

Universidad Nacional de Educación a Distancia
Escuela Técnica Superior de Ingeniería Informática
Máster Universitario en Ingeniería Informática



Editor gráfico de modelos económicos descritos en lenguaje Modelica

Trabajo Fin de Máster

Autor: Diego Puig Berenguer

Tutor: Dña. Carla Martín Villalba

Tutor: D. Alfonso Urquía Moraleda

Departamento: Departamento de Informática y Automática

Convocatoria: Septiembre 2026

Resumen

El presente Trabajo de Fin de Máster describe el diseño, implementación y validación de un **editor gráfico especializado para el modelado de sistemas económicos** utilizando el lenguaje Modelica. La herramienta facilita la construcción visual e intuitiva de modelos macroeconómicos clásicos: el **modelo IS-LM** (equilibrio en los mercados de bienes y dinero) y el **modelo de crecimiento neoclásico de Solow** (Solow, 1956) (acumulación de capital y convergencia económica a largo plazo), e integra datos reales del Banco Mundial para parametrizar las simulaciones con series históricas.

El editor se concibe con una finalidad fundamentalmente **pedagógica**: facilitar la comprensión visual e interactiva de la teoría macroeconómica clásica y de los paradigmas económicos alternativos, sirviendo como herramienta docente para estudiantes y profesorado de economía. Esta orientación pedagógica no excluye, sin embargo, su utilidad como herramienta de **exploración y análisis científico** de los propios modelos: la validación numérica frente a soluciones analíticas (Cap. 5) demuestra que los resultados producidos son acordes a la teoría económica.

El sistema implementa una **arquitectura Modelo-Vista-Controlador** en Java SE 17 con interfaz Swing. Sus componentes principales son: una **biblioteca de trece componentes** reutilizables (IS-LM y Solow), un **editor visual *drag-and-drop***, un **validador de conexiones**, un **exportador** que genera código Modelica estándar preservando el layout visual, un **parseador** bidireccional para importar y editar archivos `.mo` existentes, y un **módulo de datos reales** mediante la API REST del Banco Mundial.

La validación se realizó compilando y simulando los modelos generados en OpenModelica, verificando que los valores simulados coinciden con los valores analíticos esperados para distintos escenarios de política fiscal y monetaria. La arquitectura modular permite además ampliar el sistema hacia **modelos económicos alternativos**: se implementa y valida una variante del modelo de Solow con **economía circular**, que incorpora la reutilización del capital depreciado mediante una tasa de circularidad y eleva el nivel de capital de estado estacionario. Esta línea demuestra que el enfoque admite paradigmas postcrecimiento y se proyecta hacia marcos más amplios de sostenibilidad, como el modelo **Doughnut** de Kate Raworth y los límites planetarios.

Palabras clave: Modelica, editor gráfico, modelo IS-LM, modelo de Solow, OpenModelica, Java Swing, simulación económica, Economía Circular, modelo Doughnut.

Abstract

This Master’s Thesis presents the design, implementation and validation of a **graphical editor for economic modelling** using the Modelica language. The tool provides an intuitive visual environment for building macroeconomic models: the **IS-LM model** (simultaneous equilibrium in goods and money markets) and the **Solow neoclassical growth model** (capital accumulation and long-run economic convergence), and integrates real-world data from the World Bank to parameterise simulations with historical time series.

The editor is conceived with a primarily **pedagogical** purpose: to facilitate a visual and interactive understanding of classical macroeconomic theory and alternative economic paradigms, serving as a teaching tool for economics students and instructors. This pedagogical orientation does not, however, exclude its value as a tool for **scientific exploration and analysis** of the models themselves: the numerical validation against analytical solutions (Ch. 5) shows that the results obtained are consistent with economic theory.

The system is built on a **Model-View-Controller architecture** in Java SE 17 with a Swing GUI. Key components include: a reusable **library of thirteen economic components**, a **drag-and-drop visual canvas**, a **connection validator**, an **exporter** that generates standard Modelica code preserving the visual layout, a **bidirectional parser** for importing and iteratively editing existing `.mo` files, and a **real-data module** using the REST API from the World Bank.

