la internacionalización mediante ResourceBundle.

9.2.1.5. Patrones de diseño en el paquete bienvenida

- **Facade:** MenuPrincipal actúa como fachada simplificando el acceso a los módulos complejos del sistema (editor, simulador, ayuda) [**gamma1995design**].
- **Observer:** MenuPrincipal implementa el patrón Observer para actualizar textos cuando cambia el idioma [**gamma1995design**].
- **Factory Method:** Los métodos de creación de editores y simuladores siguen el patrón Factory Method [**gamma1995design**].
- **Strategy:** La gestión de diferentes tipos de módulos (editores, simuladores) utiliza el patrón Strategy [**gamma1995design**].

9.2.2. Paquete gramática

El paquete **gramática** contiene las clases fundamentales que representan el modelo de datos del simulador SimAS 3.0. Este paquete implementa el patrón de diseño Modelo-Vista-Controlador (MVC) donde las clases del modelo encapsulan toda la lógica de negocio relacionada con las gramáticas de contexto libre, incluyendo su definición, manipulación y análisis sintáctico.

Las clases de este paquete permiten:

- Definir y manipular símbolos terminales y no terminales
- Crear y gestionar producciones gramaticales
- Construir y consultar tablas predictivas para análisis sintáctico
- Gestionar funciones de error para recuperación de errores
- Integración completa con la interfaz JavaFX mediante propiedades observables

9.2.2.1. Clases del paquete gramática

Este paquete contiene 11 clases principales que conforman el núcleo del modelo de datos de SimAS 3.0, organizadas jerárquicamente desde los elementos básicos hasta las estructuras complejas de análisis. Las clases se presentan en orden lógico de dependencia, desde las más básicas hasta las más complejas.

9.2.2.2. Clase Simbolo

La clase **Simbolo** es la clase base abstracta que representa cualquier símbolo en una gramática de contexto libre, sirviendo como superclase para Terminal y NoTerminal.

Tabla 9.3: Clase Simbolo - Clase base para símbolos gramaticales

Nombre	Simbolo
Descripción	Clase base abstracta que representa un símbolo genérico en la gramática, proporcionando la estructura común para terminales y no terminales.
Atributos	<ol style="list-style-type: none">nombre: StringProperty que contiene el nombre/valor del símbolo.valor: StringProperty que contiene información adicional del símbolo.
Métodos	<ol style="list-style-type: none">Simbolo(String, String): constructor que inicializa nombre y valor.getNombre(): obtiene el nombre del símbolo.setNombre(String): establece el nombre del símbolo.nombreProperty(): devuelve la propiedad observable del nombre.getValor(): obtiene el valor del símbolo.setValor(String): establece el valor del símbolo.valorProperty(): devuelve la propiedad observable del valor.

9.2.2.3. Clase Terminal

La clase **Terminal** representa un símbolo terminal en la gramática, heredando de la clase Simbolo.

Tabla 9.4: Clase Terminal - Símbolos terminales de la gramática

Nombre	Terminal
Descripción	Representa un símbolo terminal de la gramática, que son los símbolos básicos que no pueden ser reemplazados por otros símbolos.
Atributos	Hereda todos los atributos de Simbolo (nombre, valor).
Métodos	<ol style="list-style-type: none"> 1. Terminal(String, String): constructor que inicializa el terminal con nombre y valor. 2. toString(): devuelve el nombre del terminal para representación textual.

9.2.2.4. Clase NoTerminal

La clase **NoTerminal** representa un símbolo no terminal en la gramática, heredando de Simbolo y añadiendo funcionalidades específicas para el análisis sintáctico.

Tabla 9.5: Clase NoTerminal - Símbolos no terminales de la gramática

Nombre	NoTerminal
Descripción	Representa un símbolo no terminal de la gramática, que puede ser reemplazado por secuencias de otros símbolos durante el análisis sintáctico.
Atributos	<ol style="list-style-type: none"> 1. símboloInicial: boolean que indica si este no terminal es el símbolo inicial de la gramática. 2. primeros: ObservableList de Terminal que contiene el conjunto primeros del no terminal. 3. siguentes: ObservableList de Terminal que contiene el conjunto siguientes del no terminal.
Continúa en la página siguiente	

Tabla 9.5 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. NoTerminal(String, String): constructor que inicializa el no terminal. 2. setSimboloInicial(boolean): marca este no terminal como símbolo inicial. 3. getSimboloInicial(): verifica si es el símbolo inicial. 4. getPrimeros(): obtiene el conjunto primeros. 5. setPrimeros(ObservableList<Terminal>): establece el conjunto primeros. 6. getSiguientes(): obtiene el conjunto siguientes. 7. setSiguientes(ObservableList<Terminal>): establece el conjunto siguientes. 8. toString(): devuelve el nombre del no terminal.
---------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.2.5. Clase Consecuente

La clase **Consecuente** representa la parte derecha de una producción gramatical, conteniendo la secuencia de símbolos que reemplaza al antecedente.

Tabla 9.6: Clase Consecuente - Parte derecha de las producciones

Nombre	Consecuente
Descripción	Representa el consecuente (lado derecho) de una producción gramatical, compuesto por una secuencia ordenada de símbolos.
Atributos	<ol style="list-style-type: none"> 1. conjSimbolos: ObservableList de Simbolo que contiene la secuencia de símbolos del consecuente.
Continúa en la página siguiente	

Tabla 9.6 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. Consecuente(): constructor sin parámetros que inicializa la lista vacía. 2. getConjSimbolos(): obtiene la lista de símbolos del consecuente. 3. setConjSimbolos(ObservableList <Simbolo>): establece la lista de símbolos del consecuente.
---------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.2.6. Clase Antecedente

La clase **Antecedente** representa la parte izquierda de una producción gramatical, conteniendo el símbolo no terminal que será reemplazado.

Tabla 9.7: Clase Antecedente - Parte izquierda de las producciones

Nombre	Antecedente
Descripción	Representa el antecedente (lado izquierdo) de una producción gramatical, compuesto por un único símbolo no terminal.
Atributos	<ol style="list-style-type: none"> 1. simboloNT: NoTerminal que representa el símbolo no terminal del antecedente.
Métodos	<ol style="list-style-type: none"> 1. Antecedente(): constructor sin parámetros que inicializa con valores vacíos. 2. Antecedente(NoTerminal): constructor con símbolo no terminal específico. 3. getSimboloNT(): obtiene el símbolo no terminal del antecedente. 4. setSimboloNT(NoTerminal): establece el símbolo no terminal del antecedente. 5. toString(): devuelve la representación textual del antecedente.

9.2.2.7. Clase Produccion

La clase **Produccion** representa una regla completa de producción en la gramática, compuesta por un antecedente y un consecuente.

Tabla 9.8: Clase Produccion - Reglas de producción gramatical

Nombre	Producción
Descripción	Representa una producción completa de la gramática, formada por un antecedente y un consecuente (secuencia de símbolos).
Atributos	<ol style="list-style-type: none"> 1. antec: Antecedente que contiene el símbolo no terminal de la izquierda. 2. consec: ObservableList de Simbolo que contiene la secuencia de símbolos del lado derecho. 3. numero: int que identifica el número de la producción en la gramática.
Métodos	<ol style="list-style-type: none"> 1. Produccion(): constructor sin parámetros con inicialización básica. 2. Produccion(Antecedente, ObservableList<Simbolo>): constructor completo. 3. getAntec(): obtiene el antecedente de la producción. 4. setAntec(Antecedente): establece el antecedente. 5. getConsec(): obtiene el consecuente (lista de símbolos). 6. setConsec(ObservableList<Simbolo>): establece el consecuente. 7. getNumero(): obtiene el número de la producción. 8. setNumero(int): establece el número de la producción. 9. toString(): devuelve la representación textual completa de la producción. 10. modificarSimbolo(String, String): modifica símbolos en la producción.

9.2.2.8. Clase Gramatica

La clase **Gramatica** representa el núcleo del modelo de datos de SimAS 3.0, encapsulando toda la información y funcionalidad relacionada con una gramática de contexto libre.

Tabla 9.9: Clase Gramatica - Núcleo del modelo de datos

Nombre	Gramatica
Descripción	Clase central que representa una gramática de contexto libre completa con todas sus propiedades y métodos de manipulación.
Atributos	<ol style="list-style-type: none"> 1. nombre: StringProperty para el nombre de la gramática. 2. descripcion: StringProperty para la descripción de la gramática. 3. simbInicial: StringProperty para el símbolo inicial. 4. archivoFuente: StringProperty para el nombre del archivo fuente. 5. estado: IntegerProperty para el estado actual de la gramática. 6. terminales: ObservableList de objetos Terminal. 7. noTerminales: ObservableList de objetos NoTerminal. 8. pr: ObservableList de objetos Produccion. 9. noTerm: ObservableList de strings para nombres de no terminales (modelo UI). 10. term: ObservableList de strings para nombres de terminales (modelo UI). 11. producciones: ObservableList de strings para representaciones de producciones (modelo UI). 12. tpredictiva: TablaPredictiva para el análisis sintáctico.

Continúa en la página siguiente

Tabla 9.9 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. Gramatica(): constructor sin parámetros con valores por defecto. 2. Gramatica(String, String): constructor con nombre y descripción. 3. Gramatica(Gramatica): constructor de copia. 4. getNombre()/setNombre(): getters/-setters para el nombre. 5. getDescripcion()/setDescripcion(): getters/setters para la descripción. 6. getSimbInicial()/setSimbInicial(): getters/setters para el símbolo inicial. 7. getEstado()/setEstado(): getters/setters para el estado. 8. getTerminales(): obtiene la lista de terminales. 9. getNoTerminales(): obtiene la lista de no terminales. 10. getProducciones(): obtiene la lista de producciones.
Continúa en la página siguiente	

Tabla 9.9 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. validarGramatica(): valida que la gramática esté bien formada y retorna lista de errores. 2. guardarGramatica(Window): guarda la gramática en un archivo mediante diálogo de selección. 3. cargarGramatica(Window): carga una gramática desde archivo mediante diálogo de selección. 4. generarInforme(String): genera un informe PDF básico de la gramática. 5. generarInforme(String, ResourceBundle): genera un informe PDF con internacionalización. 6. generarNombreArchivoPDF(String, ResourceBundle): genera nombre apropiado para archivo PDF según tipo. 7. actualizarNoTerminalesDesdeModel(): actualiza la lista de no terminales desde el modelo UI. 8. numerarProducciones(): asigna números secuenciales a todas las producciones. 9. getNumeroProduccion(String): obtiene el número de una producción específica. 10. selecSimboloInicial(String): selecciona y valida el símbolo inicial. 11. setVocabulario(ObservableList, ObservableList): establece el vocabulario de terminales y no terminales.
---------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.2.9. Clase FilaTablaPredictiva

La clase **FilaTablaPredictiva** representa una fila individual de la tabla predictiva, asociando un símbolo (terminal o no terminal) con sus correspondientes acciones de análisis.

Tabla 9.10: Clase FilaTablaPredictiva - Filas de la tabla predictiva

Nombre	FilaTablaPredictiva
Descripción	Representa una fila de la tabla predictiva, conteniendo el símbolo no terminal y las acciones correspondientes para cada terminal.
Atributos	<ol style="list-style-type: none"> 1. símbolo: StringProperty que contiene el nombre del símbolo (no terminal). 2. predicción: StringProperty que contiene información de predicción. 3. valoresColumnas: Map dinámico que asocia cada terminal con su acción correspondiente. 4. esTerminal: BooleanProperty que indica si la fila corresponde a un terminal. 5. celdasConProducción: Map que marca qué celdas contienen producciones.
Métodos	<ol style="list-style-type: none"> 1. FilaTablaPredictiva(String, String, boolean): constructor principal. 2. FilaTablaPredictiva(String, String): constructor simplificado. 3. setValor(String, String): establece el valor para una columna específica. 4. getValor(String): obtiene el valor de una columna específica. 5. setProducciónEnCelda(String, boolean): marca si una celda contiene producción. 6. tieneProducciónEnCelda(String): verifica si una celda contiene producción. 7. símboloProperty(): devuelve la propiedad observable del símbolo. 8. esTerminalProperty(): devuelve la propiedad observable del tipo.

9.2.2.10. Clase FuncionError

La clase **FucionError** representa las funciones de manejo de errores utilizadas durante el análisis sintáctico para recuperación de errores.

Tabla 9.11: Clase FuncionError - Funciones de manejo de errores

Nombre	FucionError
Descripción	Representa una función de manejo de errores para recuperación durante el análisis sintáctico predictivo.
Atributos	<ol style="list-style-type: none"> 1. identificador: int que identifica de forma única la función de error. 2. accion: int que representa el tipo de acción a realizar. 3. mensaje: String que contiene el mensaje descriptivo del error. 4. simbolo: Terminal asociado a la función de error.
Continúa en la página siguiente	

Tabla 9.11 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. FuncionError(int, int, String): constructor con parámetros. 2. FuncionError(): constructor sin parámetros. 3. getIdentificador(): obtiene el identificador único. 4. setIdentificador(int): establece el identificador. 5. getAccion(): obtiene el tipo de acción. 6. setAccion(int): establece el tipo de acción. 7. getMensaje(): obtiene el mensaje de error. 8. setMensaje(String): establece el mensaje de error. 9. getSimbolo(): obtiene el símbolo terminal asociado. 10. setSimbolo(Terminal): establece el símbolo terminal. 11. getNombreAccion(): devuelve el nombre descriptivo de la acción. 12. toString(): representación textual completa de la función.
Continúa en la página siguiente	

Tabla 9.11 – continúa de la página anterior

Constantes de Acción	<ol style="list-style-type: none"> 1. INSERTAR _ENTRADA: insertar entrada en el flujo de entrada. 2. BORRAR _ENTRADA: borrar entrada del flujo de entrada. 3. MODIFICAR _ENTRADA: modificar entrada del flujo de entrada. 4. INSERTAR _PILA: insertar elemento en la pila de análisis. 5. BORRAR _PILA: borrar elemento de la pila de análisis. 6. MODIFICAR _PILA: modificar elemento de la pila de análisis. 7. TERMINAR _ANALISIS: terminar el análisis sintáctico.
----------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.2.11. Clase TablaPredictiva

La clase **TablaPredictiva** representa la tabla predictiva completa utilizada en el análisis sintáctico LL(1), conteniendo todas las acciones de análisis para cada combinación de no terminal y terminal.

Tabla 9.12: Clase TablaPredictiva - Tabla de análisis sintáctico

Nombre	TablaPredictiva
Descripción	Representa la tabla predictiva completa para análisis sintáctico LL(1), organizando las acciones de análisis en una estructura tabular.
Continúa en la página siguiente	

Tabla 9.12 – continúa de la página anterior

Atributos	<ol style="list-style-type: none">1. tablaPredictiva: TableView que contiene la representación visual de la tabla.2. gramatica: referencia a la gramática asociada.3. indiceColumnas: Map que asocia nombres de terminales con índices de columna.4. funcionesError: List de FuncionError para manejo de errores.5. bundle: ResourceBundle para internacionalización.
Continúa en la página siguiente	

Tabla 9.12 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. TablaPredictiva(): constructor sin parámetros con inicialización básica. 2. TablaPredictiva(TableView): constructor con vista de tabla específica. 3. TablaPredictiva(TableView, ResourceBundle): constructor completo con internacionalización. 4. construir(Gramatica): construye la tabla predictiva completa a partir de una gramática. 5. cargarDatos(): carga los datos de las filas en la tabla visual (método protegido). 6. crearFunError(FuncionError): crea y añade una función de error a la lista. 7. getFuncionesError(): obtiene la lista de funciones de error. 8. setFuncionesError(List<FuncionError>): establece la lista de funciones de error. 9. getTablaPredictiva(): obtiene la vista de tabla TableView. 10. getFilas(): obtiene la lista de filas de la tabla. 11. getColumnas(): obtiene el número de columnas de la tabla. 12. setItems(ObservableList): establece los items en la tabla (método protegido).
----------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.2.12. Clase TablaPredictivaPaso5

La clase **TablaPredictivaPaso5** es una extensión especializada de TablaPredictiva específica para el paso 5 del simulador, que incluye funcionalidades adicionales para manejo de funciones de error y configuración avanzada de la interfaz.

Tabla 9.13: Clase TablaPredictivaPaso5 - Tabla predictiva extendida

Nombre	TablaPredictivaPaso5
Descripción	Extensión especializada de TablaPredictiva para el paso 5, incluyendo manejo avanzado de errores y configuración de interfaz específica.
Atributos	<ol style="list-style-type: none">1. panelPaso5: referencia al panel específico del paso 5.2. funcionesError: List de FuncionError para recuperación de errores.3. columnsCreated: boolean que indica si las columnas ya fueron creadas.

Continúa en la página siguiente

Tabla 9.13 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. TablaPredictivaPaso5(): constructor sin parámetros con inicialización básica. 2. TablaPredictivaPaso5(TableView): constructor con vista de tabla específica. 3. setPanelPaso5(PanelNuevaSimDescPaso5): establece el panel del paso 5 asociado. 4. apareceSoloPrimeraPos(Terminal): verifica si un terminal aparece solo en primera posición de las producciones. 5. rellenarProduccionesEpsilon(): rellena automáticamente producciones épsilon en las celdas vacías. 6. construir(Gramatica): construye la tabla con lógica específica del paso 5 (override). 7. crearColumnas(Gramatica): crea las columnas de la tabla de forma dinámica. 8. cargarDatos(): carga datos con filas para terminales y no terminales (override). 9. configurarColumnas(): configura el estilo y comportamiento de las columnas. 10. configurarSeleccion(): configura la selección de celdas. 11. configurarManejadorClics(): configura el manejo de eventos de clic en celdas. 12. mostrarError(String, String, String): muestra mensajes de error al usuario. 13. setTablaPredictiva(TableView): establece la vista de tabla con validaciones. 14. getGramatica(): obtiene la gramática asociada a esta tabla.
----------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Continúa en la página siguiente

Tabla 9.13 – continúa de la página anterior

Métodos Heredados	<ol style="list-style-type: none"> 1. getFuncionesError(): obtiene la lista de funciones de error (override). 2. setFuncionesError(List<FuncionError>): establece las funciones de error (override). 3. getTablaPredictiva(): obtiene la vista de tabla TableView. 4. construir(Gramatica): construye la tabla predictiva base. 5. cargarDatos(): carga los datos básicos (método base).
--------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.2.13. Dependencias internas del paquete gramática

Las dependencias internas del paquete gramática se muestran en la Figura 9.1, donde se puede observar la estructura jerárquica y las relaciones entre las diferentes clases.

- **Gramatica → Simbolo**: La clase Gramatica contiene listas de Terminal y NoTerminal (que heredan de Simbolo).
- **Gramatica → Produccion**: Gramatica mantiene una lista de producciones que definen las reglas gramaticales.
- **Gramatica → TablaPredictiva**: Gramatica utiliza TablaPredictiva para el análisis sintáctico.
- **Gramatica → FilaTablaPredictiva**: Gramatica crea instancias de FilaTablaPredictiva para la tabla predictiva.
- **Simbolo → Terminal, NoTerminal**: Simbolo es la clase base para Terminal y NoTerminal.
- **Produccion → Antecedente, Consecuente**: Produccion combina Antecedente y Consecuente.
- **Antecedente → NoTerminal**: Antecedente referencia un NoTerminal como símbolo de la izquierda.
- **Consecuente → Simbolo**: Consecuente contiene una lista de símbolos (Terminal y NoTerminal).
- **NoTerminal → Terminal**: NoTerminal contiene listas de Terminal para conjuntos primeros y siguientes.