Validation was carried out by compiling and simulating the generated models in OpenModelica, confirming that simulated values match the expected analytical results across multiple fiscal and monetary policy scenarios. The modular architecture also supports extensions to alternative economic models: a **Circular Economy** variant of the Solow model is implemented and validated, incorporating the reuse of depreciated capital through a circularity rate and raising the steady-state capital level. This demonstrates that the approach accommodates post-growth paradigms and opens a path towards broader sustainability frameworks such as Kate Raworth’s **Doughnut model** and planetary boundaries.

Keywords: Modelica, graphical editor, IS-LM model, Solow model, OpenModelica, Java Swing, economic simulation, Circular Economy, Doughnut model.

Agradecimientos

A mis tutores, Carla Martín Villalba y Alfonso Urquía Moraleda, por su orientación, sus consejos y la dedicación que han prestado a la dirección de este trabajo durante todo el proceso.

También dar las gracias a los profesores del Grado y Master de Informática de la Uned, compañeros de estudio, en general todos los que con su trabajo, ilusión y conocimientos han iluminado el camino. También al Centro Asociado de Denia que ha ayudado en todo este tiempo de estudio.

A mis padres, familia y amigos, por su paciencia, comprensión y aliento constante a lo largo de estos meses.

Listado de Siglas y Acrónimos

API	Interfaz de Programación de Aplicaciones (<i>Application Programming Interface</i>)
BCI	Índice de Bien Común (<i>Balance of the Common Good Index</i>)
CES	Elasticidad de Sustitución Constante (<i>Constant Elasticity of Substitution</i>); función de producción que generaliza la Cobb-Douglas permitiendo elasticidades variables entre capital y trabajo.
CSV	Valores Separados por Comas (<i>Comma-Separated Values</i>)
DAE	Ecuaciones Diferenciales Algebraicas (<i>Differential Algebraic Equations</i>)
DIP	Principio de Inversión de Dependencias (<i>Dependency Inversion Principle</i>)
EDT	Hilo de Despacho de Eventos (<i>Event Dispatch Thread</i>); hilo único de Swing responsable de actualizar la interfaz gráfica; todas las actualizaciones del canvas se delegan a él mediante <code>SwingUtilities.invokeLater</code> .
DSGE	Equilibrio General Dinámico Estocástico (<i>Dynamic Stochastic General Equilibrium</i>)
EC	Economía Circular
FED	Reserva Federal de los Estados Unidos (<i>Federal Reserve</i>)
FMI	Fondo Monetario Internacional / <i>Functional Mock-up Interface</i> (estándar de intercambio de modelos de simulación; el significado se indica en cada contexto)
FMU	Unidad de Maqueta Funcional (<i>Functional Mock-up Unit</i>)
GUI	Interfaz Gráfica de Usuario (<i>Graphical User Interface</i>)
HTML	Lenguaje de Marcado de Hipertexto (<i>HyperText Markup Language</i>)
HTTP	Protocolo de Transferencia de Hipertexto (<i>HyperText Transfer Protocol</i>)
IDE	Entorno de Desarrollo Integrado (<i>Integrated Development Environment</i>)

INE	Instituto Nacional de Estadística; organismo español productor de estadísticas oficiales. En este trabajo se optó por la API del Banco Mundial como fuente alternativa de acceso abierto.
IS-LM	<i>Investment-Saving / Liquidity preference-Money supply</i> (Inversión-Ahorro / Preferencia de Liquidez-Cantidad de Dinero)
ISP	Principio de Segregación de Interfaces (<i>Interface Segregation Principle</i>)
ISO	Organización Internacional de Normalización (<i>International Organization for Standardization</i>)
JDK	Kit de Desarrollo de Java (<i>Java Development Kit</i>)
JSON	Notación de Objetos JavaScript (<i>JavaScript Object Notation</i>)
JVM	Máquina Virtual de Java (<i>Java Virtual Machine</i>)
LSP	Principio de Sustitución de Liskov (<i>Liskov Substitution Principle</i>)
MATLAB	<i>Matrix Laboratory</i> ; entorno de cómputo numérico y lenguaje de programación de MathWorks, ampliamente utilizado en economía para la simulación de modelos dinámicos.
MOSES	Librería de Modelos Socioeconómicos Sostenibles (<i>Modeling of Sustainable Economic Systems</i>)
MSL	<i>Modelica Standard Library</i> (Modelica Association, 2026)
MVC	Modelo-Vista-Controlador (<i>Model-View-Controller</i>)
OCP	Principio Abierto/Cerrado (<i>Open/Closed Principle</i>)
OCDE	Organización para la Cooperación y el Desarrollo Económicos
ODE	Ecuaciones Diferenciales Ordinarias (<i>Ordinary Differential Equations</i>)
ONU	Organización de las Naciones Unidas
PIB	Producto Interior Bruto
ρ	Tasa de circularidad ($\rho \in [0,1]$); fracción del capital depreciado que se reintegra al ciclo productivo en el componente <code>Solow_Circularidad</code> (Economía Circular). Con $\rho = 0$ coincide con el modelo de Solow estándar; con $\rho = 1$ toda la depreciación se recupera.
REST	<i>Representational State Transfer</i>
SOLID	Acrónimo de los cinco principios de diseño orientado a objetos: SRP, OCP, LSP, ISP y DIP

SRP	Principio de Responsabilidad Única (<i>Single Responsibility Principle</i>)
SSP	Estructura y Parametrización del Sistema (<i>System Structure & Parameterization</i>)
UML	Lenguaje Unificado de Modelado (<i>Unified Modeling Language</i>)
URL	Localizador Uniforme de Recursos (<i>Uniform Resource Locator</i>)
XML	Lenguaje de Marcado Extensible (<i>Extensible Markup Language</i>)

Glosario de Términos Técnicos

Acausal	Estilo de modelado en Modelica donde las ecuaciones se declaran sin especificar la dirección del flujo de información; el compilador determina automáticamente la causalidad.
Canvas	Área de trabajo visual del editor donde el usuario arrastra, posiciona y conecta los componentes económicos.
Componente	En el contexto del editor, bloque reutilizable que encapsula una ecuación económica o una relación entre variables (p. ej. <code>FuncionConsumo</code> , <code>Solow_Capital</code>). Cada componente tiene conectores de entrada/salida, parámetros configurables y genera código Modelica válido al exportar.
Conector	Puerto de interfaz de un componente Modelica que define las variables que pueden intercambiarse con otros componentes mediante <code>connect()</code> .
Curva IS	Lugar geométrico de combinaciones (Y, r) que equilibran el mercado de bienes: $Y = C(Y) + I(r) + G$.
Curva LM	Lugar geométrico de combinaciones (Y, r) que equilibran el mercado de dinero: $M_d(Y, r) = M_s$.
DAE	<i>Differential Algebraic Equations</i> : sistema que mezcla ecuaciones diferenciales ordinarias con restricciones algebraicas, como los generados por Modelica.
Drag-and-drop	Técnica de interacción en la que el usuario arrastra un elemento visual con el ratón y lo suelta en una nueva posición para añadirlo o moverlo.

Economía Circular	Paradigma económico que propone sustituir el modelo lineal «extraer-producir-desechar» por ciclos de reutilización y regeneración del capital. En este TFM se implementa mediante el parámetro de circularidad ρ en la ecuación de acumulación del modelo Solow.
Estado estacionario	En el modelo Solow, valor de capital por trabajador k^* donde la inversión iguala exactamente la depreciación y el capital per cápita no varía: $s \cdot f(k^*) = (\delta + n) \cdot k^*$.
Modelo Doughnut	Marco teórico propuesto por Kate Raworth (2017) que define el espacio seguro y justo para la humanidad entre un suelo social (necesidades mínimas) y un techo ecológico (límites planetarios). Se utiliza en este TFM como referencia para las extensiones del editor.
MVC	<i>Model-View-Controller</i> : patrón arquitectónico que separa los datos (<i>Model</i>), la presentación (<i>View</i>) y la lógica de control (<i>Controller</i>).
ODE	<i>Ordinary Differential Equation</i> : ecuación que relaciona una función desconocida con sus derivadas respecto a una única variable independiente (generalmente el tiempo).
Parseador	Componente software que analiza la sintaxis de un texto (en este caso, un archivo <code>.mo</code>) y extrae su estructura para su procesamiento.
Tasa de ahorro	Fracción $s \in (0, 1)$ de la renta que se invierte; parámetro clave del modelo Solow que determina el nivel del estado estacionario.
Tasa de circularidad	Parámetro $\rho \in [0, 1]$ introducido en la extensión de Economía Circular del modelo Solow. Representa la fracción del capital depreciado que se recupera y reutiliza, reduciendo la depreciación efectiva a $(1 - \rho) \cdot \delta$ y elevando el estado estacionario de capital.
Techo ecológico	En el modelo Doughnut, conjunto de nueve límites planetarios que no deben superarse para mantener la estabilidad del sistema Tierra.