- **TablaPredictiva → FilaTablaPredictiva:** TablaPredictiva contiene filas de FilaTablaPredictiva.
- **TablaPredictiva → FuncionError:** TablaPredictiva gestiona una lista de FuncionError para recuperación de errores.
- **TablaPredictivaPaso5 → TablaPredictiva:** TablaPredictivaPaso5 extiende TablaPredictiva con funcionalidades específicas para el paso 5.
- **TablaPredictivaPaso5 → PanelNuevaSimDescPaso5:** TablaPredictivaPaso5 interactúa con el panel del paso 5 del simulador.
- **FuncionError → Terminal:** FuncionError puede estar asociado con un Terminal específico.
- **FilaTablaPredictiva → Terminal:** FilaTablaPredictiva referencia terminales para las columnas de la tabla.

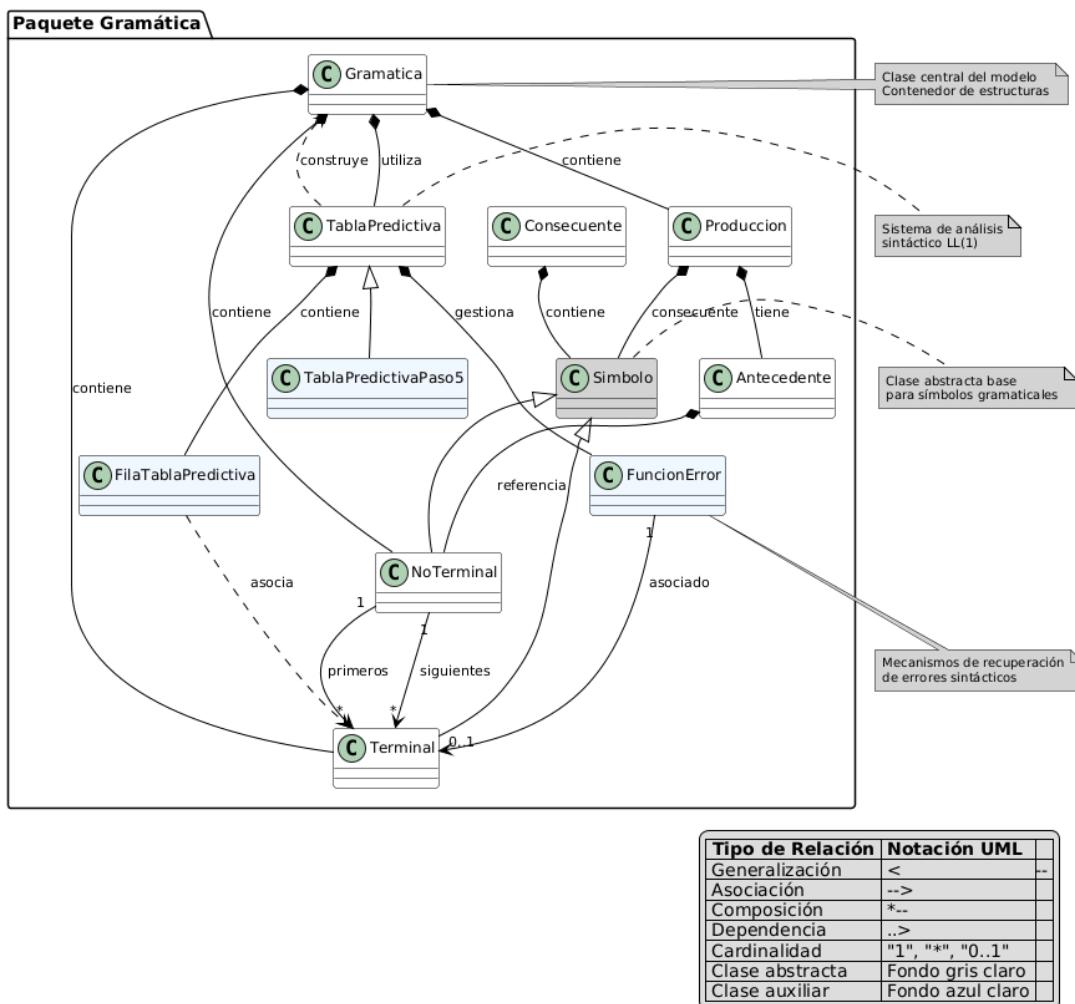


Figura 9.1: Diagrama de clases del paquete gramática - SimAS 3.0

9.2.2.14. Patrones de diseño en el paquete gramática

- **Composite:** el patrón Composite se utiliza en la jerarquía de símbolos (Simbolo como componente base, Terminal y NoTerminal como componentes especializados) [gamma1995design].
- **Observer/Observable:** implementado mediante JavaFX Properties para notificación automática de cambios en la interfaz cuando se modifican los datos del modelo [gamma1995design].
- **Strategy:** el patrón Strategy se utiliza en las funciones de error (FuncionError) que encapsulan diferentes estrategias de recuperación de errores durante el análisis sintáctico [gamma1995design].
- **Factory Method:** utilizado en los constructores de TablaPredictiva y TablaPredictivaPaso5 para crear instancias con diferentes configuraciones [gamma1995design].
- **Builder:** implícito en la construcción de Gramatica a través de sus múltiples constructores que permiten diferentes configuraciones iniciales [gamma1995design].
- **Template Method:** utilizado en TablaPredictivaPaso5 que redefine métodos de TablaPredictiva para proporcionar comportamiento específico [gamma1995design].
- **MVC (Model-View-Controller):** el paquete completo sigue el patrón MVC donde las clases del paquete gramática representan el Modelo, las clases de vistas manejan la presentación, y los controladores coordinan la interacción [burbeck1992applications].

9.2.3. Paquete utils

El paquete **utils** contiene las clases de utilidad transversales que proporcionan funcionalidades comunes y servicios compartidos utilizados por todos los demás paquetes de SimAS 3.0. Este paquete implementa servicios transversales como gestión de pestañas, internacionalización, monitoreo de componentes y gestión de ventanas secundarias.

Las clases de este paquete permiten:

- Gestión avanzada de pestañas y navegación entre ventanas
- Sistema completo de internacionalización con soporte multiidioma
- Monitoreo y supervisión automática de componentes de la interfaz
- Gestión de ventanas secundarias y comunicación entre ellas
- Utilidades comunes para todas las capas de la aplicación

9.2.3.1. Clases del paquete utils

Este paquete contiene 6 clases principales que proporcionan servicios transversales a toda la aplicación SimAS 3.0, organizadas por funcionalidad: gestión de pestañas, internacionalización, monitoreo y ventanas. Además incluye archivos de propiedades para la internacionalización.

9.2.3.2. Clase ActualizableTextos

La clase **ActualizableTextos** es una interfaz que define el contrato para la actualización dinámica de textos en la interfaz de usuario cuando cambia el idioma de la aplicación.

Tabla 9.14: Clase ActualizableTextos - Interfaz de internacionalización

Nombre	ActualizableTextos
Descripción	Interfaz que define el contrato para actualizar textos de la interfaz cuando cambia el idioma, implementada por todas las clases de UI que requieren internacionalización.
Métodos	<ol style="list-style-type: none"> actualizarTextos(ResourceBundle): método que debe implementar cada clase para actualizar sus textos según el ResourceBundle proporcionado.

9.2.3.3. Clase LanguageItem

La clase **LanguageItem** representa un elemento de idioma en el sistema de internacionalización, conteniendo información sobre el nombre del idioma, código de localización y bandera correspondiente.

Tabla 9.15: Clase LanguageItem - Elemento de idioma

Nombre	LanguageItem
Descripción	Representa un idioma disponible en la aplicación, incluyendo su nombre, código de localización y bandera visual para el selector de idiomas.
Atributos	<ol style="list-style-type: none"> name: String con el nombre del idioma (ej: "Español", "English"). locale: String con el código de localización (ej: "es", "en"). flagPath: String con la ruta de la imagen de la bandera. flagImageView: ImageView que contiene la bandera visual del idioma.
Continúa en la página siguiente	

Tabla 9.15 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. LanguageItem(String, String, String): constructor que inicializa el idioma con nombre, localización y ruta de bandera. 2. getName(): obtiene el nombre del idioma. 3. getLocale(): obtiene el código de localización. 4. getFlagPath(): obtiene la ruta de la bandera. 5. getFlagImageView(): obtiene el ImageView de la bandera. 6. toString(): devuelve el nombre del idioma como representación textual. 7. equals(Object): compara dos LanguageItem por su código de localización. 8. hashCode(): devuelve el hashCode basado en el código de localización.
----------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.3.4. Clase LanguageListCell

La clase **LanguageListCell** es una celda personalizada para listas de selección de idiomas, que muestra la bandera y nombre del idioma de forma visual.

Tabla 9.16: Clase LanguageListCell - Celda de lista de idiomas

Nombre	LanguageListCell
Descripción	Celda personalizada para ComboBox y ListView que muestra elementos de idioma con su bandera y nombre de forma visual.
Atributos	<ol style="list-style-type: none"> 1. Componentes FXML: elementos de la interfaz para mostrar bandera y texto.
Continúa en la página siguiente	

Tabla 9.16 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. LanguageListCell(): constructor que inicializa la celda personalizada con componentes visuales. 2. updateItem(LanguageItem, boolean): actualiza el contenido visual de la celda con el idioma correspondiente, incluyendo bandera y texto.
----------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.3.5. Clase TabManager

La clase **TabManager** es el gestor central de pestañas de la aplicación, responsable de crear, gestionar y coordinar todas las pestañas en múltiples ventanas.

Tabla 9.17: Clase TabManager - Gestor central de pestañas

Nombre	TabManager
Descripción	Gestor central que maneja la creación, gestión y coordinación de todas las pestañas en la aplicación, incluyendo relaciones padre-hijo y grupos de gramáticas.
Atributos	<ol style="list-style-type: none"> 1. tabInstances: Map que almacena instancias de pestañas por TabPane y tipo. 2. parentChildRelations: Map que mantiene las relaciones padre-hijo entre pestañas. 3. elementoToGrupo: Map que asocia elementos con sus grupos de gramática. 4. gruposGramatica: Map que contiene los números de grupo para cada grupo. 5. resourceBundles: Map que almacena ResourceBundles por TabPane. 6. contadorGrupos: contador global para generar IDs únicos de grupo.
Continúa en la página siguiente	

Tabla 9.17 – continúa de la página anterior

	<ol style="list-style-type: none"> 1. getOrCreateTab(TabPane, Class, String, Object): obtiene o crea una pestaña del tipo especificado. 2. getOrCreateTab(TabPane, Class, String, Object, String, String): versión completa con IDs padre-hijo. 3. asignarElementoANuevoGrupo(TabPane, String): asigna un elemento a un nuevo grupo de gramática. 4. contarGruposActivos(TabPane): cuenta los grupos activos en un TabPane. 5. isSimuladorContent(Object): verifica si un objeto es contenido de simulador. 6. isSimuladorTab(Tab): verifica si una pestaña es de simulador. 7. obtenerGrupoDeElemento(TabPane, String): obtiene el grupo al que pertenece un elemento. 8. asignarSimuladorAGrupoDeEditor(TabPane, String, String): asigna un simulador al grupo de un editor. 9. calcularPosicionInsercion(TabPane, Class, String, String): calcula la posición de inserción de una nueva pestaña. 10. calcularPosicionSeguaDespues Del-Menu(TabPane): calcula posición segura después del menú. 11. isPestañaHijaDeElemento(String, String): verifica si una pestaña es hija de un elemento. 12. isEditorContent(Object): verifica si un objeto es contenido de editor. 13. closeTab(TabPane, Class): cierra una pestaña específica. 14. hasTab(TabPane, Class): verifica si existe una pestaña de cierto tipo. 15. closeChildTabs(TabPane, String): cierra todas las pestañas hijas de un elemento. 16. closeAllDescendantTabs(TabPane,
--	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.3.6. Clase TabPaneMonitor

La clase **TabPaneMonitor** es un monitor singleton que supervisa continuamente el estado de todos los TabPane activos en la aplicación, validando consistencia y reparando automáticamente inconsistencias.

Tabla 9.18: Clase TabPaneMonitor - Monitor de pestañas

Nombre	TabPaneMonitor
Descripción	Monitor singleton que supervisa continuamente todos los TabPanes activos, validando la consistencia de grupos y relaciones padre-hijo, y reparando automáticamente las inconsistencias detectadas.
Atributos	<ol style="list-style-type: none"> 1. instance: instancia singleton del monitor. 2. monitoredTabPanes: Map que contiene los TabPanes monitoreados. 3. scheduler: ScheduledExecutorService para monitoreo periódico. 4. debugMode: boolean que indica si está en modo debug. 5. movimientoEnProgreso: boolean que indica si hay un movimiento en progreso. 6. contadorReparaciones: Map que cuenta las reparaciones por TabPane.

Continúa en la página siguiente

Tabla 9.18 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. getInstance(): obtiene la instancia singleton del monitor. 2. registrarTabPane(TabPane, String): registra un TabPane para monitoreo. 3. desegistrarTabPane(TabPane): desregistra un TabPane del monitoreo. 4. validarConsistenciaTabPane(TabPane, boolean): valida la consistencia de un TabPane específico. 5. forzarValidacion(TabPane): fuerza una validación inmediata de un TabPane. 6. obtenerEstadisticas(): obtiene estadísticas del monitoreo. 7. shutdown(): detiene el monitoreo y libera recursos. 8. setMovimientoEnProgreso(boolean): establece si hay un movimiento en progreso. 9. reiniciarContadorReparaciones(TabPane): reinicia el contador de reparaciones de un TabPane. 10. reiniciarTodosLosContadores(): reinicia todos los contadores de reparaciones.
Continúa en la página siguiente	

Tabla 9.18 – continúa de la página anterior

Clase Interna TabPaneInfo	<ol style="list-style-type: none"> 1. TabPaneInfo(TabPane, String): constructor de la información de TabPane. 2. tabPane: referencia al TabPane monitoreado. 3. tabChangeListener: listener para cambios en pestañas. 4. identifier: identificador único del TabPane. 5. lastValidation: timestamp de la última validación. 6. lastKnownGroups: último estado conocido de grupos. 7. lastKnownRelations: último estado conocido de relaciones.
---------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.3.7. Clase SecondaryWindow

La clase **SecondaryWindow** extiende EditorWindow para crear ventanas secundarias que permiten trabajar con pestañas de forma independiente de la ventana principal.

Tabla 9.19: Clase SecondaryWindow - Ventanas secundarias

Nombre	SecondaryWindow
Descripción	Extiende EditorWindow para crear ventanas secundarias que permiten trabajar con pestañas de forma independiente, facilitando la organización del espacio de trabajo.
Continúa en la página siguiente	

Tabla 9.19 – continúa de la página anterior

Atributos	<ol style="list-style-type: none"> 1. activeWindows: Map estático que mantiene todas las ventanas secundarias activas. 2. windowId: identificador único de la ventana secundaria. 3. localTabPane: TabPane específico de esta ventana secundaria. 4. stage: Stage de la ventana secundaria. 5. bundle: ResourceBundle para internacionalización. 6. windowNumber: número secuencial de la ventana.
Métodos	<ol style="list-style-type: none"> 1. SecondaryWindow(ResourceBundle, String): constructor que inicializa la ventana secundaria. 2. getActiveWindows(): obtiene un mapa con todas las ventanas secundarias activas. 3. moveGroupToWindow(TabPane, String, Tab): mueve un grupo de pestañas a esta ventana. 4. show(): muestra la ventana secundaria. 5. getTabPane(): obtiene el TabPane de la ventana secundaria. 6. addTab(Tab): añade una pestaña individual a la ventana. 7. getStage(): obtiene el Stage de la ventana secundaria. 8. actualizarIdioma(ResourceBundle): actualiza el idioma de la ventana secundaria. 9. getWindowNumber(): obtiene el número de la ventana.

Continúa en la página siguiente

Tabla 9.19 – continúa de la página anterior

Métodos Estáticos	<ol style="list-style-type: none"> 1. getActiveWindows(): obtiene todas las ventanas secundarias activas. 2. getActiveWindowCount(): obtiene el conteo de ventanas secundarias activas.
--------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.3.8. Dependencias internas del paquete utils

Las dependencias internas del paquete utils se ilustran en la Figura 9.2, donde se puede apreciar la estructura organizada de servicios transversales que proporciona este paquete.

- **TabManager ↔ TabPaneMonitor**: TabManager utiliza TabPaneMonitor para supervisar los TabPane que gestiona.
- **SecondaryWindow → EditorWindow**: SecondaryWindow extiende EditorWindow para crear ventanas independientes.
- **SecondaryWindow → TabManager**: SecondaryWindow utiliza TabManager para gestionar sus pestañas.
- **LanguageItem ↔ LanguageListCell**: LanguageListCell utiliza LanguageItem para mostrar idiomas en listas.
- **ActualizableTextos ← (todas las clases de UI)**: interfaz implementada por todas las clases que requieren internacionalización.
- **TabManager ↔ SecondaryWindow**: comunicación bidireccional para gestión de pestañas entre ventanas.
- **Archivos .properties**: todos los archivos de propiedades de idiomas son utilizados por el sistema de internacionalización.

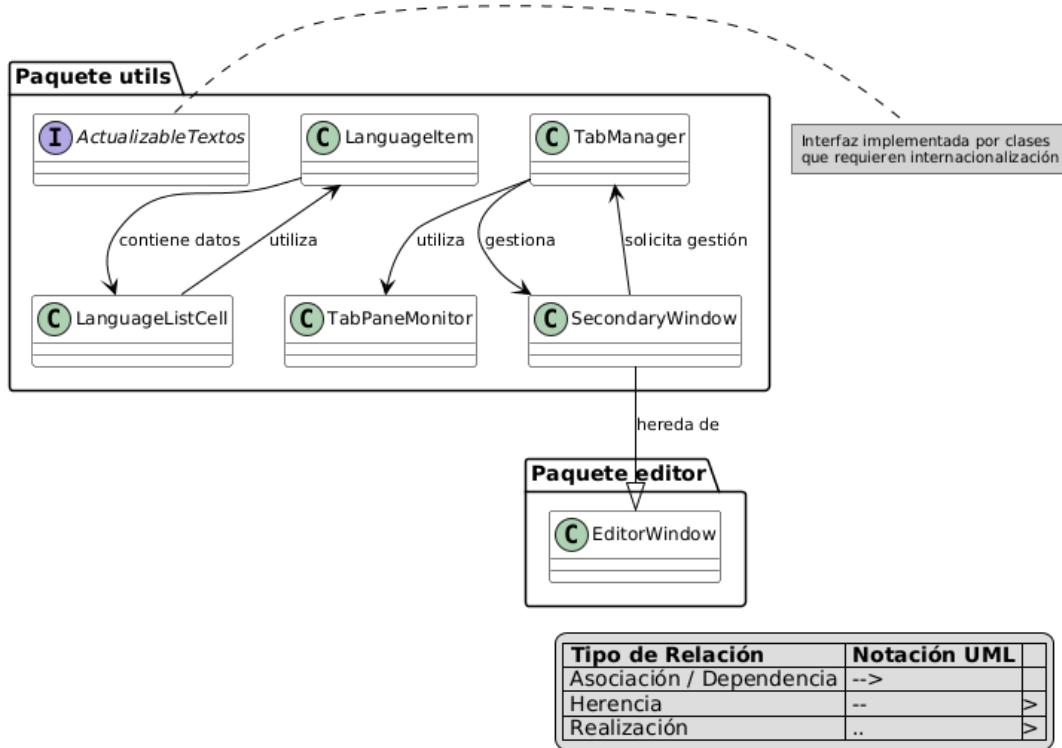


Figura 9.2: Diagrama de dependencias del paquete utils - SimAS 3.0

9.2.3.9. Patrones de diseño en el paquete utils

- **Singleton**: implementado en **TabPaneMonitor** para asegurar una única instancia que supervise toda la aplicación [gamma1995design].
- **Observer**: utilizado en **TabPaneMonitor** para observar cambios en los **TabPane** y reaccionar automáticamente [gamma1995design].
- **Facade**: **TabManager** actúa como fachada que simplifica la gestión compleja de pestanas y relaciones [gamma1995design].
- **Factory Method**: implícito en los constructores de **TabManager** para crear diferentes tipos de pestanas [gamma1995design].
- **Monitor Object**: **TabPaneMonitor** implementa el patrón Monitor Object para sincronización segura de acceso a recursos compartidos.
- **Registry**: **SecondaryWindow** utiliza un registro estático para mantener todas las ventanas secundarias activas.

9.2.4. Paquete editor

El paquete **editor** contiene las clases responsables de la interfaz de usuario para la creación y edición de gramáticas de contexto libre en SimAS 3.0. Este paquete implementa el patrón de diseño MVC donde las clases del editor actúan como controladores

que coordinan la interacción entre la vista (componentes JavaFX) y el modelo (paquete gramática).

Las clases de este paquete permiten:

- Crear gramáticas mediante un asistente paso a paso
- Editar símbolos terminales y no terminales
- Gestionar producciones gramaticales
- Validar gramáticas creadas
- Gestionar múltiples editores en pestañas
- Integración completa con el sistema de internacionalización

9.2.4.1. Clases del paquete editor

Este paquete contiene 10 clases principales que conforman la interfaz de edición de SimAS 3.0, organizadas jerárquicamente desde la gestión de ventanas hasta los componentes específicos de edición.

9.2.4.2. Clase EditorWindow

La clase **EditorWindow** representa la ventana principal del editor de gramáticas, gestionando el ciclo de vida de la aplicación de edición y coordinando múltiples pestañas de edición.

Tabla 9.20: Clase EditorWindow - Ventana principal del editor

Nombre	EditorWindow
Descripción	Gestiona la ventana principal del editor, incluyendo el sistema de pestañas, atajos de teclado y funcionalidades de arrastrar y soltar.
Atributos	<ol style="list-style-type: none"> 1. stage: Stage principal de la ventana del editor. 2. tabPane: TabPane que contiene todas las pestañas de edición. 3. bundle: ResourceBundle para internacionalización.
Continúa en la página siguiente	

Tabla 9.20 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. EditorWindow(ResourceBundle): constructor que inicializa la ventana con internacionalización. 2. initialize(): configura la ventana, escena y componentes principales. 3. configurarAtajosTeclado(Scene): establece los atajos de teclado globales. 4. findFirstTabInGroup(int): encuentra la primera pestaña de un grupo específico. 5. show(): muestra la ventana del editor. 6. addTab(Tab): añade una pestaña a la ventana. 7. moveGroupToWindow(TabPane, String, Tab): mueve un grupo de pestañas a esta ventana. 8. addEditor(Editor): añade un editor como nueva pestaña. 9. getTabPane(): obtiene el TabPane de la ventana. 10. abrirEditorEnEditorWindow(): abre un nuevo editor en esta ventana. 11. abrirSimuladorEnEditorWindow(): abre el simulador directo en esta ventana. 12. cargarGramaticaYSimular DirectamenteEnEditorWindow(): carga gramática y va directo a simulación. 13. crearSimuladorDirectoAl Paso6EnEditorWindow(Gramatica): crea simulador directo al paso 6. 14. abrirAyuda(): abre el manual de ayuda. 15. abrirTutorial(): abre el tutorial. 16. salirAplicacion(): sale de la aplicación. 17. mostrarErroresValidacion(ObservableList<String>): muestra errores de validación. 18. mostrarError(String, String): muestra mensaje de error simple.
----------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.4.3. Clase Editor

La clase **Editor** representa la interfaz principal de edición de una gramática específica, proporcionando acceso a todas las funcionalidades de edición y gestión.

Tabla 9.21: Clase Editor - Interfaz principal de edición

Nombre	Editor
Descripción	Clase principal que representa un editor de gramáticas, gestionando la interfaz de usuario y la coordinación con el modelo de datos.
Atributos	<ol style="list-style-type: none"> 1. gramatica: instancia de Gramatica que se está editando. 2. tabPane: referencia al TabPane contenedor. 3. menuPane: referencia al MenuPrincipal. 4. bundle: ResourceBundle para internacionalización. 5. editorId: identificador único del editor. 6. rootPane: BorderPane raíz de la interfaz. 7. Componentes FXML: botones, campos de texto, listas, etiquetas.

Continúa en la página siguiente

Tabla 9.21 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. Editor(): constructor vacío para carga FXML. 2. Editor(TabPane, MenuPrincipal): constructor con TabPane y MenuPrincipal. 3. Editor(TabPane, Gramatica, MenuPrincipal): constructor con gramática preexistente. 4. Editor(TabPane, MenuPrincipal, ResourceBundle): constructor con internacionalización. 5. configurarRelacionesPadreHijo(): configura las relaciones padre-hijo para cerrar pestañas hijas. 6. cargarFXML(): carga la interfaz desde archivo FXML. 7. getEditorId(): obtiene el ID único del editor. 8. getGramatica(): obtiene la gramática actual. 9. setGramatica(Gramatica): establece una nueva gramática. 10. initialize(): método de inicialización FXML. 11. setTabPane(TabPane): establece la referencia al TabPane. 12. setMenuPane(MenuPrincipal): establece la referencia al MenuPrincipal. 13. getRoot(): obtiene el componente raíz. 14. cargarGramatica(): carga una gramática desde archivo. 15. grabarGramatica(): guarda la gramática actual. 16. actualizarVisualizacion(): actualiza la visualización de los datos. 17. validarGramatica(Gramatica): valida una gramática específica. 18. actualizarTextos(ResourceBundle): actualiza textos para internacionalización.
----------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.4.4. Clase PanelCreacionGramatica

La clase **PanelCreacionGramatica** coordina el asistente de creación de gramáticas paso a paso, gestionando la navegación entre los diferentes pasos del proceso.

Tabla 9.22: Clase PanelCreacionGramatica - Asistente de creación

Nombre	PanelCreacionGramatica
Descripción	Coordina el asistente paso a paso para la creación de gramáticas, gestionando la navegación y el flujo de datos entre los diferentes paneles.
Atributos	<ol style="list-style-type: none"> 1. tabPane: TabPane contenedor de las pestañas. 2. paso1-4: instancias de los paneles de cada paso del asistente. 3. panelPadre: referencia al Editor padre. 4. menuPane: referencia al MenuPrincipal. 5. gramaticaTemporal: copia temporal de la gramática en edición. 6. bundle: ResourceBundle para internacionalización. 7. creacionId: identificador único del proceso de creación.

Continúa en la página siguiente

Tabla 9.22 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. PanelCreacionGramatica(Editor, TabPane, Gramatica, MenuPrincipal): constructor principal. 2. PanelCreacionGramatica(Editor, TabPane, Gramatica, MenuPrincipal, String): constructor con ID específico. 3. configurarRelacionesPadreHijo(): configura las relaciones padre-hijo. 4. getCreacionId(): obtiene el ID único del proceso de creación. 5. getGramatica(): obtiene la gramática temporal. 6. setGramatica(Gramatica): establece la gramática temporal. 7. cambiarPaso(int): cambia al paso especificado. 8. getMenuPane(): obtiene la referencia al MenuPrincipal. 9. getPanelPadre(): obtiene la referencia al Editor padre. 10. cancelarEdicion(): cancela la edición actual. 11. mostrarAlerta(String, String): muestra una alerta al usuario. 12. actualizarTextos(ResourceBundle): actualiza textos para internacionalización. 13. getBundle(): obtiene el ResourceBundle.
----------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.4.5. Clase PanelCreacionGramaticaPaso1

La clase **PanelCreacionGramaticaPaso1** maneja el primer paso del asistente de creación, donde se definen los metadatos básicos de la gramática.

Tabla 9.23: Clase PanelCreacionGramaticaPaso1 - Paso 1: Metadatos

Nombre	PanelCreacionGramaticaPaso1
Descripción	Primer paso del asistente de creación de gramáticas, donde se definen el nombre, descripción y símbolo inicial.
Atributos	<ol style="list-style-type: none"> 1. panelPadre: referencia al PanelCreacionGramatica padre. 2. bundle: ResourceBundle para internacionalización. 3. Componentes FXML: campos de texto para nombre, descripción y símbolo inicial. 4. btnSiguiente: botón para avanzar al siguiente paso. 5. btnCancelar: botón para cancelar la creación.
Métodos	<ol style="list-style-type: none"> 1. PanelCreacionGramaticaPaso1(PanelCreacionGramatica, ResourceBundle): constructor. 2. cargarFXML(): carga la interfaz desde archivo FXML. 3. initialize(): método de inicialización FXML. 4. configurarValidaciones(): configura las validaciones en tiempo real. 5. setNombre(String): establece el nombre de la gramática. 6. setDescripcion(String): establece la descripción de la gramática. 7. actualizarTextos(ResourceBundle): actualiza textos para internacionalización.

9.2.4.6. Clase PanelCreacionGramaticaPaso2

La clase **PanelCreacionGramaticaPaso2** maneja el segundo paso del asistente de creación, donde se definen los símbolos terminales y no terminales.

Tabla 9.24: Clase PanelCreacionGramaticaPaso2 - Paso 2: Símbolos

Nombre	PanelCreacionGramaticaPaso2
Descripción	Segundo paso del asistente donde se definen y gestionan los símbolos terminales y no terminales de la gramática.
Atributos	<ol style="list-style-type: none"> 1. panelPadre: referencia al PanelCreacionGramatica padre. 2. menuPane: referencia al MenuPrincipal. 3. tabPane: TabPane para gestión de pestañas. 4. panelTerminales: PanelSimbolosTerminales para gestión de terminales. 5. panelNoTerminales: PanelSimbolosNoTerminales para gestión de no terminales. 6. bundle: ResourceBundle para internacionalización. 7. btnSiguiente, btnAnterior, btnCancelar: botones de navegación.

Continúa en la página siguiente

Tabla 9.24 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. PanelCreacionGramaticaPaso2(PanelCreacionGramatica, MenuPrincipal, TabPane): constructor. 2. cargarFXML(): carga la interfaz desde archivo FXML. 3. initialize(): método de inicialización FXML. 4. aplicarEstilosResponsivos(): aplica estilos responsivos según el tamaño de pantalla. 5. cargarDatosDesdeGramatica(): carga datos desde la gramática padre. 6. configurarValidacionSimbolos(): configura la validación de símbolos. 7. asignarListaSimbolosNoTerminales(ObservableList<String>): asigna la lista de no terminales. 8. asignarListaSimbolosTerminales(ObservableList<String>): asigna la lista de terminales. 9. actualizarTextos(ResourceBundle): actualiza textos para internacionalización.
---------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.4.7. Clase PanelCreacionGramaticaPaso3

La clase **PanelCreacionGramaticaPaso3** maneja el tercer paso del asistente, donde se definen las producciones de la gramática.

Tabla 9.25: Clase PanelCreacionGramaticaPaso3 - Paso 3: Producciones

Nombre	PanelCreacionGramaticaPaso3
Descripción	Tercer paso del asistente donde se definen y gestionan las producciones de la gramática.
Continúa en la página siguiente	

Tabla 9.25 – continúa de la página anterior

Atributos	<ol style="list-style-type: none"> 1. panelPadre: referencia al PanelCreacionGramatica padre. 2. tabPane: TabPane para gestión de pestañas. 3. menuPane: referencia al MenuPrincipal. 4. panelProducciones: PanelProducciones para gestión de producciones. 5. bundle: ResourceBundle para internacionalización. 6. btnSiguiente, btnAnterior, btnCancelar: botones de navegación.
Métodos	<ol style="list-style-type: none"> 1. PanelCreacionGramaticaPaso3(PanelCreacionGramatica, TabPane, MenuPrincipal): constructor. 2. cargarFXML(): carga la interfaz desde archivo FXML. 3. initialize(): método de inicialización FXML. 4. asignarProducciones(ObservableList<Produccion>): asigna las producciones existentes. 5. configurarValidacionProducciones(): configura la validación de producciones. 6. actualizarTextos(ResourceBundle): actualiza textos para internacionalización.

9.2.4.8. Clase PanelCreacionGramaticaPaso4

La clase **PanelCreacionGramaticaPaso4** maneja el cuarto y último paso del asistente, donde se realiza la validación final y se completa la creación.

Tabla 9.26: Clase PanelCreacionGramaticaPaso4 - Paso 4: Validación

Nombre	PanelCreacionGramaticaPaso4
	Continúa en la página siguiente

Tabla 9.26 – continúa de la página anterior

Descripción	Cuarto y último paso del asistente donde se elige el símbolo inicial y se finaliza el proceso de creación.
Atributos	<ol style="list-style-type: none"> 1. panelPadre: referencia al PanelCreacionGramatica padre. 2. tabPane: TabPane para gestión de pestañas. 3. bundle: ResourceBundle para internacionalización. 4. Componentes FXML: etiquetas, botones de finalizar y cancelar. 5. areaValidacion: área de texto para mostrar resultados de validación.
Métodos	<ol style="list-style-type: none"> 1. PanelCreacionGramaticaPaso4(PanelCreacionGramatica, TabPane): constructor. 2. cargarFXML(): carga la interfaz desde archivo FXML. 3. initialize(): método de inicialización FXML. 4. configurarValidacionSimboloInicial(): configura la validación del símbolo inicial. 5. cerrarAsistente(): cierra el asistente de creación. 6. actualizarTextos(ResourceBundle): actualiza textos para internacionalización.

9.2.4.9. Clase PanelProducciones

La clase **PanelProducciones** proporciona la interfaz para editar y gestionar las producciones de la gramática.

Tabla 9.27: Clase PanelProducciones - Gestión de producciones

Nombre	PanelProducciones
Descripción	Panel especializado para la creación, edición y eliminación de producciones gramaticales.
Continúa en la página siguiente	

Tabla 9.27 – continúa de la página anterior

Atributos	<ol style="list-style-type: none"> 1. panelPadre: referencia al PanelCreacionGramaticaPaso3 padre. 2. tabPane: TabPane para gestión de pestañas. 3. producciones: ObservableList de producciones. 4. noTerminales: ObservableList de símbolos no terminales disponibles. 5. terminales: ObservableList de símbolos terminales disponibles. 6. bundle: ResourceBundle para internacionalización. 7. Componentes FXML: ComboBox, TextField, ListViews, Buttons, Labels.
Métodos	<ol style="list-style-type: none"> 1. PanelProducciones(PanelCreacionGramaticaPaso3, ObservableList<Produccion>, TabPane): constructor. 2. cargarFXML(): carga la interfaz desde archivo FXML. 3. initialize(): método de inicialización FXML. 4. actualizarTextos(ResourceBundle): actualiza textos para internacionalización.

9.2.4.10. Clase PanelSimbolosTerminales

La clase **PanelSimbolosTerminales** proporciona la interfaz para gestionar los símbolos terminales de la gramática.

Tabla 9.28: Clase PanelSimbolosTerminales - Gestión de terminales

Nombre	PanelSimbolosTerminales
Descripción	Panel especializado para la gestión de símbolos terminales, incluyendo símbolos predefinidos y personalizados.
Continúa en la página siguiente	

Tabla 9.28 – continúa de la página anterior

Atributos	<ol style="list-style-type: none"> 1. panelPadre: referencia al PanelCreacionGramaticaPaso2 padre. 2. tabPane: TabPane para gestión de pestañas. 3. simbolosTerminales: ObservableList de símbolos terminales. 4. simbolosTemporales: copia temporal de los símbolos. 5. simbolosSet: Set para validación de unicidad. 6. bundle: ResourceBundle para internacionalización. 7. simbolosPredefinidos: array de símbolos comunes predefinidos. 8. Componentes FXML: FlowPane, TextField, ListView, Buttons, Labels.
Continúa en la página siguiente	

Tabla 9.28 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. PanelSimbolosTerminales(ObservableList<String>, TabPane, PanelCreacionGramaticaPaso2): constructor. 2. PanelSimbolosTerminales(ObservableList<String>, TabPane, PanelCreacionGramaticaPaso2, ResourceBundle): constructor con internacionalización. 3. cargarFXML(): carga la interfaz desde archivo FXML. 4. initialize(): método de inicialización FXML. 5. generarBotonesPredefinidos(): genera botones para símbolos predefinidos. 6. agregarSimbolo(String): agrega un símbolo a la lista. 7. cerrarPestanaActual(): cierra la pestaña actual. 8. actualizarTextos(ResourceBundle): actualiza textos para internacionalización.
----------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.4.11. Clase PanelSimbolosNoTerminales

La clase **PanelSimbolosNoTerminales** proporciona la interfaz para gestionar los símbolos no terminales de la gramática.

Tabla 9.29: Clase PanelSimbolosNoTerminales - Gestión de no terminales

Nombre	PanelSimbolosNoTerminales
Descripción	Panel especializado para la gestión de símbolos no terminales de la gramática.
Continúa en la página siguiente	

Tabla 9.29 – continúa de la página anterior

Atributos	<ol style="list-style-type: none"> 1. panelPadre: referencia al PanelCreacionGramaticaPaso2 padre. 2. tabPane: TabPane para gestión de pestañas. 3. simbolosNoTerminales: ObservableList de símbolos no terminales. 4. simbolosTemporales: copia temporal de los símbolos. 5. simbolosSet: Set para validación de unicidad. 6. bundle: ResourceBundle para internacionalización. 7. Componentes FXML: TextField, ListView, Buttons, Labels.
Métodos	<ol style="list-style-type: none"> 1. PanelSimbolosNoTerminales(ObservableList<String>, TabPane, PanelCreacionGramaticaPaso2): constructor. 2. PanelSimbolosNoTerminales(ObservableList<String>, TabPane, PanelCreacionGramaticaPaso2, ResourceBundle): constructor con internacionalización. 3. cargarFXML(): carga la interfaz desde archivo FXML. 4. initialize(): método de inicialización FXML. 5. generarBotonesPredefinidos(): genera botones para símbolos predefinidos. 6. agregarSimbolo(String): agrega un símbolo a la lista. 7. cerrarPestanaActual(): cierra la pestaña actual. 8. actualizarTextos(ResourceBundle): actualiza textos para internacionalización.

9.2.4.12. Dependencias internas del paquete editor

Las dependencias internas del paquete editor se ilustran en la Figura 9.3, donde se puede apreciar la estructura jerárquica del asistente de creación y las relaciones entre los diferentes componentes de la interfaz.

- **EditorWindow → Editor:** EditorWindow crea y gestiona múltiples instancias de Editor.
- **Editor → PanelCreacionGramatica:** Editor utiliza PanelCreacionGramatica para el asistente de creación.
- **PanelCreacionGramatica → PanelCreacionGramaticaPaso1-4:** PanelCreacionGramatica coordina los cuatro pasos del asistente.
- **PanelCreacionGramaticaPaso2 → PanelSimbolosTerminales, PanelSimbolosNoTerminales:** Paso 2 utiliza paneles específicos para gestión de símbolos.
- **PanelCreacionGramaticaPaso3 → PanelProducciones:** Paso 3 utiliza PanelProducciones para gestión de producciones.
- **PanelProducciones → gramatica.Gramatica:** PanelProducciones accede a la gramática para obtener símbolos disponibles.
- **PanelSimbolosTerminales → gramatica.Terminal:** PanelSimbolosTerminales gestiona símbolos terminales.
- **PanelSimbolosNoTerminales → gramatica.NoTerminal:** PanelSimbolosNoTerminales gestiona símbolos no terminales.
- **Todas las clases → utils.ActualizableTextos:** todas las clases del editor implementan esta interfaz para internacionalización.
- **Todas las clases → bienvenida.MenuPrincipal:** acceso al menú principal para navegación.
- **Todas las clases → utils.TabManager:** gestión de pestañas y grupos de gramáticas.

9.2.4.13. Patrones de diseño en el paquete editor

- **MVC (Model-View-Controller):** las clases del editor actúan como controladores coordinando entre las vistas JavaFX y el modelo gramática [burbeck1992applications].
- **Observer:** implementado mediante JavaFX Properties para notificación automática de cambios en la UI [gamma1995design].
- **Strategy:** los diferentes paneles (PanelProducciones, PanelSimbolosTerminales, etc.) implementan estrategias específicas de edición [gamma1995design].
- **Factory Method:** los constructores de las clases crean instancias con diferentes configuraciones iniciales [gamma1995design].