Índice general

Resumen	II
Abstract	III
Agradecimientos	IV
Listado de Siglas y Acrónimos	v
Índice de Figuras	xv
Índice de Tablas	xvii
Índice de Listados de Código	xix
1. Introducción, Objetivos y Estructura	1
1.1. Contexto y motivación	2
1.1.1. Simulación dinámica y modelos económicos clásicos	2
1.1.2. Limitaciones de las herramientas actualmente disponibles	3
1.1.3. Ventajas de Modelica y OpenModelica	3
1.2. Objetivos del trabajo y requisitos	4
1.2.1. Diseño de la biblioteca de componentes	6
1.2.2. Desarrollo del editor gráfico interactivo	6
1.2.3. Sistema de validación semántica	7
1.2.4. Exportador de código Modelica	7
1.2.5. Analizador sintáctico bidireccional	8
1.2.6. Integración con fuentes de datos económicos	8
1.2.7. Validación de los modelos	9
1.3. Estructura del documento	10
1.4. Conclusiones	12
2. Revisión del Estado del Arte	14
2.1. Introducción	14
2.2. Modelo IS-LM	14
2.3. Modelo Solow	17
2.4. Modelica como entorno de modelado orientado a objetos	19
2.5. Herramientas existentes para la simulación	21

2.5.1.	Herramientas web e interactivas	21
2.5.2.	Software de cálculo y lenguajes de programación	24
2.5.3.	Software especializado en macroeconomía	24
2.5.4.	Implementación en Modelica y librerías específicas	26
2.6.	Economía Circular y modelo Doughnut: estado del arte	27
2.6.1.	Investigaciones previas	30
2.7.	Conclusiones	30
3.	Diseño e Implementación de los Modelos IS-LM y Solow	32
3.1.	Introducción	32
3.2.	IS-LM: hipótesis y simplificaciones	33
3.2.1.	Hipótesis del modelo IS-LM	34
3.2.2.	Simplificaciones del modelo IS-LM	35
3.3.	IS-LM: descripción y formulación matemática	35
3.3.1.	Ecuaciones del mercado de bienes (curva IS)	36
3.3.2.	Ecuaciones del mercado de dinero (curva LM)	37
3.3.3.	Equilibrio general IS-LM	37
3.4.	IS-LM: implementación en Modelica y componentes	38
3.4.1.	Implementación en Modelica del mercado de bienes	38
3.4.2.	Implementación en Modelica del mercado monetario	41
3.4.3.	Ensamblaje y ecuaciones de equilibrio	42
3.5.	Modelo de Solow: hipótesis y simplificaciones	45
3.5.1.	Hipótesis del modelo de Solow	45
3.5.2.	Simplificaciones del modelo de Solow	46
3.6.	Modelo de Solow: descripción y formulación matemática	46
3.6.1.	Función de producción Cobb-Douglas	47
3.6.2.	Ecuación de acumulación de capital	47
3.6.3.	Crecimiento poblacional	48
3.6.4.	Inversión	48
3.6.5.	Estado estacionario	48
3.6.6.	Variables per cápita	49
3.6.7.	Implementación en Modelica del modelo de Solow	50
3.7.	Extensión del modelo Solow: Economía Circular	54
3.7.1.	Motivación y formulación	54
3.7.2.	Componente Solow_Circularidad	55
3.8.	Conclusiones	57
4.	Patrones de Diseño y Arquitectura Software	58
4.1.	Introducción	58
4.2.	Arquitectura general	59