- **Composite:** la estructura jerárquica de paneles (PanelCreacionGramatica conteniendo PanelCreacionGramaticaPaso1-4) [**gamma1995design**].
- **Singleton:** uso implícito en la gestión de recursos compartidos como ResourceBundle [**gamma1995design**].

9.2.5. Paquete simulador

El paquete **simulador** contiene las clases responsables de la simulación del análisis sintáctico descendente en SimAS 3.0. Este paquete implementa el núcleo funcional de la aplicación, permitiendo a los usuarios simular paso a paso el proceso de análisis sintáctico de cadenas de entrada utilizando gramáticas libres de contexto.

Las clases de este paquete permiten:

- Simulación completa del análisis sintáctico descendente
- Visualización paso a paso del proceso de análisis
- Gestión de funciones de error y recuperación
- Creación y edición de cadenas de entrada
- Interfaz interactiva para la simulación
- Generación de informes y árboles de derivación

9.2.5.1. Clases del paquete simulador

Este paquete contiene 13 clases principales que implementan el sistema completo de simulación de análisis sintáctico, organizadas por funcionalidad: interfaz de pasos, clases principales de simulación, pasos de la simulación descendente y componentes auxiliares.

9.2.5.2. Interfaz PanelNuevaSimDescPaso

La interfaz **PanelNuevaSimDescPaso** define el contrato que deben implementar todos los pasos de la simulación descendente.

Tabla 9.30: Interfaz PanelNuevaSimDescPaso - Interfaz de pasos de simulación

Nombre	PanelNuevaSimDescPaso
Descripción	Interfaz que define el contrato para todos los pasos de la simulación descendente, asegurando consistencia en la implementación.
	Continúa en la página siguiente

Tabla 9.30 – continúa de la página anterior

Métodos	1. getRoot() : retorna el nodo raíz del paso de simulación.
----------------	--------------------------------------------------------------------

9.2.5.3. Clase PanelNuevaSimDescPaso1

La clase **PanelNuevaSimDescPaso1** implementa el primer paso de la simulación descendente, mostrando la gramática original y permitiendo la navegación entre pasos.

Tabla 9.31: Clase PanelNuevaSimDescPaso1 - Paso 1 de simulación

Nombre	PanelNuevaSimDescPaso1
Descripción	Implementa el primer paso de la simulación descendente, mostrando la gramática original y permitiendo navegación entre los diferentes pasos del proceso.
Atributos	<ol style="list-style-type: none"> 1. panelPadre: PanelSimuladorDesc que contiene este paso. 2. gramatica: Gramatica que se está simulando. 3. bundle: ResourceBundle para internacionalización. 4. root: Parent que representa la interfaz gráfica del paso.
Métodos	<ol style="list-style-type: none"> 1. PanelNuevaSimDescPaso1(PanelSimuladorDesc): constructor que inicializa el paso con su panel padre. 2. getRoot(): retorna el nodo raíz de la interfaz gráfica. 3. actualizarTextos(ResourceBundle): actualiza los textos según el idioma seleccionado. 4. cargarFXML(): carga el archivo FXML correspondiente al paso. 5. inicializarBotones(): configura los botones de navegación.

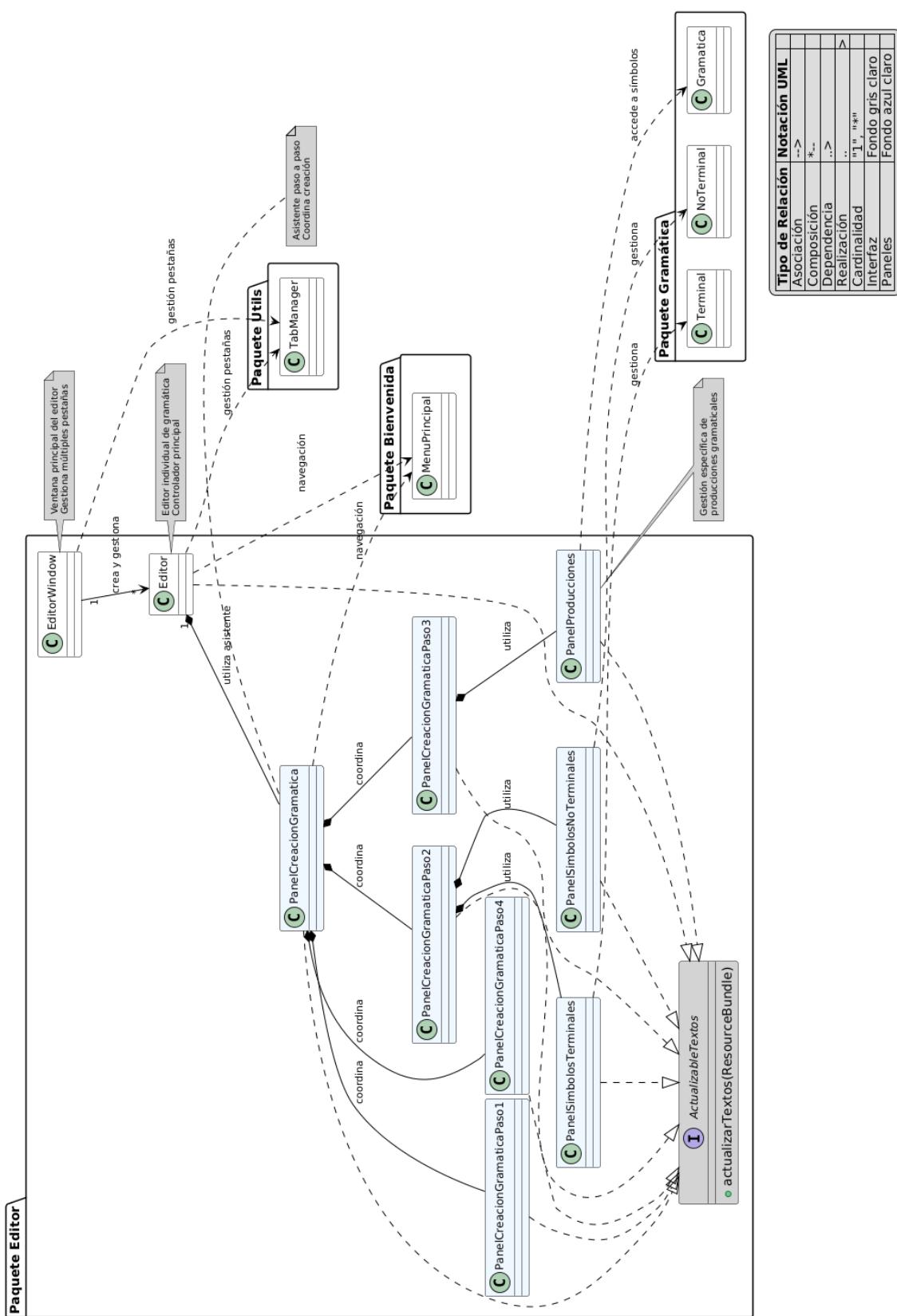


Figura 9.3: Diagrama de clases del paquete editor - SimAS 3.0

9.2.5.4. Clase PanelNuevaSimDescPaso2

La clase **PanelNuevaSimDescPaso2** implementa el segundo paso de la simulación descendente, mostrando los símbolos terminales y no terminales.

Tabla 9.32: Clase PanelNuevaSimDescPaso2 - Paso 2 de simulación

Nombre	PanelNuevaSimDescPaso2
Descripción	Implementa el segundo paso de la simulación descendente, mostrando los símbolos terminales y no terminales de la gramática.
Atributos	<ol style="list-style-type: none"> 1. panelPadre: PanelSimuladorDesc que contiene este paso. 2. gramatica: Gramatica que se está simulando. 3. bundle: ResourceBundle para internacionalización. 4. root: Parent que representa la interfaz gráfica del paso.
Métodos	<ol style="list-style-type: none"> 1. PanelNuevaSimDescPaso2(PanelSimuladorDesc): constructor que inicializa el paso con su panel padre. 2. getRoot(): retorna el nodo raíz de la interfaz gráfica. 3. actualizarTextos(ResourceBundle): actualiza los textos según el idioma seleccionado.

9.2.5.5. Clase PanelNuevaSimDescPaso3

La clase **PanelNuevaSimDescPaso3** implementa el tercer paso de la simulación descendente, mostrando los conjuntos First y Follow.

Tabla 9.33: Clase PanelNuevaSimDescPaso3 - Paso 3 de simulación

Nombre	PanelNuevaSimDescPaso3
Descripción	Implementa el tercer paso de la simulación descendente, mostrando los conjuntos First y Follow de los símbolos no terminales.
	Continúa en la página siguiente

Tabla 9.33 – continúa de la página anterior

Atributos	<ol style="list-style-type: none"> 1. panelPadre: PanelSimuladorDesc que contiene este paso. 2. gramatica: Gramatica que se está simulando. 3. bundle: ResourceBundle para internacionalización. 4. root: Parent que representa la interfaz gráfica del paso.
Métodos	<ol style="list-style-type: none"> 1. PanelNuevaSimDescPaso3(PanelSimuladorDesc): constructor que inicializa el paso con su panel padre. 2. getRoot(): retorna el nodo raíz de la interfaz gráfica. 3. actualizarTextos(ResourceBundle): actualiza los textos según el idioma seleccionado.

9.2.5.6. Clase PanelNuevaSimDescPaso4

La clase **PanelNuevaSimDescPaso4** implementa el cuarto paso de la simulación descendente, mostrando la tabla predictiva.

Tabla 9.34: Clase PanelNuevaSimDescPaso4 - Paso 4 de simulación

Nombre	PanelNuevaSimDescPaso4
Descripción	Implementa el cuarto paso de la simulación descendente, mostrando la tabla predictiva completa con sus celdas y funciones de error.
Atributos	<ol style="list-style-type: none"> 1. panelPadre: PanelSimuladorDesc que contiene este paso. 2. gramatica: Gramatica que se está simulando. 3. bundle: ResourceBundle para internacionalización. 4. root: Parent que representa la interfaz gráfica del paso.
Continúa en la página siguiente	

Tabla 9.34 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. PanelNuevaSimDescPaso4(PanelSimuladorDesc): constructor que inicializa el paso con su panel padre. 2. getRoot(): retorna el nodo raíz de la interfaz gráfica. 3. funcionError(): gestiona las funciones de error de la gramática. 4. reordenarIndicesFuncionesError(): reordena los índices de las funciones de error. 5. actualizarTextos(ResourceBundle): actualiza los textos según el idioma seleccionado.
----------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.5.7. Clase PanelNuevaSimDescPaso5

La clase **PanelNuevaSimDescPaso5** implementa el quinto paso de la simulación descendente, permitiendo la edición de funciones de error.

Tabla 9.35: Clase PanelNuevaSimDescPaso5 - Paso 5 de simulación

Nombre	PanelNuevaSimDescPaso5
Descripción	Implementa el quinto paso de la simulación descendente, permitiendo la edición y configuración de funciones de error para recuperación de errores.
Atributos	<ol style="list-style-type: none"> 1. panelPadre: PanelSimuladorDesc que contiene este paso. 2. gramatica: Gramatica que se está simulando. 3. bundle: ResourceBundle para internacionalización. 4. root: Parent que representa la interfaz gráfica del paso.
Continúa en la página siguiente	

Tabla 9.35 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. PanelNuevaSimDescPaso5(PanelSimuladorDesc): constructor que inicializa el paso con su panel padre. 2. getRoot(): retorna el nodo raíz de la interfaz gráfica. 3. funcionErrorToString(FuncionError): convierte una función de error a string para mostrar. 4. refrescarVista(): refresca la vista de la tabla predictiva. 5. guardarDatosTabla(): guarda los datos de la tabla predictiva. 6. getFuncionErrorSeleccionada(): obtiene la función de error seleccionada. 7. getBundle(): obtiene el ResourceBundle para internacionalización. 8. actualizarTextos(ResourceBundle): actualiza los textos según el idioma seleccionado.
---------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.5.8. Clase PanelNuevaSimDescPaso6

La clase **PanelNuevaSimDescPaso6** implementa el sexto y último paso de la simulación descendente, que corresponde al **Simulador** principal donde se ejecuta la simulación completa del análisis sintáctico.

Tabla 9.36: Clase PanelNuevaSimDescPaso6 - Paso 6 (Simulador) de simulación

Nombre	PanelNuevaSimDescPaso6
Descripción	Implementa el sexto y último paso de la simulación descendente (el Simulador principal), permitiendo ejecutar la simulación completa del análisis sintáctico.
Continúa en la página siguiente	

Tabla 9.36 – continúa de la página anterior

Atributos	<ol style="list-style-type: none"> 1. panelPadre: PanelSimuladorDesc que contiene este paso. 2. gramatica: Gramatica que se está simulando. 3. bundle: ResourceBundle para internacionalización. 4. root: Parent que representa la interfaz gráfica del paso.
Métodos	<ol style="list-style-type: none"> 1. PanelNuevaSimDescPaso6(Gramatica, PanelSimuladorDesc): constructor principal con gramática y panel padre. 2. PanelNuevaSimDescPaso6(Gramatica): constructor alternativo con solo gramática. 3. getRoot(): retorna el nodo raíz de la interfaz gráfica. 4. actualizarVisualizacion(): actualiza la visualización de la simulación. 5. actualizarTextos(ResourceBundle): actualiza los textos según el idioma seleccionado.

9.2.5.9. Clase PanelSimuladorDesc

La clase **PanelSimuladorDesc** es el controlador principal para la simulación descendente, coordinando todos los pasos del proceso de análisis sintáctico.

Tabla 9.37: Clase PanelSimuladorDesc - Controlador principal de simulación

Nombre	PanelSimuladorDesc
Descripción	Controlador principal que coordina toda la simulación descendente, gestionando los diferentes pasos, la gramática y la interacción con el usuario.
Continúa en la página siguiente	

Tabla 9.37 – continúa de la página anterior

Atributos	<ol style="list-style-type: none"> 1. tabPane: TabPane donde se muestran los pasos de simulación. 2. gramatica: Gramatica que se está simulando. 3. gramaticaOriginal: copia independiente de la gramática original. 4. pasoActual: número del paso actual en la simulación. 5. pasos: ArrayList con todos los pasos de la simulación. 6. bundle: ResourceBundle para internacionalización. 7. tablaPredictivaExtendidaGlobal: TablaPredictivaPaso5 con funciones de error. 8. simuladorId: identificador único del simulador. 9. simuladoresActivos: Map estático con todos los simuladores activos.
Continúa en la página siguiente	

Tabla 9.37 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. PanelSimuladorDesc(Gramatica, TabPane, ResourceBundle): constructor principal. 2. PanelSimuladorDesc(Gramatica, TabPane, ResourceBundle, String): constructor con ID personalizado. 3. PanelSimuladorDesc(Gramatica, TabPane, MenuPrincipal, String, ResourceBundle): constructor completo. 4. getTablaPredictivaExtendidaGlobal(): obtiene la tabla predictiva extendida global. 5. setTablaPredictivaExtendidaGlobal(TablaPredictivaPaso5): establece la tabla predictiva extendida global. 6. getGramaticaOriginal(): obtiene la gramática original. 7. mostrarGramaticaOriginal(): muestra la gramática original. 8. cancelarSimulacion(): cancela la simulación actual. 9. cambiarPaso(int): cambia al paso especificado. 10. cerrarPestañaFuncionesError(): cierra la pestaña de funciones de error. 11. getBundle(): obtiene el ResourceBundle. 12. setBundle(ResourceBundle): establece el ResourceBundle. 13. getRoot(): obtiene el nodo raíz. 14. getPasoActual(): obtiene el paso actual. 15. configurarRelacionesPadreHijo(): configura las relaciones padre-hijo. 16. getSimuladorId(): obtiene el ID único del simulador.
Continúa en la página siguiente	

Tabla 9.37 – continúa de la página anterior

Métodos Estáticos	<ol style="list-style-type: none"> 1. actualizarTodosLosSimuladores(ResourceBundle): actualiza todos los simuladores activos. 2. desregarstrarSimulador(String): desregistra un simulador del registro global. 3. obtenerSimulador(String): obtiene un simulador por su ID.
--------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.5.10. Clase PanelSimulacion

La clase **PanelSimulacion** implementa un panel completo para la simulación interactiva del análisis sintáctico.

Tabla 9.38: Clase PanelSimulacion - Panel de simulación interactiva

Nombre	PanelSimulacion
Descripción	Panel que proporciona una interfaz completa para la simulación interactiva del análisis sintáctico, permitiendo controlar paso a paso el proceso de análisis.
Continúa en la página siguiente	

Tabla 9.38 – continúa de la página anterior

Atributos	<ol style="list-style-type: none"> 1. gramatica: Gramatica que se está simulando. 2. tablaPredictiva: TablaPredictivaPaso5 con funciones de error. 3. funcionesError: List de FuncionError disponibles. 4. entrada: String que representa la cadena de entrada. 5. pila: Stack que simula la pila del analizador. 6. entradaActual: List que contiene los símbolos de entrada. 7. posicionEntrada: posición actual en la cadena de entrada. 8. simulacionEnCurso: boolean que indica si la simulación está activa. 9. bundle: ResourceBundle para internacionalización.
Métodos	<ol style="list-style-type: none"> 1. PanelSimulacion(Gramatica, ResourceBundle): constructor que iniciaiza el panel de simulación. 2. getRoot(): obtiene el componente raíz VBox de la interfaz.

9.2.5.11. Clase SimulacionFinal

La clase **SimulacionFinal** implementa la simulación completa del análisis sintáctico con capacidades avanzadas de visualización y control.

Tabla 9.39: Clase SimulacionFinal - Simulación completa avanzada

Nombre	SimulacionFinal
Descripción	Implementa la simulación completa del análisis sintáctico con capacidades avanzadas como historial, navegación, generación de informes y visualización de árboles de derivación.
Continúa en la página siguiente	

Tabla 9.39 – continúa de la página anterior

Atributos	<ol style="list-style-type: none">1. gramatica: Gramatica que se está simulando.2. tablaPredictiva: TablaPredictivaPaso5 con funciones de error.3. funcionesError: List de FuncionError para recuperación.4. pila: Stack que simula la pila del analizador.5. entradaActual: ArrayList con la cadena de entrada.6. historialPasos: ObservableList con el historial de pasos.7. pasoActual: posición actual en la simulación.8. bundle: ResourceBundle para internacionalización.
Continúa en la página siguiente	

Tabla 9.39 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. SimulacionFinal(Gramatica, TablaPredictivaPaso5, TabPane, ResourceBundle): constructor principal. 2. setSimulacionId(String): establece el ID único de la simulación. 3. actualizarGrupoYTitulo(): actualiza el grupo y título de la simulación. 4. actualizarTitulosPestañas(int, boolean, int, boolean): actualiza títulos de pestañas con parámetros. 5. actualizarTitulosPestañas(): actualiza títulos de pestañas sin parámetros. 6. perteneceASimulador(String): verifica si pertenece a un simulador específico. 7. esHijaDeLaSimulacion(Tab): verifica si una pestaña es hija de la simulación. 8. getSimuladorPadreId(): obtiene el ID del simulador padre. 9. setGroupId(String): establece el ID del grupo. 10. setNumeroGrupo(int): establece el número del grupo. 11. getGroupId(): obtiene el ID del grupo. 12. getNumeroGrupo(): obtiene el número del grupo. 13. actualizarTextos(ResourceBundle): actualiza textos según el idioma.
---------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.5.12. Clase PanelGramaticaOriginal

La clase **PanelGramaticaOriginal** muestra la gramática original en una pestaña separada con soporte para internacionalización.