4.3.	Clase principal. <code>EditorModelicaMultiDominio</code>	67
4.3.1.	Atributos de instancia	67
4.3.2.	Inicialización de la interfaz	68
4.3.3.	Componentes Swing de la ventana principal	68
4.3.4.	Gestión del ciclo de vida de los modelos	69
4.3.5.	Auto-lanzamiento de <code>OMEdit</code>	69
4.3.6.	Escenarios económicos y descarga de datos reales	70
4.3.7.	Fuentes de datos soportadas	71
4.3.8.	Componentes de vista	71
4.4.	El lienzo del editor. <code>CanvasModelica</code>	72
4.4.1.	Sistema de interacción con el ratón	74
4.4.2.	Ciclo de renderizado	74
4.4.3.	Eliminación de elementos	75
4.5.	Representación de componentes	75
4.5.1.	Estructura interna	75
4.5.2.	Clase interna conector	76
4.5.3.	Método <code>inicializar()</code> como factoría	77
4.5.4.	Plantillas de modelos predefinidos	78
4.6.	Representación de conexiones. <code>ConexiónModelica</code>	80
4.7.	Generador-Exportador a código Modelica	81
4.7.1.	Estructura del archivo <code>.mo</code> generado	82
4.7.2.	Transformación de coordenadas	84
4.7.3.	Ausencia de validación sintáctica en el exportador	84
4.7.4.	Generación de documentación HTML embebida	85
4.7.5.	Integración de datos reales: componentes <code>TimeTable</code>	85
4.8.	Parseador de archivos Modelica	86
4.8.1.	Patrones de extracción	87
4.8.2.	Transformación inversa de coordenadas	88
4.8.3.	Integración con el sistema editor	88
4.9.	Sistema de validación de conexiones	89
4.9.1.	Estructura de <code>ResultadoValidacion</code>	91
4.9.2.	Validación sintáctica de tipos	91
4.9.3.	Validación semántica IS-LM	91
4.9.4.	Validación semántica Solow	92
4.9.5.	Validación semántica Doughnut	93
4.9.6.	Integración con el canvas	93
4.10.	Integración con fuentes de datos externas	95
4.10.1.	Arquitectura interna del servicio	96
4.10.2.	Gestión de errores y rendimiento	97

4.11. Visor de código integrado	98
4.11.1. Flujo de datos del visor	98
4.11.2. Resaltado de sintaxis	99
4.12. Mecanismo de extensibilidad	99
4.13. Conclusiones	102
5. Validación del Sistema y Pruebas	104
5.1. Introducción	104
5.2. Verificación del modelo IS-LM	105
5.3. Verificación del modelo de Solow	107
5.4. Prueba de validación de conexiones incompatibles	107
5.4.1. Reglas del modelo IS-LM	107
5.4.2. Reglas del modelo Solow	108
5.4.3. Resultados de las pruebas de conexiones	109
5.5. Prueba de importación de modelo existente	110
5.6. Validación analítica del equilibrio IS-LM	111
5.6.1. Derivación de la curva IS	112
5.6.2. Derivación de la curva LM	113
5.6.3. Resolución del sistema: equilibrio simultáneo	113
5.6.4. Verificación del equilibrio	114
5.7. Validación de política fiscal y monetaria	115
5.7.1. Variación de la política fiscal	116
5.7.2. Cuantificación del efecto multiplicador fiscal	117
5.7.3. Cuantificación del efecto crowding-out	117
5.7.4. Variación de la política monetaria	119
5.7.5. Efectos sobre las variables macroeconómicas	120
5.7.6. Multiplicador monetario	121
5.7.7. Caso España: Gasto Público 2010–2023	122
5.7.8. IS-LM con datos reales EE.UU.	124
5.8. Verificación matemática del modelo de Solow	126
5.8.1. Parámetros de validación	126
5.8.2. Cálculo analítico del estado estacionario	127
5.8.3. Convergencia a 200 años	127
5.8.4. Simulación de convergencia a 100 años	129
5.9. Solow EE.UU. con datos históricos reales (1970–2024)	130
5.9.1. Solow con datos reales EE.UU.	130
5.9.2. Concepto clave: estado estacionario móvil	132
5.9.3. Parámetros de simulación EE.UU.	133
5.9.4. Estado estacionario teórico	133