Tabla 9.40: Clase PanelGramaticaOriginal - Panel de gramática original

Nombre	PanelGramaticaOriginal
Continúa en la página siguiente	

Tabla 9.40 – continúa de la página anterior

Descripción	Panel que muestra la gramática original en una pestaña separada, permitiendo visualizar todas las producciones de forma clara y organizada.
Atributos	<ol style="list-style-type: none"> 1. gramaticaOriginal: Gramatica original que se muestra. 2. bundle: ResourceBundle para internacionalización.
Métodos	<ol style="list-style-type: none"> 1. PanelGramaticaOriginal(Gramatica, ResourceBundle): constructor que inicializa el panel. 2. getRoot(): retorna el nodo raíz del panel. 3. actualizarTextos(ResourceBundle): actualiza los textos según el idioma seleccionado.

9.2.5.13. Clase NuevaFuncionError

La clase **NuevaFuncionError** permite crear y configurar nuevas funciones de error para la recuperación en el análisis sintáctico.

Tabla 9.41: Clase NuevaFuncionError - Creación de funciones de error

Nombre	NuevaFuncionError
Descripción	Permite crear y configurar nuevas funciones de error para la recuperación automática durante el análisis sintáctico descendente.
Atributos	<ol style="list-style-type: none"> 1. gramatica: Gramatica donde se aplicarán las funciones de error. 2. paso4: PanelNuevaSimDescPaso4 que contiene este componente. 3. bundle: ResourceBundle para internacionalización. 4. root: Parent que representa la interfaz gráfica.

Continúa en la página siguiente

Tabla 9.41 – continúa de la página anterior

Métodos	<ol style="list-style-type: none"> 1. NuevaFuncionError(Gramatica, PanelNuevaSimDescPaso4, ResourceBundle): constructor principal. 2. getRoot(): retorna el nodo raíz de la interfaz. 3. actualizarTextos(ResourceBundle): actualiza los textos según el idioma seleccionado.
---------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.2.5.14. Clase EditorCadenaEntradaController

La clase **EditorCadenaEntradaController** proporciona una interfaz para editar y gestionar las cadenas de entrada que se utilizarán en la simulación.

Tabla 9.42: Clase EditorCadenaEntradaController - Editor de cadenas de entrada

Nombre	EditorCadenaEntradaController
Descripción	Controlador FXML que gestiona la edición de cadenas de entrada mediante una interfaz gráfica con lista de terminales disponibles y campo de texto para composición de la cadena.
Atributos	<ol style="list-style-type: none"> 1. stage: Stage que contiene la ventana del editor. 2. resultado: String que contiene el resultado final de la edición.
Métodos	<ol style="list-style-type: none"> 1. setStage(Stage): establece la referencia al Stage de la ventana. 2. setTerminales(List<Terminal>): establece la lista de terminales disponibles. 3. setCadenaInicial(String): establece la cadena inicial en el campo de texto. 4. getResultado(): obtiene el resultado final de la edición.

9.2.5.15. Dependencias internas del paquete simulador

Las dependencias internas del paquete simulador se ilustran en la Figura 9.4, donde se puede apreciar la estructura organizada del sistema de simulación paso a paso y sus

interacciones con otros paquetes.

- **PanelSimuladorDesc → PanelNuevaSimDescPaso1-6**: PanelSimuladorDesc contiene y coordina todos los pasos de simulación, siendo PanelNuevaSimDescPaso6 el Simulador principal.
- **PanelNuevaSimDescPaso1-6 → PanelNuevaSimDescPaso**: todos los pasos implementan la interfaz PanelNuevaSimDescPaso.
- **PanelSimuladorDesc ↔ Gramatica**: utiliza la gramática para la simulación.
- **PanelSimuladorDesc ↔ TablaPredictivaPaso5**: utiliza la tabla predictiva extendida.
- **SimulacionFinal → Gramatica**: utiliza la gramática para simulación completa.
- **PanelSimulacion → TablaPredictivaPaso5**: utiliza la tabla predictiva para simulación interactiva.
- **NuevaFuncionError → PanelNuevaSimDescPaso4**: se integra en el paso 4 de la simulación.
- **PanelGramaticaOriginal → Gramatica**: muestra la gramática original.
- **EditorCadenaEntradaController → Gramatica**: valida cadenas contra la gramática.
- **Todas las clases → ResourceBundle**: utilizan internacionalización.

9.2.5.16. Patrones de diseño en el paquete simulador

- **MVC (Model-View-Controller)**: el paquete sigue el patrón MVC donde las clases del paquete gramática son el Modelo, las clases del simulador son los Controladores, y los componentes FXML son las Vistas [burbeck1992applications].
- **Strategy**: los diferentes pasos de simulación (PanelNuevaSimDescPaso1-6) implementan estrategias diferentes para cada fase del análisis [gamma1995design].
- **Observer**: las clases implementan ActualizableTextos para observar cambios en el idioma y actualizar automáticamente la interfaz [gamma1995design].
- **Factory Method**: los constructores de PanelSimuladorDesc crean diferentes configuraciones de simuladores según los parámetros [gamma1995design].
- **Registry**: PanelSimuladorDesc mantiene un registro estático de todos los simuladores activos para gestión centralizada.

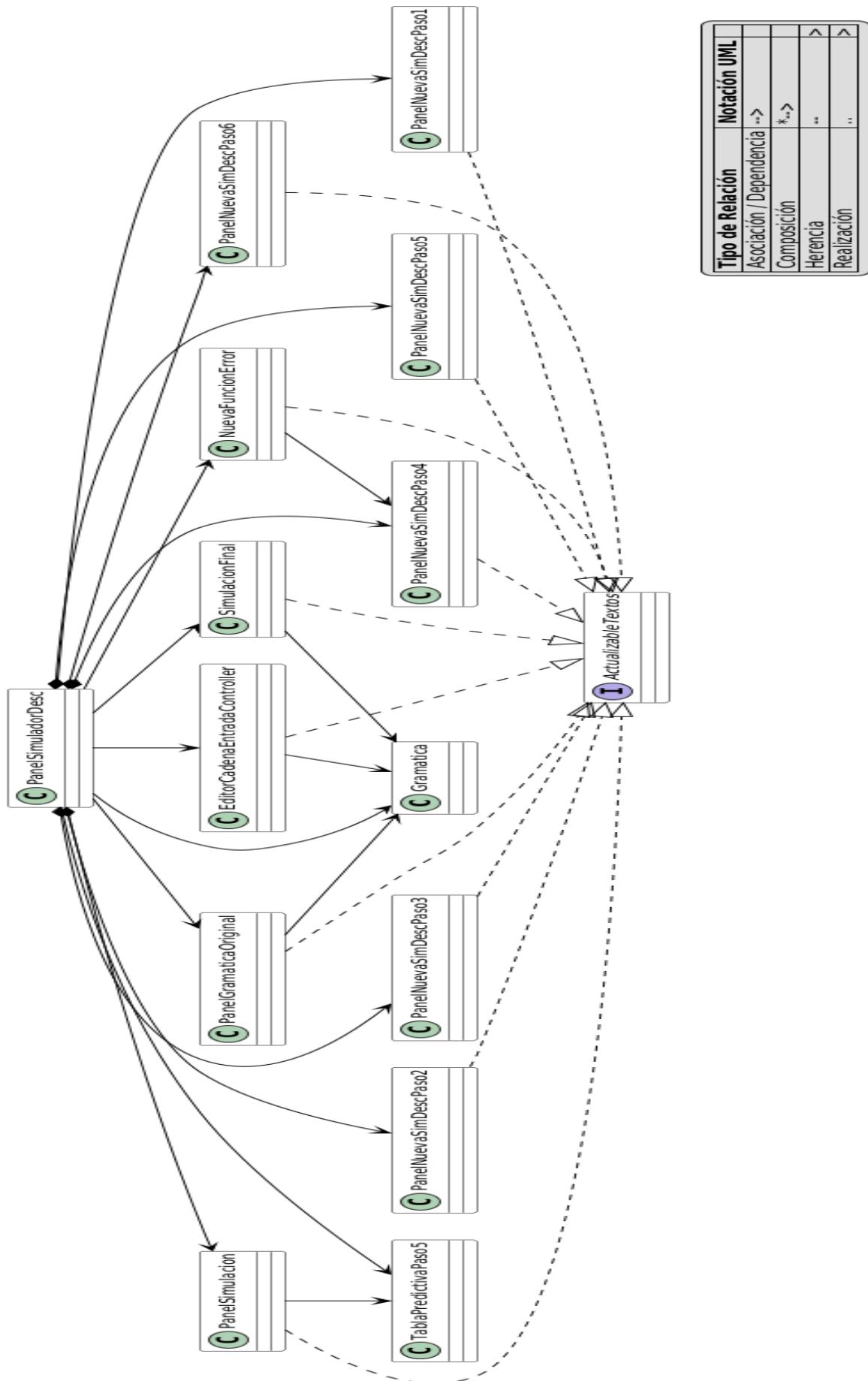


Figura 9.4: Diagrama de dependencias del paquete simulador - SimAS 3.0

9.3. Paquetes de soporte

9.3.1. Paquete centroayuda

El paquete **centroayuda** contiene recursos y componentes relacionados con el sistema de ayuda y tutoriales de la aplicación SimAS 3.0. Este paquete incluye:

- **Contenido HTML:** archivos de ayuda interactiva y tutoriales accesibles desde la aplicación.
- **Recursos multimedia:** imágenes, diagramas y elementos visuales que complementan la documentación.
- **Archivos PDF:** manuales técnicos y guías de usuario en formato PDF para consulta offline.
- **Documentación integrada:** sistema de ayuda contextual que se activa desde diferentes puntos de la interfaz.

Este paquete proporciona soporte integral al usuario, desde tutoriales interactivos hasta documentación técnica detallada, facilitando el aprendizaje y uso efectivo de la aplicación.

9.3.2. Paquete vistas

El paquete **vistas** contiene todos los archivos de interfaz de usuario definidos en formato FXML (JavaFX XML), que constituyen la capa de presentación de la aplicación SimAS 3.0. Este paquete incluye:

- **Archivos FXML principales:** definiciones de las interfaces principales como Bienvenida, MenuPrincipal y Editor.
- **Componentes reutilizables:** paneles y controles personalizados utilizados en múltiples secciones.
- **Interfaces de simulación:** archivos FXML para cada paso del proceso de simulación descendente.
- **Hojas de estilo:** archivos CSS que definen el aspecto visual y temas de la aplicación.

Este paquete implementa completamente la separación entre lógica de negocio y presentación visual, siguiendo el patrón MVC. Una descripción más detallada de la estructura FXML, organización de componentes y diseño de interfaces se presenta en el Capítulo 10 de este manual.

9.3.3. Paquete resources

El paquete **resources** centraliza todos los recursos estáticos y elementos multimedia utilizados por la aplicación SimAS 3.0. Este paquete incluye:

- **Imágenes de interfaz:** iconos, botones y elementos gráficos utilizados en la interfaz de usuario.
- **Recursos de internacionalización:** archivos de propiedades (.properties) para soporte multiidioma (español, inglés, francés, japonés, portugués).
- **Elementos visuales:** imágenes decorativas, logos y componentes gráficos de la aplicación.
- **Archivos de configuración:** recursos adicionales necesarios para el funcionamiento correcto de la aplicación.

Este paquete garantiza que todos los recursos necesarios estén organizados de manera eficiente y sean fácilmente accesibles desde cualquier parte de la aplicación, facilitando el mantenimiento y actualización de elementos visuales e internacionalización.

9.4. Dependencias entre Paquetes

Esta sección presenta un análisis detallado de las dependencias entre los diferentes paquetes del sistema SimAS 3.0, mostrando cómo se interconectan las clases para formar una arquitectura coherente y modular.

9.4.1. Dependencias Detalladas por Paquete

9.4.1.1. Dependencias del paquete bienvenida

- **bienvenida.Bienvenida → bienvenida.MenuPrincipal:** la clase Bienvenida transita automáticamente al MenuPrincipal después de mostrar la pantalla de bienvenida.
- **bienvenida.MenuPrincipal → editor.EditorWindow:** MenuPrincipal crea instancias de EditorWindow para iniciar editores.
- **bienvenida.MenuPrincipal → simulador.PanelSimuladorDesc:** MenuPrincipal crea instancias del simulador descendente.
- **bienvenida.MenuPrincipal → utils.TabManager:** MenuPrincipal utiliza TabManager para gestión avanzada de pestañas.
- **bienvenida.MenuPrincipal → utils.LanguageItem:** MenuPrincipal utiliza LanguageItem para el selector de idiomas.

- **bienvenida.MenuPrincipal** → **java.util.ResourceBundle**: MenuPrincipal gestiona la internacionalización mediante ResourceBundle.
- **bienvenida.MenuPrincipal** → **utils.SecondaryWindow**: MenuPrincipal puede crear ventanas secundarias para gestión de pestañas.
- **bienvenida.MenuPrincipal** → **centroayuda**: MenuPrincipal accede a recursos de ayuda cuando el usuario solicita asistencia.

9.4.1.2. Dependencias del paquete editor

- **editor.*** → **gramatica.Gramatica**: todas las clases del editor utilizan la clase Gramatica como modelo de datos central.
- **editor.*** → **utils.TabManager**: las clases del editor utilizan TabManager para gestión de pestañas y grupos.
- **editor.*** → **utils.ActualizableTextos**: las clases del editor implementan esta interfaz para internacionalización.
- **editor.PanelCreacionGramatica** → **editor.PanelCreacionGramaticaPaso***: PanelCreacionGramatica coordina todos los pasos del asistente.
- **editor.*** → **bienvenida.MenuPrincipal**: las clases del editor referencian al MenuPrincipal para navegación.
- **editor.PanelProducciones** → **gramatica.Produccion**: PanelProducciones gestiona objetos Produccion.
- **editor.PanelSimbolos*** → **gramatica.Terminal/NoTerminal**: los paneles de símbolos gestionan terminales y no terminales.
- **editor.*** → **utils.ResourceBundle**: las clases del editor utilizan recursos de internacionalización.

9.4.1.3. Dependencias del paquete gramática

- **gramatica.*** → **gramatica.Simbolo**: todas las clases gramaticales heredan o utilizan la clase base Simbolo.
- **gramatica.Gramatica** → **gramatica.Terminal/NoTerminal/Produccion**: Gramatica contiene colecciones de estos elementos.
- **gramatica.Gramatica** → **gramatica.TablaPredictiva**: Gramatica utiliza TablaPredictiva para análisis sintáctico.
- **gramatica.TablaPredictiva** → **gramatica.FilaTablaPredictiva**: TablaPredictiva contiene filas de tabla predictiva.
- **gramatica.TablaPredictiva** → **gramatica.FuncionError**: TablaPredictiva gestiona funciones de error.

- **gramatica.TablaPredictivaPaso5** → **gramatica.TablaPredictiva**: herencia y extensión de funcionalidad.
- **gramatica.*** → **java.util.ObservableList**: las clases gramaticales utilizan ObservableList para notificación automática.

9.4.1.4. Dependencias del paquete simulador

- **simulador.*** → **gramatica.Gramatica**: todas las clases del simulador utilizan Gramatica como modelo.
- **simulador.*** → **gramatica.TablaPredictivaPaso5**: los simuladores utilizan la tabla predictiva extendida.
- **simulador.PanelSimuladorDesc** → **simulador.PanelNuevaSimDescPaso***: PanelSimuladorDesc coordina todos los pasos.
- **simulador.PanelNuevaSimDescPaso*** → **simulador.PanelNuevaSimDescPaso**: implementan la interfaz común.
- **simulador.*** → **utils.ActualizableTextos**: las clases del simulador implementan internacionalización.
- **simulador.PanelNuevaSimDescPaso4** → **simulador.NuevaFuncionError**: Paso 4 utiliza NuevaFuncionError para gestión de errores.
- **simulador.*** → **utils.TabManager**: los simuladores utilizan gestión de pestañas.
- **simulador.*** → **utils.ResourceBundle**: los simuladores utilizan recursos de internacionalización.

9.4.1.5. Dependencias del paquete utils

- **utils.TabManager** → **utils.TabPaneMonitor**: TabManager utiliza el monitor para supervisión.
- **utils.SecondaryWindow** → **utils.TabManager**: SecondaryWindow utiliza gestión de pestañas.
- **utils.*** → **utils.ActualizableTextos**: las clases utils implementan o utilizan esta interfaz.
- **utils.TabPaneMonitor** → **javafx.scene.control.TabPane**: el monitor supervisa TabPanes.
- **utils.*** → **java.util.ResourceBundle**: las clases utils utilizan internacionalización.
- **utils.LanguageItem** → **java.util.ResourceBundle**: LanguageItem utiliza recursos de idioma.

9.4.1.6. Dependencias transversales del sistema

- **Todos los paquetes → utils.ActualizableTextos:** interfaz implementada por clases de todos los paquetes para internacionalización.
- **Todos los paquetes → java.util.ResourceBundle:** todos los paquetes utilizan recursos de internacionalización.
- **Todos los paquetes → utils.TabManager:** gestión centralizada de pestañas utilizada por múltiples paquetes.
- **Paquet vista → Todos los paquetes:** los archivos FXML referencian controladores de todos los paquetes.
- **Paquet resources → Todos los paquetes:** recursos utilizados por clases de todos los paquetes.

9.4.2. Análisis de acoplamiento

9.4.2.1. Acoplamiento bajo (recomendable)

- **Interfaces:** el uso de interfaces como ActualizableTextos y PanelNuevaSimDescPaso reduce el acoplamiento.
- **Inyección de dependencias:** las clases reciben sus dependencias a través de constructores, facilitando testing.
- **Patrones de diseño:** MVC, Strategy, Observer y Factory Method promueven bajo acoplamiento.

9.4.2.2. Acoplamiento alto (áreas de atención)

- **Dependencia circular:** algunos paquetes tienen dependencias bidireccionales que podrían refactorizarse.
- **Dependencias transversales:** ResourceBundle y TabManager son utilizados por muchos paquetes.
- **Dependencias específicas:** algunas clases tienen dependencias muy específicas que limitan la reutilización.

9.4.3. Principios SOLID aplicados

En esta sección se analiza la aplicación de los principios SOLID (Single Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, Dependency Inversion) en la arquitectura de SimAS 3.0, con referencias específicas a clases concretas y fundamentación teórica.

9.4.3.1. Principio de Responsabilidad Única (SRP)

El Principio de Responsabilidad Única establece que una clase debe tener una sola razón para cambiar, es decir, debe tener una única responsabilidad [martin2003agile, martin2018clean].

- **Clase gramatica.Gramatica:** responsable únicamente de gestionar la lógica de datos de una gramática (terminales, no terminales, producciones). No maneja presentación ni persistencia.
 - *Ejemplo:* métodos como `validarGramatica()`, `generarConjPrim()`, `generarConjSig()` se centran exclusivamente en la lógica gramatical.
- **Clase utils.TabManager:** responsable únicamente de la gestión centralizada de pestañas y grupos. No conoce detalles específicos de editores o simuladores.
 - *Ejemplo:* métodos como `getOrCreateTab()`, `asignarElementoANuevoGrupo()` manejan exclusivamente lógica de pestañas.
- **Clase editor.PanelCreacionGramatica:** responsable únicamente de coordinar el asistente de creación paso a paso. No implementa la lógica de cada paso.
 - *Ejemplo:* método `cambiarPaso(int)` coordina navegación, mientras que cada `PanelCreacionGramaticaPaso*` implementa su propio paso.
- **Clase simulador.PanelSimuladorDesc:** responsable únicamente de orquestar la simulación descendente. No implementa la lógica de cada paso individual.
 - *Ejemplo:* gestiona el estado global de la simulación (`pasoActual`, `pasos`) sin conocer detalles de análisis sintáctico.

9.4.3.2. Principio de Abierto/Cerrado (OCP)

El Principio de Abierto/Cerrado establece que las entidades de software deben estar abiertas para extensión pero cerradas para modificación [meyer1988object, martin2003agile].

- **Interfaz simulador.PanelNuevaSimDescPaso:** permite agregar nuevos pasos de simulación sin modificar el código existente del simulador principal.
 - *Ejemplo:* para agregar un nuevo paso 7, solo se crea una clase `PanelNuevaSimDescPaso7` que implemente la interfaz, sin modificar `PanelSimuladorDesc`.
- **Interfaz utils.ActualizableTextos:** permite agregar internacionalización a cualquier clase sin modificar su implementación interna.
 - *Ejemplo:* cualquier clase puede implementar `actualizarTextos(ResourceBundle)` para soporte multiidioma sin cambiar su lógica principal.
- **Clase gramatica.TablaPredictiva:** puede extenderse con nuevas estrategias de análisis sin modificar la implementación base.

- *Ejemplo:* TablaPredictivaPaso5 extiende TablaPredictiva agregando funcionalidad específica para el paso 5 del simulador.
- Clase **utils.TabManager**: puede manejar nuevos tipos de pestañas sin modificar su lógica central.
 - *Ejemplo:* método `getOrCreateTab(Class<?>tabType, ...)` acepta cualquier tipo de controlador, permitiendo extensión sin modificación.

9.4.3.3. Principio de Sustitución de Liskov (LSP)

El Principio de Sustitución de Liskov establece que los objetos de una clase derivada deben poder sustituirse por objetos de la clase base sin afectar la corrección del programa [liÑAÐžÐš1994program, martin2003agile].

- **Jerarquía gramatica.Simbolo:** las clases Terminal y NoTerminal pueden sustituirse por Simbolo en cualquier contexto.
 - *Ejemplo:* en `Gramatica.getTerminales()` y `Gramatica.getNoTerminales()`, ambos retornan `ObservableList<? extends Simbolo>`, permitiendo tratamiento uniforme.
- **Implementaciones de PanelNuevaSimDescPaso:** cualquier implementación de la interfaz puede sustituirse por otra sin afectar `PanelSimuladorDesc`.
 - *Ejemplo:* `PanelNuevaSimDescPaso1` puede reemplazarse por cualquier otra implementación sin modificar el código del simulador principal.
- **Clases de pasos de creación:** los diferentes `PanelCreacionGramaticaPaso*` pueden sustituirse entre sí manteniendo compatibilidad.
 - *Ejemplo:* cualquier paso del asistente puede reemplazarse por otro que implemente la misma interfaz sin afectar `PanelCreacionGramatica`.
- **Implementaciones de ActualizableTextos:** cualquier clase que implemente la interfaz puede utilizarse donde se espere internacionalización.
 - *Ejemplo:* tanto `Editor` como `PanelSimuladorDesc` implementan la interfaz y pueden tratarse uniformemente para cambios de idioma.

9.4.3.4. Principio de Segregación de Interfaces (ISP)

El Principio de Segregación de Interfaces establece que los clientes no deben verse forzados a depender de interfaces que no utilizan [martin2003agile, martin2018clean].

- **Interfaz utils.ActualizableTextos:** interfaz específica y minimalista solo para internacionalización.
 - *Ejemplo:* solo declara `actualizarTextos(ResourceBundle)`, evitando forzar a las clases a implementar métodos innecesarios de internacionalización compleja.

- **Interfaz simulador.PanelNuevaSimDescPaso:** interfaz minimalista solo para la estructura de pasos de simulación.
 - *Ejemplo:* solo declara `getRoot()`, evitando obligar a los pasos a implementar métodos de navegación o estado que maneja el coordinador.
- **Clase utils.TabManager:** proporciona métodos específicos separados en lugar de una interfaz monolítica.
 - *Ejemplo:* métodos como `isSimuladorContent()`, `isEditorContent()` permiten a los clientes usar solo la funcionalidad necesaria.
- **Ausencia de interfaces infladas:** las clases evitan depender de interfaces con métodos no utilizados.
 - *Ejemplo:* `Editor` no implementa interfaces con métodos de simulación que no necesita, manteniendo separación clara de responsabilidades.

9.4.3.5. Principio de Inversión de Dependencias (DIP)

El Principio de Inversión de Dependencias establece que los módulos de alto nivel no deben depender de módulos de bajo nivel, sino que ambos deben depender de abstracciones [martin2003agile, martin2018clean].

- **Dependencias de interfaces en lugar de clases concretas:** las clases dependen de interfaces, no de implementaciones concretas.
 - *Ejemplo:* `PanelSimuladorDesc` depende de `PanelNuevaSimDescPaso` (interfaz), no de implementaciones concretas como `PanelNuevaSimDescPaso1`.
- **Inyección de dependencias a través de constructores:** las clases reciben sus dependencias externas en lugar de crearlas internamente.
 - *Ejemplo:* `Editor(TabPane, MenuPrincipal, ResourceBundle)` recibe todas sus dependencias, facilitando testing y reutilización.
 - *Ejemplo:* `PanelSimuladorDesc(Gramatica, TabPane, ResourceBundle)` permite inyectar diferentes implementaciones.
- **Dependencia de abstracciones:** las clases de alto nivel dependen de interfaces, no de implementaciones concretas.
 - *Ejemplo:* `MenuPrincipal` depende de `utils.ActualizableTextos` (interfaz) para internacionalización, no de implementaciones específicas.
- **Fábricas y constructores parametrizados:** permiten crear diferentes configuraciones sin violar DIP.
 - *Ejemplo:* los múltiples constructores de `PanelSimuladorDesc` permiten diferentes configuraciones manteniendo la dependencia de interfaces.

9.4.4. Evaluación de Calidad Arquitectural

9.4.4.1. Métricas de Acoplamiento y Cohesión

Basado en el análisis de dependencias, la arquitectura presenta:

- **Cohesión Alta:** cada paquete tiene responsabilidades claramente definidas
 - Paquete `gramatica`: cohesión máxima en lógica gramatical
 - Paquete `utils`: cohesión máxima en utilidades transversales
 - Paquete `editor`: cohesión máxima en interfaces de edición
- **Acoplamiento Controlado:** uso estratégico de interfaces reduce acoplamiento
 - Acoplamiento a través de interfaces (`PanelNuevaSimDescPaso`, `ActualizableTextos`)
 - Inyección de dependencias reduce acoplamiento temporal
 - Patrones Observer y Strategy mantienen bajo acoplamiento
- **Complejidad Ciclomática Aceptable:** separación clara de responsabilidades
 - Clases con responsabilidades únicas tienen baja complejidad
 - Coordinadores (como `PanelSimuladorDesc`) manejan complejidad de integración

9.4.4.2. Puntos Fuertes de la Arquitectura

- **Mantenibilidad:** principios SOLID facilitan modificaciones localizadas
- **Testabilidad:** inyección de dependencias permite mock objects
- **Reutilización:** interfaces permiten uso polimórfico
- **Escalabilidad:** OCP permite agregar funcionalidad sin modificar código existente
- **Comprensibilidad:** separación clara de responsabilidades facilita comprensión

9.4.4.3. Áreas de Mejora Identificadas

- **Dependencias transversales:** `TabManager` y `ResourceBundle` podrían refactorizarse para reducir acoplamiento
- **Complejidad de constructores:** algunos constructores tienen muchos parámetros, podrían beneficiarse de Builder Pattern
- **Dependencias circulares:** algunas relaciones bidireccionales podrían eliminarse con eventos o mediadores

9.5. Conclusiones

Este capítulo ha presentado un análisis exhaustivo de la arquitectura de clases de SimAS 3.0, demostrando una implementación robusta de principios de diseño orientado a objetos y patrones de diseño reconocidos. La estructura modular y bien organizada facilita el mantenimiento, extensión y comprensión del sistema.

Los paquetes principales (bienvenida, editor, gramática, simulador, utils) han sido analizados en detalle, mostrando cómo cada componente cumple con responsabilidades específicas mientras mantiene una cohesión adecuada y un acoplamiento flexible. Los paquetes de soporte (centroayuda, vistas, resources) complementan esta arquitectura proporcionando recursos esenciales para la experiencia del usuario.

La implementación del patrón MVC, junto con otros patrones como Strategy, Observer, Factory Method y Registry, demuestra un diseño profesional que facilita la evolución del software y su adaptación a nuevos requisitos. La documentación detallada de atributos y métodos asegura que cualquier desarrollador pueda comprender y extender el sistema de manera efectiva.

En el siguiente capítulo se profundizará en la capa de presentación (paquete vistas), analizando en detalle los archivos FXML y el diseño de interfaces de usuario que complementan esta arquitectura de clases.

Capítulo 10

Diseño de la interfaz

10.1. Introducción

En este capítulo se abordará el diseño de cada uno de los componentes que forman la interfaz de la aplicación SimAS 3.0. Esta versión representa una evolución significativa respecto a las versiones anteriores, incorporando mejoras sustanciales tanto en funcionalidad como en experiencia de usuario.

La interfaz de SimAS 3.0 ha sido completamente rediseñada con un enfoque moderno y profesional, integrando las mejores prácticas de diseño de interfaces gráficas. Se ha implementado un sistema avanzado de pestañas que permite trabajar simultáneamente con múltiples gramáticas y simulaciones, mejorando significativamente la productividad del usuario. El manejo inteligente de ventanas secundarias facilita la comparación de resultados y el trabajo paralelo, características que no estaban disponibles en versiones anteriores.

Uno de los aspectos más destacados es la incorporación de acciones contextuales intuitivas, con iconos descriptivos que representan fielmente las operaciones que realizan. La interfaz presenta un aspecto formal y elegante, con una paleta de colores coherente y una tipografía legible que facilita la lectura prolongada. La navegación se ha optimizado mediante atajos de teclado estratégicos y un sistema de menús jerárquicos que agrupan las funcionalidades de manera lógica.

La aplicación ha sido estructurada siguiendo el patrón Modelo-Vista-Controlador (MVC), donde el módulo de vistas (implementado en clases como `MenuPrincipal.java`, `Editor.java`, `PanelSimulacion.java`, `TabManager.java` y `SecondaryWindow.java`) se encarga de la presentación e interacción con el usuario, proporcionando una separación clara entre la lógica de negocio y la interfaz gráfica.

Se ha prestado especial atención a la accesibilidad y usabilidad, incorporando características como soporte multiidioma completo, indicadores visuales claros para el estado de las operaciones, y validación en tiempo real de las entradas del usuario. La interfaz se adapta automáticamente a diferentes resoluciones de pantalla, manteniendo una experiencia consistente en diversos entornos de trabajo.

Puesto que muchos componentes son similares, se mostrará únicamente un com-

ponente de cada tipo en este capítulo técnico. Una descripción más detallada de cada componente de la interfaz, orientada al usuario final, se puede consultar en el *Manual de Usuario*.

A continuación se muestran los elementos gráficos de la interfaz final de la aplicación con una explicación detallada de cada una de las partes que los componen, destacando las innovaciones técnicas implementadas.

10.2. Menú principal

El menú principal contiene las acciones a realizar en una ventana agrupadas por categorías. En la figura 10.1, se muestra un ejemplo de una barra de menús para el Editor de gramáticas del programa SimAS.

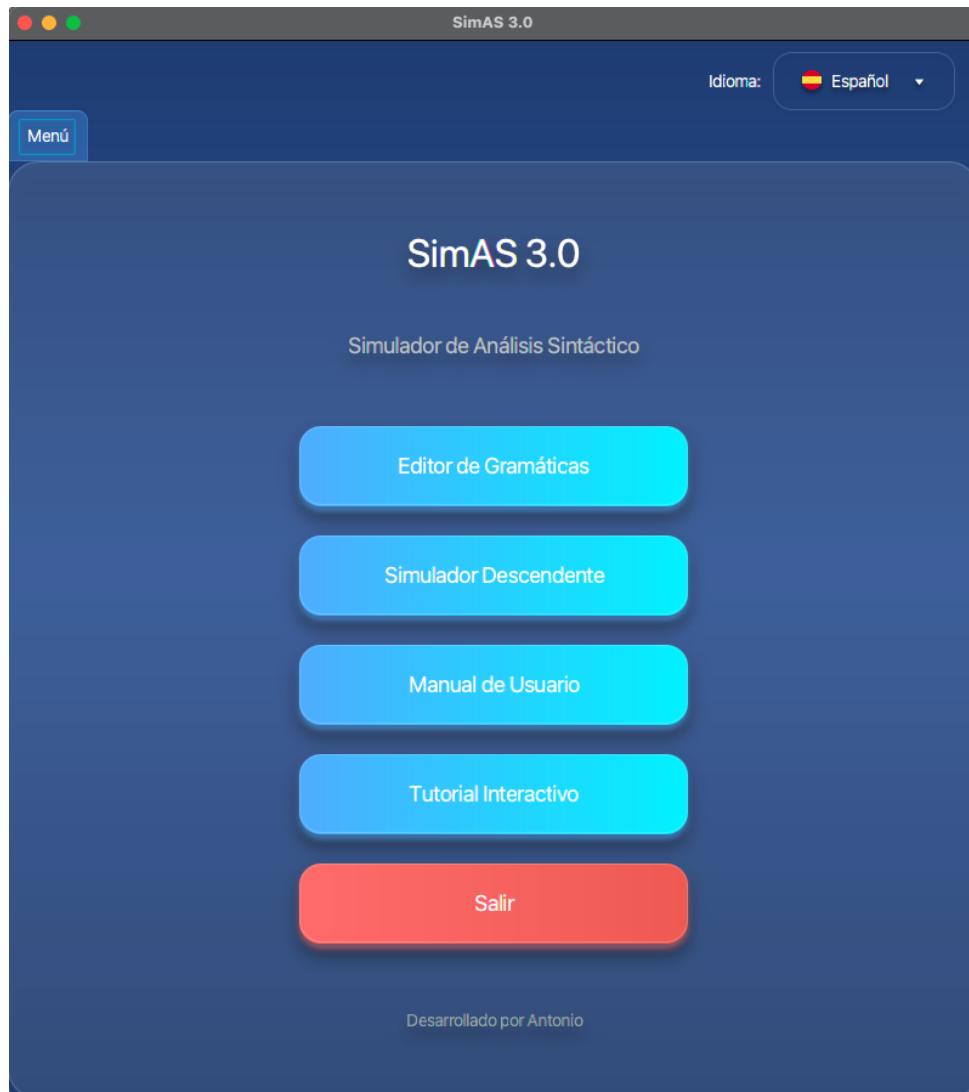


Figura 10.1: Menú principal de SimAS.

La barra de menús contiene cada una de las acciones que se puede realizar en la aplicación SimAS agrupadas según la funcionalidad de la acción. Además, se han dado

nombres muy descriptivos a cada elemento, para así procurar que la interfaz sea lo más intuitiva posible.

10.3. Barras de herramientas

Las barras de herramientas contienen los accesos directos a las acciones que se pueden realizar en una ventana, que están representadas gráficamente. En la figura 10.2, se muestra un ejemplo de una barra de herramientas para el Editor de gramáticas del programa SimAS.

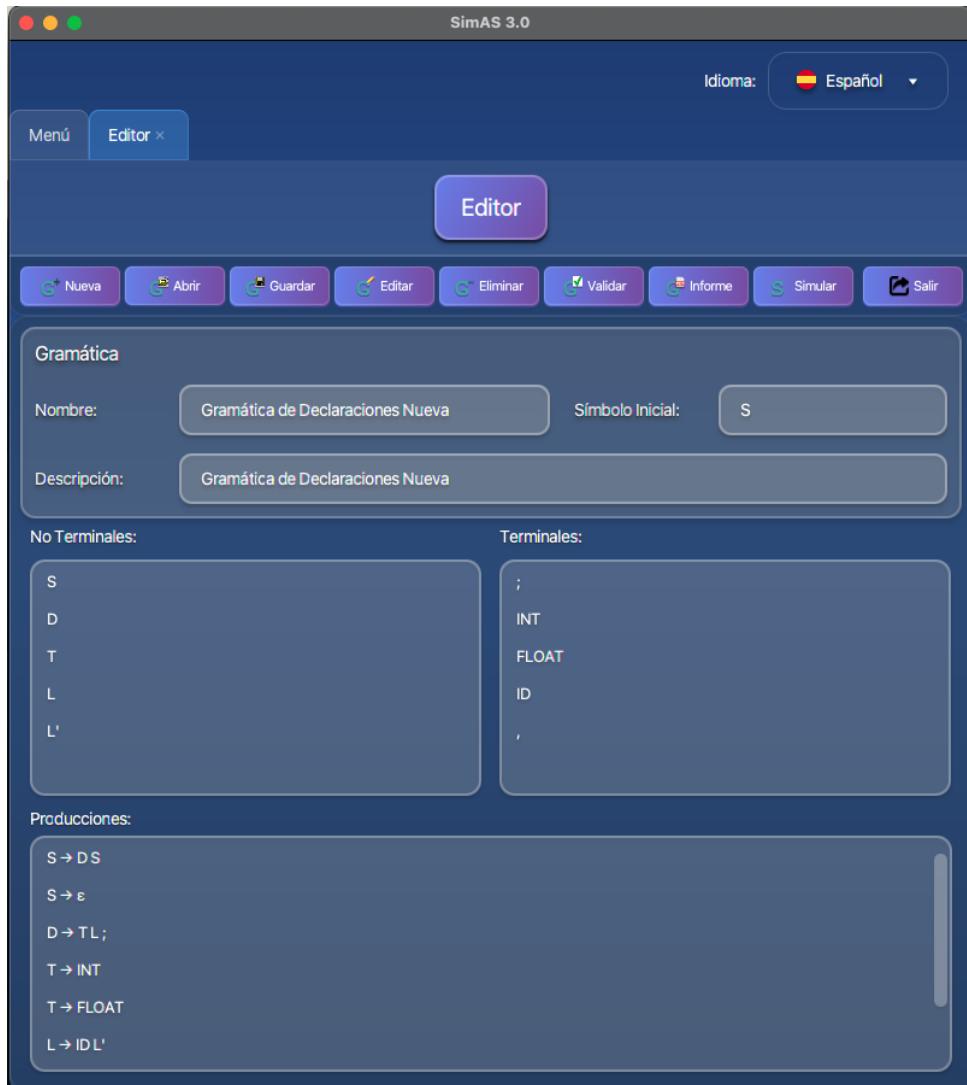


Figura 10.2: Barra de herramientas de SimAS.

Se ha procurado que cada botón sea descriptivo y represente fielmente la operación que realiza. Por esto, se ha tomado como convenio usar una forma base en los iconos según el objeto al que involucran en la operación, por ejemplo: las operaciones de la gramática está representada con la letra G y las operaciones de las simulaciones con la letra S .

A esta forma base, se le ha superpuesto una forma que representa de forma gráfica la acción a llevar a cabo. Así, se ha superpuesto la forma base del objeto con la acción a

realizar, siempre en la esquina superior derecha de cada uno de los iconos de la barra de herramientas. Todo esto permite que la interfaz sea muy intuitiva y permita al usuario trabajar con eficacia y con el mínimo de confusiones posibles.

10.4. Ventana del editor

La ventana del editor permite crear, editar y validar las gramáticas de contexto libre para su posterior simulación. En la figura 10.3, se muestra un ejemplo de esta ventana con una gramática creada y validada.

La ventana está compuesta de los siguientes elementos:

1. **Barra de menús**: barra de menús de la ventana.
2. **Barra de herramientas**: barra de herramientas de la ventana.
3. **Panel de edición**: panel que almacena y representa las gramáticas.

En las secciones anteriores ya se vio con detalle la barra de menús y la barra de herramientas; a continuación se va a llevar a cabo la descripción de cada una de las partes que compone el panel de edición.

10.4.1. Panel de edición

El panel de edición permite visualizar cada uno de los componentes de las gramáticas de contexto libre así como su estado: validada o no validada. En la figura 10.4, se muestra un ejemplo de este panel.

El panel está compuesto de los siguientes elementos:

1. **Nombre de la gramática**: nombre descriptivo de la gramática.
2. **Estado de la gramática**: muestra si la gramática ha sido validada o no.
3. **Descripción**: descripción de la gramática.
4. **Símbolos de la gramática**: se muestran los símbolos terminales y los no terminales.
5. **Símbolo inicial**: muestra cuál de los símbolos no terminales es el símbolo inicial de la gramática.
6. **Producciones de la gramática**: visualización de las reglas de producción de la gramática.