5.9.5. Comparativa con datos reales (1970–2024)	134
5.9.6. Proyección a 130 años	135
5.10. Validación del modelo Solow con Economía Circular	136
5.10.1. Parámetros de validación	136
5.10.2. Cálculo analítico del estado estacionario	137
5.10.3. Comparativa de resultados	137
5.10.4. Validación del modelo Solow circular mediante simulación en Open-Modelica	138
5.10.5. Interpretación económica	140
5.11. Dymola. Carga de modelos	141
5.12. Conclusiones	143
6. Planificación y Presupuesto	145
6.1. Introducción	145
6.2. Planificación	145
6.3. Presupuesto	149
6.4. Conclusiones	150
7. Conclusiones y Trabajo Futuro	151
7.1. Introducción	151
7.2. Limitaciones del trabajo	151
7.3. Conclusiones	152
7.4. Líneas de trabajo futuro	154
7.4.1. Extensiones de los modelos IS-LM y Solow	155
7.4.2. Componente <code>Solow_Circularidad</code> : extensión de Economía Circular	156
7.4.3. Componentes del modelo Doughnut	157
7.4.4. Arquitectura de software: patrones y calidad	161
7.4.5. Integración profunda con OpenModelica	162
7.4.6. Exportación e interoperabilidad	162
7.4.7. Integración con herramientas externas	163
7.5. Conclusiones	164
Bibliografía	164
A. Ejemplos de Código Generado	170
A.1. Modelo IS-LM completo	171
A.2. Modelo de Solow completo	173
B. Guía Rápida de Uso	176
B.1. Instalación	176
B.2. Configuración de OpenModelica (OMEdit)	176

B.3. Añadir componentes	177
B.4. Editar parámetros	179
B.5. Conectar componentes	180
B.6. Cargar datos históricos reales	180
B.7. Exportar y simular en OpenModelica	182
B.8. Zoom y navegación	182

Índice de figuras

1.1. Flujo de trabajo del editor	4
1.2. Visión general del editor	10
2.1. Wolfram Demonstration Project	22
2.2. Diagrama causal del modelo IS-LM en macrosimulation.org	23
2.3. Producto de equilibrio Y bajo seis escenarios en macrosimulation.org	23
2.4. Interfaz de EViews 9.5	25
2.5. El ciclo de la Economía Circular	28
2.6. Proyecto MOSES en OpenModelica	30
4.1. Arquitectura MVC detallada	63
4.2. Diagrama de clases UML: capa Vista	65
4.3. Diagrama de clases UML: capa Modelo y Controlador	66
4.4. Diagrama de secuencia UML	73
4.5. De Componente y Conector a Conexión y código	81
4.6. Pipeline de exportación a Modelica	82
4.7. Arquitectura de parseo regex	87
4.8. Diagrama de clases de <code>ValidadorConectores</code>	90
4.9. Arquitectura de procesamiento JSON	96
4.10. Secuencia de descarga de datos	97
4.11. Mecanismo de extensibilidad en 4 pasos	101
5.1. Editor gráfico con modelo IS-LM	106
5.2. Modelo IS-LM en OMEdit	106
5.3. Validación de conexión coherente IS-LM.	110
5.4. Editor gráfico: prueba de exportación.	111
5.5. Variables de equilibrio IS-LM en OpenModelica.	115
5.6. OpenModelica: política fiscal $G = 300$	118
5.7. OpenModelica: valores con la política monetaria $M_s = 1200$	121
5.8. Panel de variables tras expansión monetaria	122
5.9. OpenModelica: gasto público de España 2010–2023.	124
5.10. IS-LM EE.UU.: producto nacional Y	125
5.11. IS-LM EE.UU.: tipo de interés r	125
5.12. OpenModelica: simulación Solow 200 años.	128
5.13. OpenModelica: estado estacionario Solow.	129