Figura 10.3: Ventana del editor.

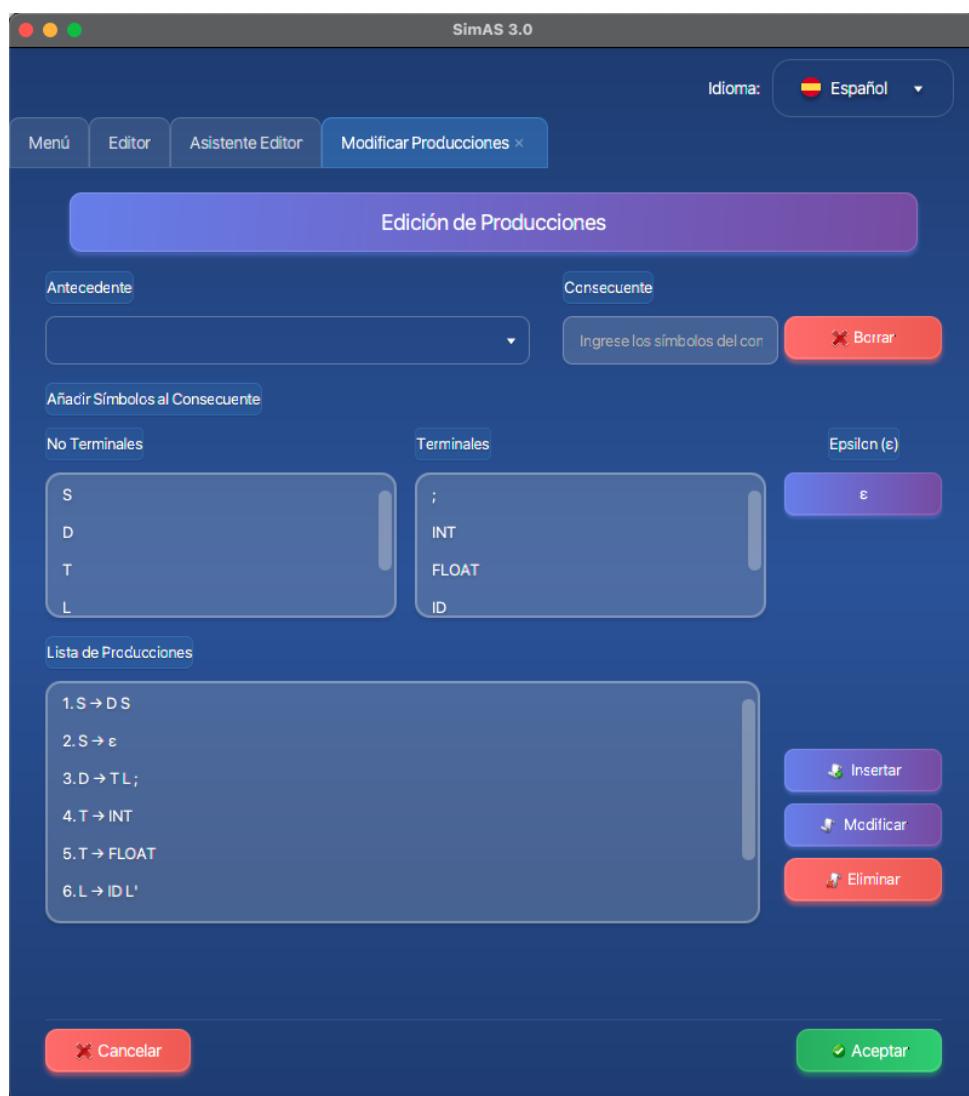


Figura 10.4: Panel de edición.

10.5. Ventana del simulador

La ventana del simulador permite ejecutar la simulación de un analizador descendente o ascendente utilizando una gramática. En la figura 10.5, se muestra un ejemplo de esta ventana.

La ventana está compuesta de los siguientes elementos:

1. **Barra de menús**: barra de menús de la ventana.
2. **Barra de herramientas**: barra de herramientas de la ventana.
3. **Panel de simulación**: panel que muestra los datos de la gramática a simular.

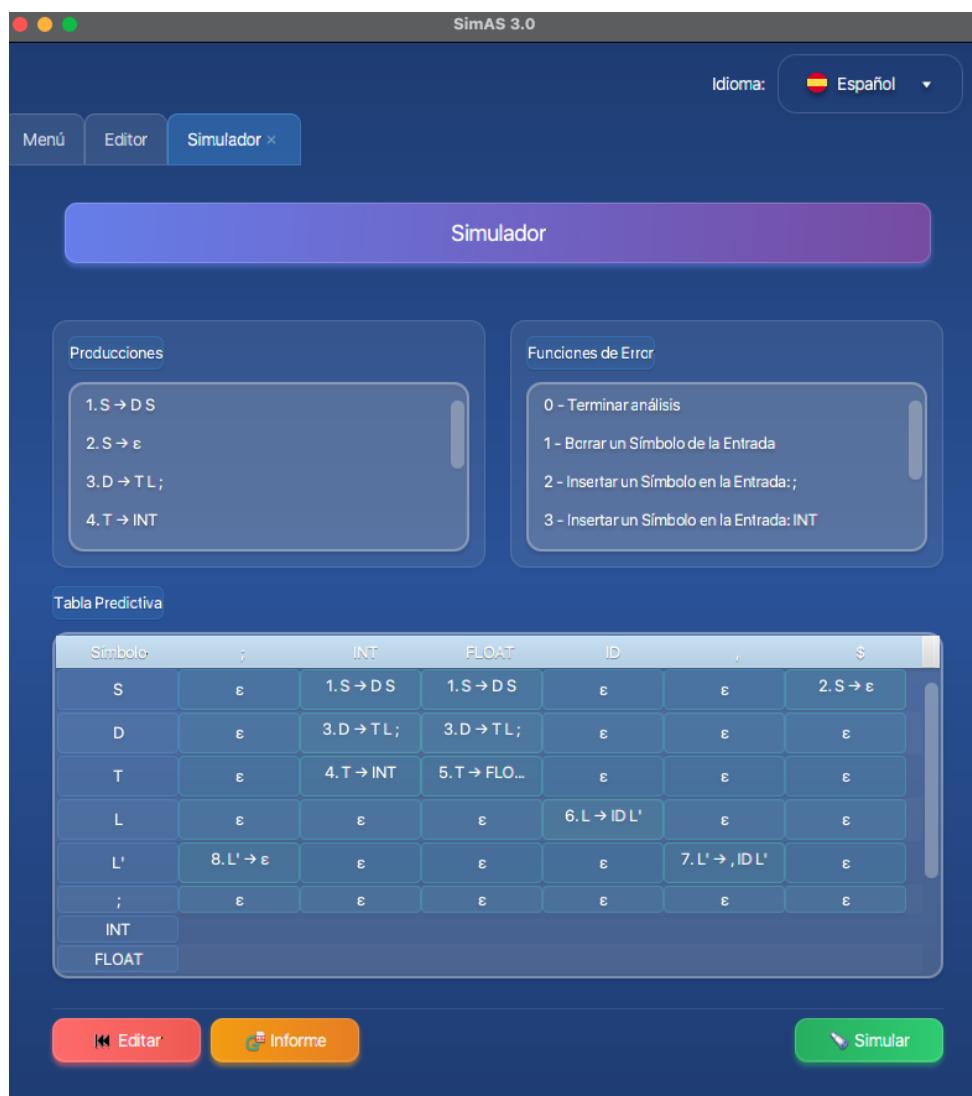


Figura 10.5: Ventana de simulación.

En las secciones anteriores ya se vio con detalle la barra de menús y la barra de herramientas; a continuación, se llevará a cabo la descripción de cada una de las partes que compone el panel de simulación.

10.5.1. Panel de simulación

El panel de simulación permite visualizar las simulaciones de los analizadores sintácticos. En la figura 10.6, se muestra un ejemplo de este panel.

El panel está compuesto de los siguientes elementos:

1. **Tipo de análisis:** muestra el tipo de simulación que se va a llevar a cabo, *descendente* o *ascendente*. En el caso de ser la simulación del análisis ascendente, también se detalla el tipo: *SLR*, *LR-canónico* o *LALR*.
2. **Producciones de la gramática:** visualización gráfica de las producciones de la gramática.
3. **Funciones de error:** lista de funciones de error de la gramática.
4. **Tabla predictiva o tabla LR:** muestra la tabla predictiva o la tabla LR, dependiendo del método seleccionado.

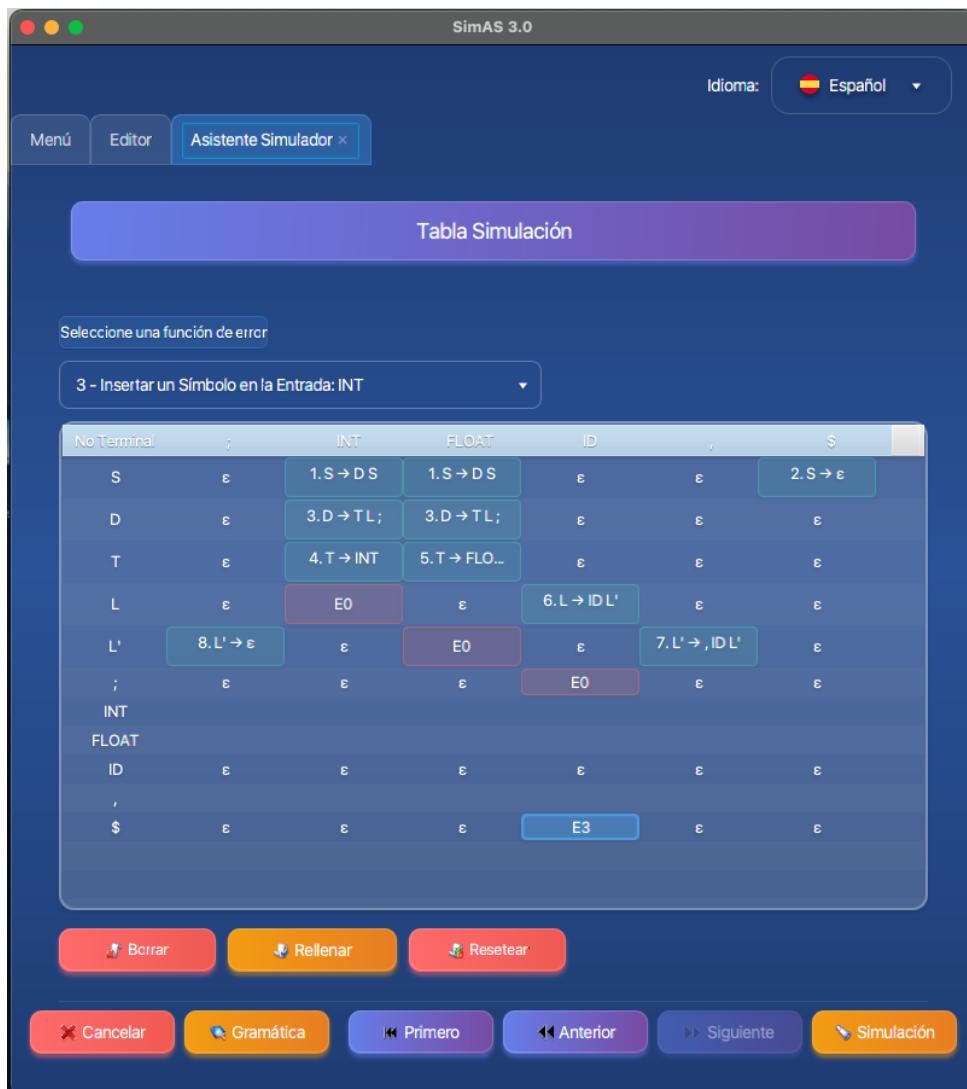


Figura 10.6: Panel de simulación.

10.5.2. Ventana de Gramática Original

Visualización de la Gramática Original (Novedad): a medida que se generan los diferentes conjuntos, tablas y funciones de error, se le permite al usuario visualizar la gramática original para así poder realizar las comparaciones que deseé si dicha gramática se ha transformado en otra porque se ha eliminado la recursividad por la izquierda o se ha factorizado por la izquierda. La ventana será un panel simple donde se muestre la gramática original de la misma forma que se muestra la gramática modificada (Figura 10.7).

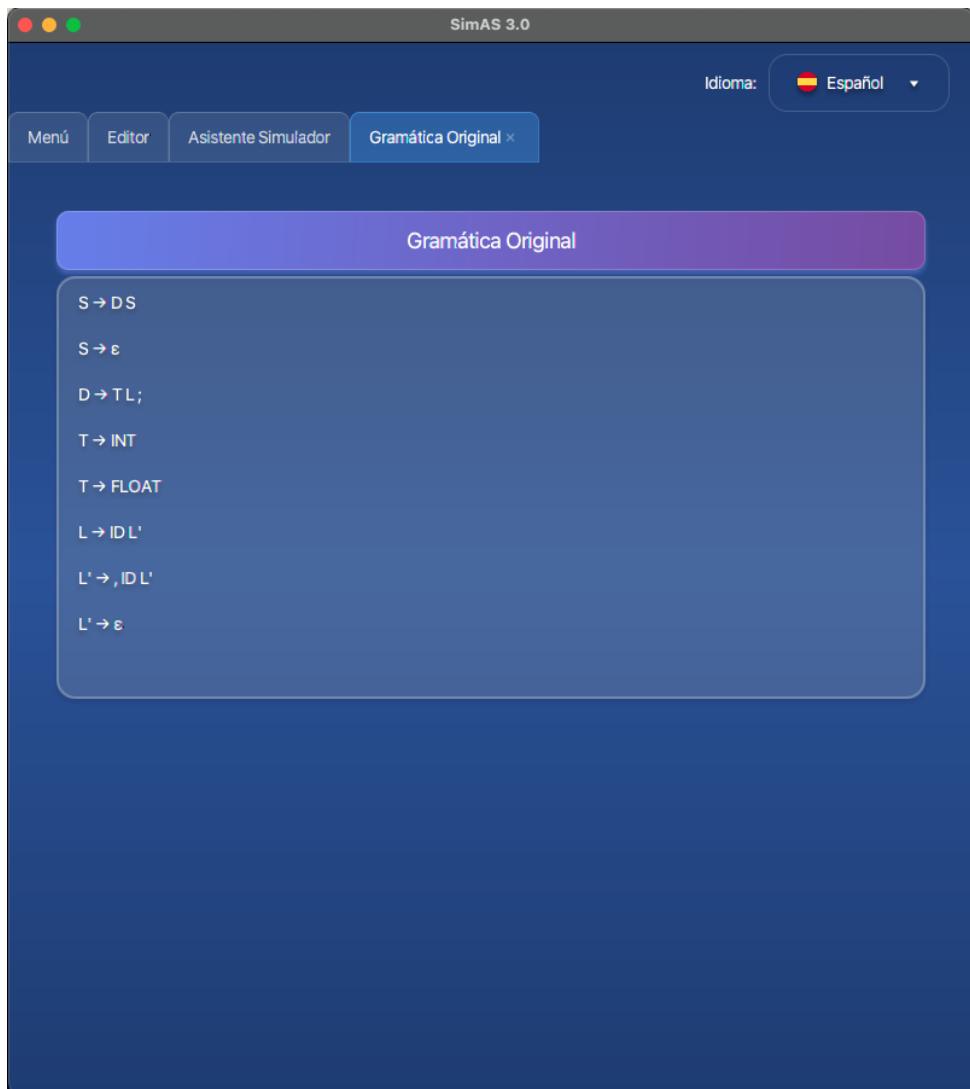


Figura 10.7: Gramática Original.

10.6. Ventana de simulación

La ventana de simulación se encarga de ejecutar las simulaciones de una cadena de entrada. En la figura 10.8, se muestra un ejemplo de esta ventana.

La ventana está compuesta de los siguientes elementos:

1. **Tipo de análisis:** muestra el tipo de simulación que se va a llevar a cabo, *descendente* o *ascendente*. En el caso de ser la simulación del análisis ascendente, también se detalla el tipo: *SLR*, *LR-canónico* o *LALR*.
2. **Cadena de entrada:** permite introducir los símbolos terminales que forman parte de la cadena de entrada que se va a simular.
3. **Botones de Avance y Retroceso:** la simulación se podrá realizar paso a paso o de forma completa, pudiendo avanzar y retroceder en cada paso.
4. **Tabla de análisis:** representa la tabla de resultados del análisis.
5. **Generar Árbol Sintáctico** (Novedad): permite abrir una ventana en la que se genere el árbol sintáctico correspondiente.
6. **Generar Derivación** (Novedad): permite abrir una ventana en la que se genere la derivación de la simulación correspondiente.
7. **Informe de la Simulación:** permite al usuario generar un informe en pdf de la simulación.

10.7. Ventana de Visualización del Árbol Sintáctico

El árbol sintáctico se podrá ver en una ventana la cual se irá repintando a medida que se continue con la simulación (Figura 10.9).

10.8. Ventana de Derivación Sintáctica

Además del árbol, es posible generar la derivación de la simulación actual en una ventana adicional tal y como se muestra en la imagen 10.10:

10.9. Ventana de asistente

En SimAS se han creado dos tipos de asistentes: el *asistente de creación de una gramática* y el *asistente de creación de una simulación*. En la figura 10.11, se muestra un ejemplo de esta ventana; en concreto, el primer paso del asistente de creación de una gramática.

La ventana está compuesta de los siguientes elementos:

1. **Título:** contiene un título descriptivo de la ventana en la que se sitúe el asistente.
2. **Contenido:** información que se muestra en el panel en función del paso en el que se está.
3. **Botonera:** permite controlar la evolución del asistente (cancelar, ir al paso anterior, ir al primer paso, etcétera).

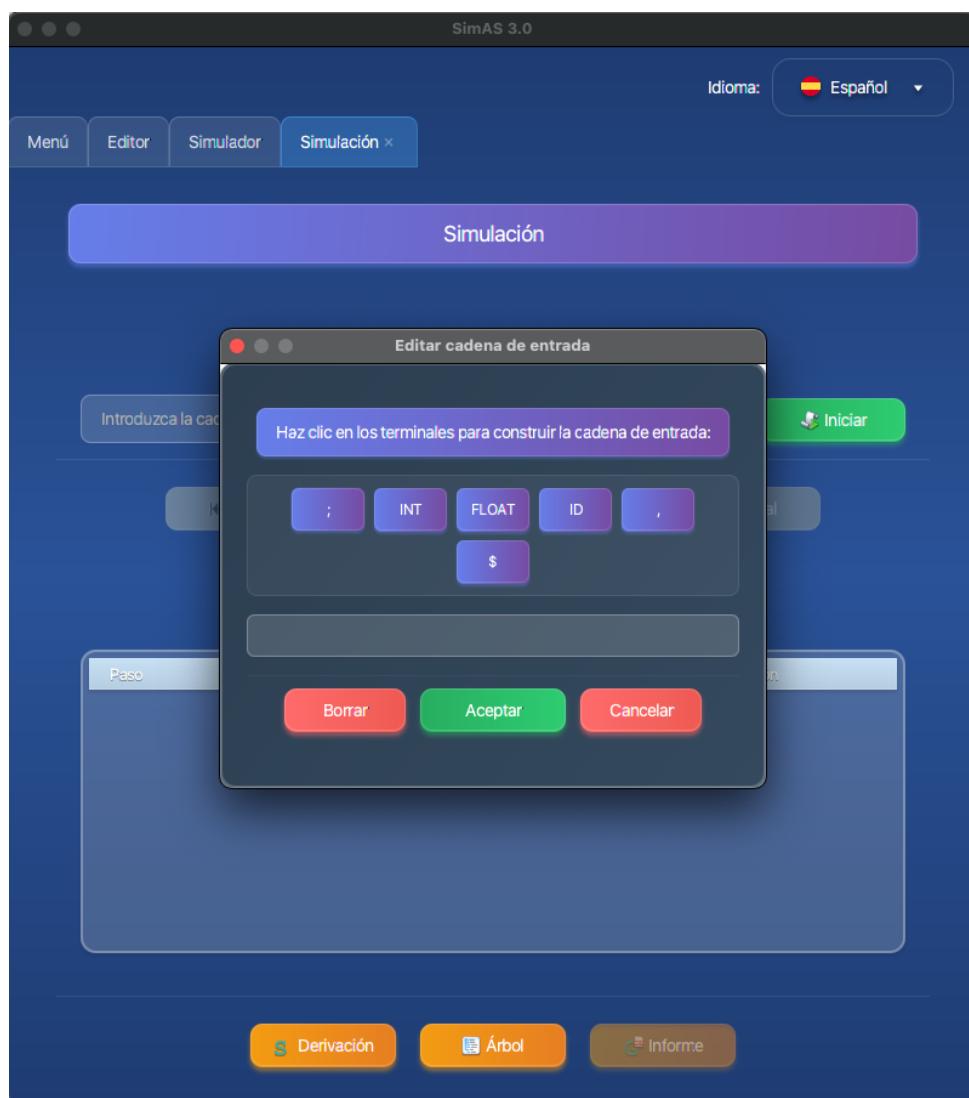


Figura 10.8: Ventana de simulación.

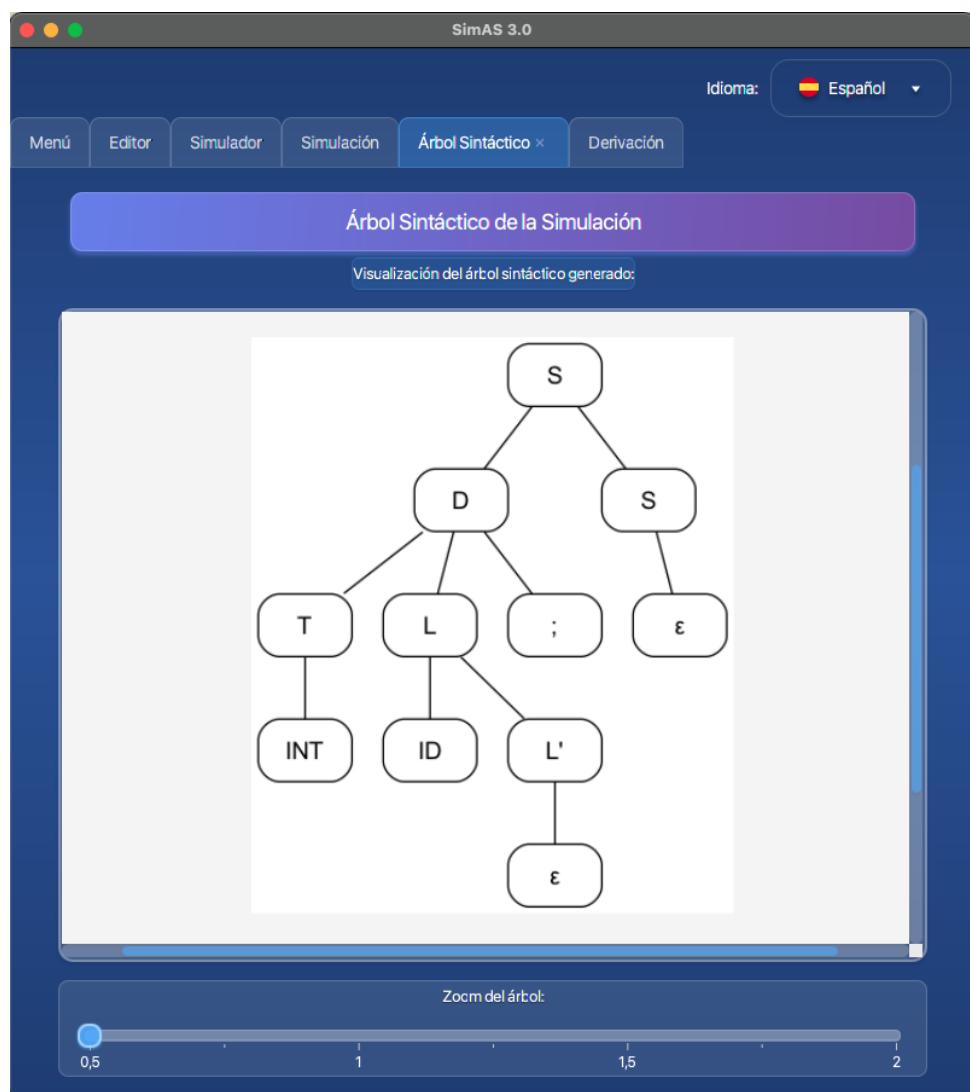


Figura 10.9: Ventana de simulación.

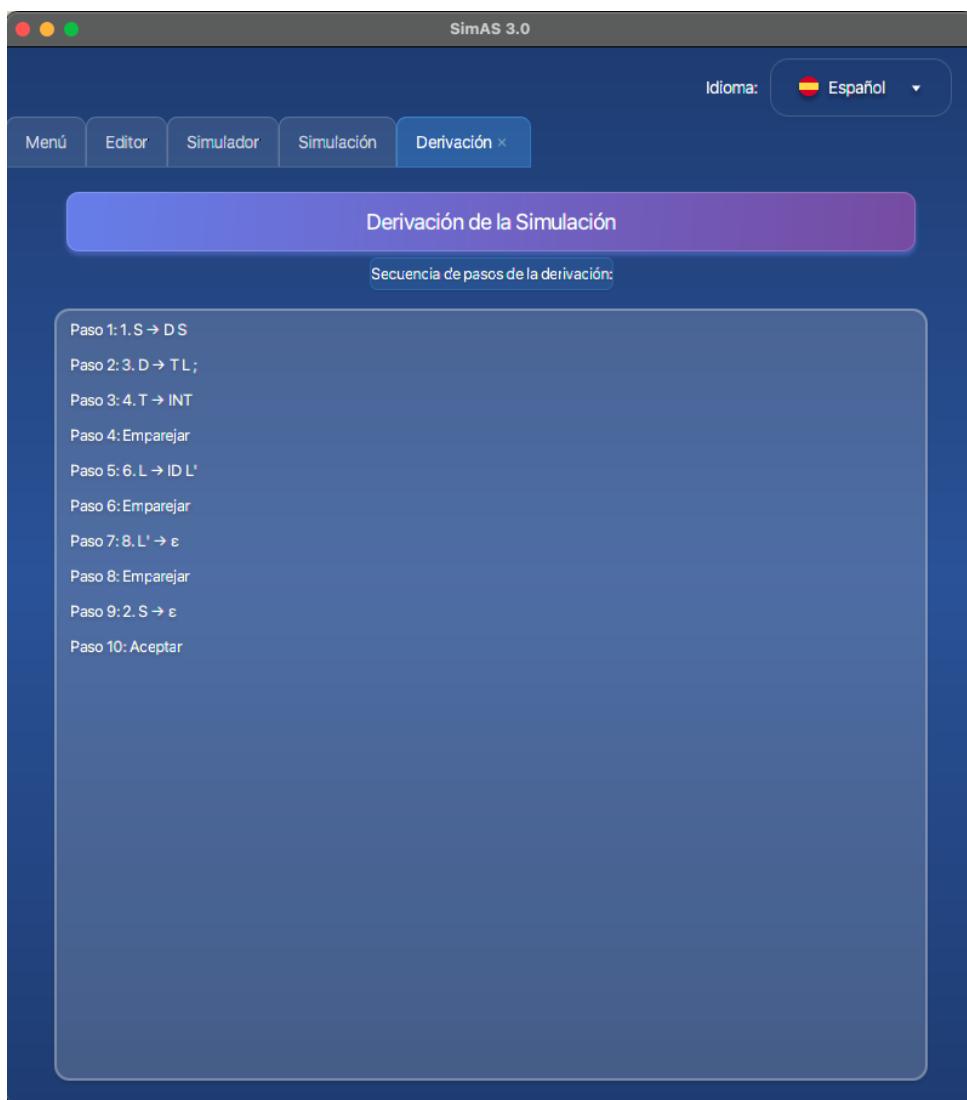


Figura 10.10: Ventana de simulación.

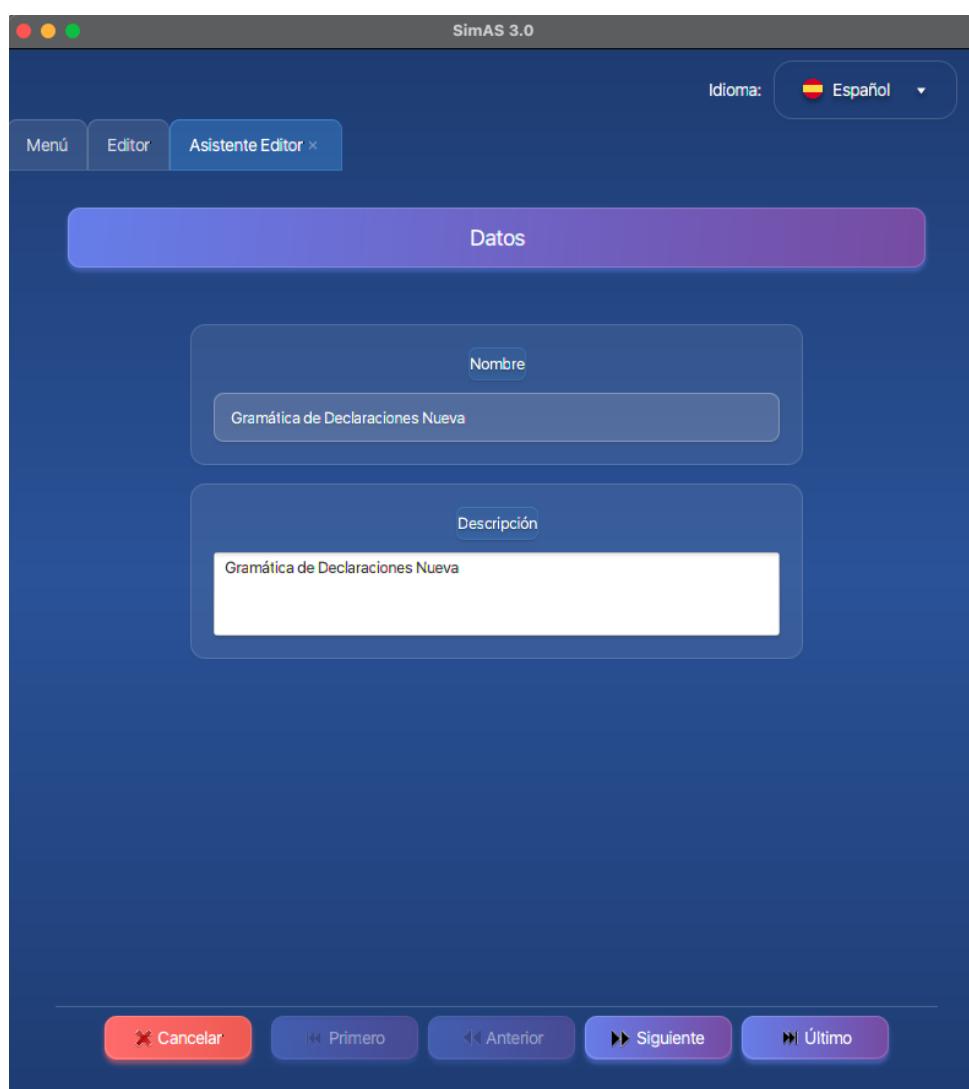


Figura 10.11: Ventana de asistente.

10.10. Ventanas de edición

Las ventanas de edición representan a todas las ventanas que permiten realizar una edición de un objeto (nombre, descripción, símbolos, producciones, funciones de error, etcétera). En la figura 10.12, se muestra un ejemplo de esta ventana; en concreto, la ventana de edición de los símbolos de la gramática.

La ventana está compuesta de los siguientes elementos:

1. **Título:** contiene un título descriptivo de la ventana.
2. **Contenido:** información a editar que se muestra en la ventana.
3. **Botonera:** permite cancelar o aceptar la operación. El botón de cancelar siempre estará situado a la izquierda, mientras que el de aceptar lo estará en la derecha.

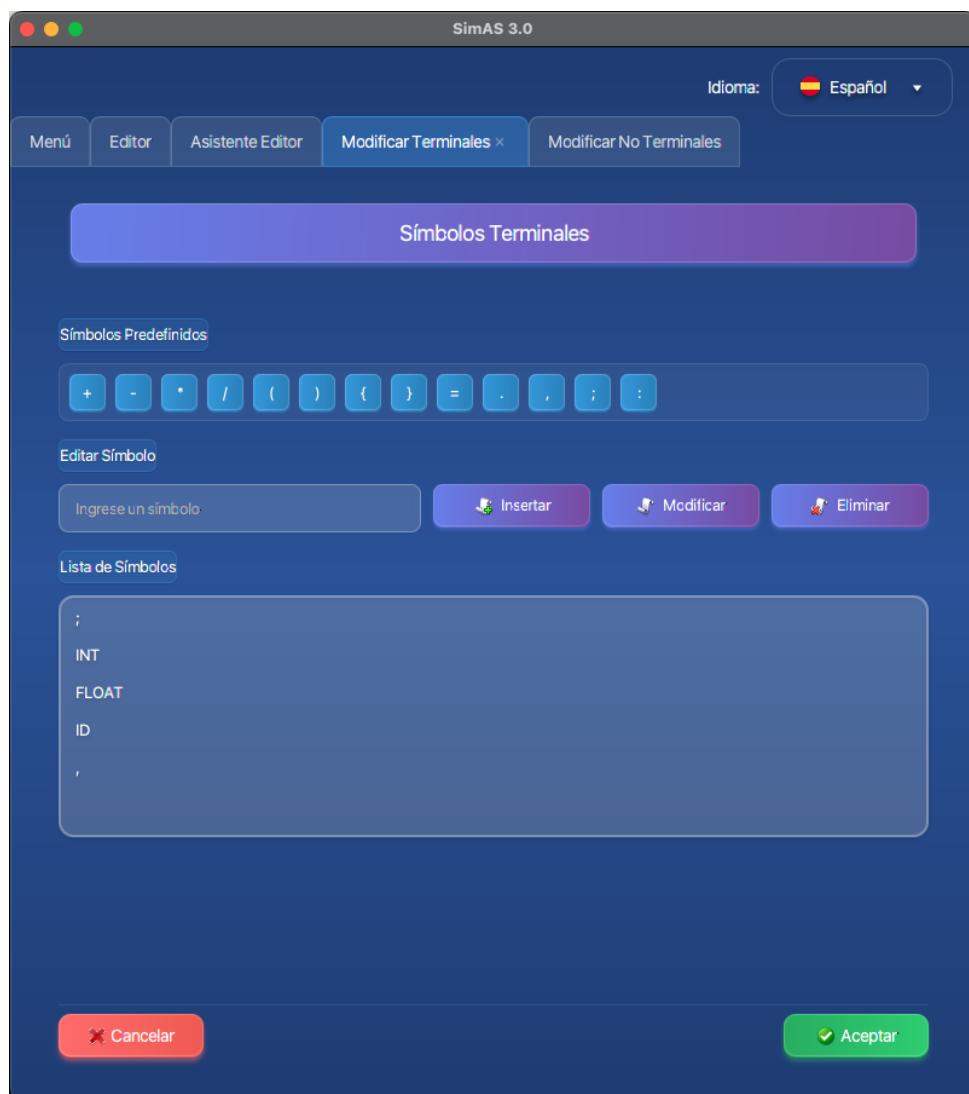


Figura 10.12: Ventana de edición.

Parte IV

Pruebas

Capítulo 11

Pruebas

11.1. Introducción

11.2. Pruebas del módulo Editor

11.2.1. Almacenamiento y recuperación de gramáticas

11.2.2. Validación de gramáticas

11.3. Pruebas del módulo Simulador

Bibliografía

Bibliografía

- [1] A. V. Aho, M. S. Lam, R. Sethi y J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. 2nd. Addison-Wesley Professional, 2008. ISBN: 978-0321486813.
- [2] Apache Software Foundation. *NetBeans IDE*. <https://netbeans.apache.org/>. Versión 7.3.1. 2013.
- [3] J. Bloch. *Effective Java*. Addison-Wesley Professional, 2008. ISBN: 978-0321356680.
- [4] Canonical Ltd. *Ubuntu Linux*. <https://ubuntu.com/>. Versión 12.10. 2012.
- [5] J. A. F. Díaz. *SimAS 2.0: Simulador de Análisis Sintáctico*. Trabajo de Fin de Grado, Universidad de Córdoba. 2023.
- [6] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002. ISBN: 978-0321127420.
- [7] E. Freeman y E. Robson. *Head First Design Patterns*. O'Reilly Media, 2004. ISBN: 978-0596007126.
- [8] E. Gamma, R. Helm, R. Johnson y J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. ISBN: 978-0201633610.
- [9] GitHub, Inc. *GitHub*. <https://github.com/>. Accedido: 2024. 2024.
- [10] Gluon. *JavaFX Scene Builder*. <https://gluonhq.com/products/scene-builder/>. Accedido: 2024. 2024.
- [11] Graphviz Team. *Graphviz: Graph Visualization Software*. <https://graphviz.org/>. Accedido: 2024. 2024.
- [12] iText Group. *iText PDF Library*. <https://itextpdf.com/>. Versión 5.5.0. 2015.
- [13] JetBrains. *IntelliJ IDEA*. <https://www.jetbrains.com/idea/>. Accedido: 2024. 2024.
- [14] JGraph Ltd. *Draw.io (Diagrams.net)*. <https://www.diagrams.net/>. Accedido: 2024. 2024.
- [15] L. Lamport. *LaTeX: A Document Preparation System*. 2nd. Addison-Wesley Professional, 1994. ISBN: 978-0201529838.
- [16] Microsoft. *Layered Architecture*. <https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/layered-architecture>. Accedido: 2024. 2023.
- [17] Microsoft Corporation. *Microsoft Windows 7*. <https://www.microsoft.com/windows/>. 2009.

BIBLIOGRAFÍA

- [18] Oracle Corporation. *Factory Method Pattern*. <https://www.oracle.com/technetwork/java/factory-method-139282.html>. Accedido: 2024. 2024.
- [19] Oracle Corporation. *Java Platform, Standard Edition*. <https://www.oracle.com/java/>. Accedido: 2024. 2024.
- [20] Oracle Corporation. *Java Swing*. Accedido: 2024. 2024.
- [21] Oracle Corporation. *JavaFX*. <https://openjfx.io/>. Accedido: 2024. 2024.
- [22] Oracle Corporation. *Model-View-Controller Pattern*. <https://www.oracle.com/technetwork/java/mvc-140477.html>. Accedido: 2024. 2024.
- [23] Oracle Corporation. *Observer Pattern*. <https://www.oracle.com/technetwork/java/observer-137868.html>. Accedido: 2024. 2024.
- [24] Oracle Corporation. *Singleton Pattern*. <https://www.oracle.com/technetwork/java/singleton-137897.html>. Accedido: 2024. 2024.
- [25] Oracle Corporation. *Strategy Pattern*. <https://www.oracle.com/technetwork/java/strategy-137857.html>. Accedido: 2024. 2024.
- [26] Overleaf Ltd. *Overleaf: LaTeX Editor*. <https://www.overleaf.com/>. Accedido: 2024. 2024.
- [27] V. G. Pérez. *SimAS 1.0: Simulador de Análisis Sintáctico*. Proyecto de Fin de Carrera, Universidad de Córdoba. 2015.
- [28] A. Shalloway y J. R. Trott. *Design Patterns Explained: A New Perspective on Object-Oriented Design*. Addison-Wesley Professional, 2001. ISBN: 978-0201715941.
- [29] The Git Project. *Git: Distributed Version Control System*. <https://git-scm.com/>. Accedido: 2024. 2024.
- [30] World Wide Web Consortium. *Extensible Markup Language (XML) 1.0*. <https://www.w3.org/TR/xml/>. Accedido: 2024. 2008.