5.14. Solow EE.UU.: capital por trabajador k	131
5.15. Solow EE.UU.: producción por trabajador y	131
5.16. Solow EE.UU. (1970–2024): inversión por trabajador i	132
5.17. OpenModelica: Solow EE.UU. $t=2024$	134
5.18. OpenModelica: Solow EE.UU. $t=2100$	136
5.19. Solow Circular $\rho = 0,0$	138
5.20. Solow Circular $\rho = 0,3$	139
5.21. Solow Circular $\rho = 0,6$	139
5.22. Dymola: modelo Solow cargado correctamente.	141
5.23. Dymola: código Modelica del modelo IS-LM EE.UU.	142
6.1. Cronograma del proyecto TFM: editor gráfico de modelos económicos (diagrama de Gantt)	146
7.1. Modelo lineal vs. circular	155
7.2. Diagrama del modelo Doughnut	159
B.1. Diálogo de configuración de la ruta de OMEdit.	177
B.2. Paleta de componentes y plantillas disponibles.	178
B.3. Panel de propiedades del componente seleccionado.	179
B.4. Menú contextual del canvas (botón derecho del ratón).	180
B.5. Carga del modelo IS-LM con datos reales del Banco Mundial.	181
B.6. Selector de país para la descarga de datos históricos.	181

Índice de tablas

2.1. Comparativa de herramientas	25
2.2. Variables del modelo Doughnut	29
3.1. Variables y parámetros del modelo IS-LM.	33
3.2. Dependencia de variables IS-LM	43
3.3. Variables y parámetros del modelo de Solow.	45
3.4. Dependencia de variables Solow	52
4.1. Patrones y principios aplicados en el editor	59
4.2. Mapa de clases del editor	60
4.3. Atributos principales de <code>EditorModelicaMultiDominio</code>	67
4.4. Componentes Swing de la ventana principal.	68
4.5. Atributos principales de <code>ComponenteModelica</code>	76
4.6. Registro completo de componentes del editor (16 tipos).	78
4.7. Patrones Regex del parseador y elementos extraídos.	88
4.8. Indicadores del Banco Mundial	95
5.1. Resultados de las pruebas de validación de conexiones.	109
5.2. Parámetros del modelo IS-LM base para la validación analítica.	112
5.3. Resumen expansión fiscal ($G: 200 \rightarrow 300$).	118
5.4. Efectos de la expansión monetaria	120
5.5. Gasto público de España	123
5.6. Parámetros estándar del modelo de Solow para la validación.	127
5.7. Comparativa estado estacionario Solow: simulación 200 años.	128
5.8. Comparativa estado estacionario Solow: simulación 100 años.	129
5.9. Evolución del estado estacionario objetivo en EE.UU. con datos reales. . .	132
5.10. Parámetros de la simulación Solow EE.UU. 1970–2024.	133
5.11. Comparativa resultados Solow EE.UU. 54 años (1970–2024).	135
5.12. Tabla comparativa Solow EE.UU. 130 años.	135
5.13. Escenarios de circularidad	137
5.14. Estado estacionario circular	138
5.15. Estado estacionario: analítico vs. simulado	140
5.16. Resumen de los resultados de validación (todas las pruebas superadas). . .	143
6.1. Cronograma de desarrollo del editor gráfico.	147
6.2. Presupuesto estimado del proyecto TFM (agosto–diciembre 2025).	150

7.1. Resumen de objetivos del TFM y resultados obtenidos.	154
7.2. Extensiones futuras del editor	156
7.3. Vías de integración técnica del editor con herramientas externas.	163

Índice de Listados de Código

3.1. Función de consumo en Modelica (<code>FuncionConsumo</code>).	39
3.2. Función de inversión en Modelica (<code>FuncionInversion</code>).	39
3.3. Gasto público en Modelica (<code>GastoPublico</code>).	40
3.4. Mercado de bienes IS en Modelica (<code>MercadoBienes</code>).	40
3.5. Oferta monetaria y demanda de dinero en Modelica.	41
3.6. Mercado monetario LM en Modelica (<code>MercadoMonetario</code>).	42
3.7. Modelo IS-LM completo en Modelica (anotaciones visuales omitidas). . . .	44
3.8. Componente <code>Solow_Capital</code> en Modelica.	50
3.9. Componente <code>Solow_Produccion</code> en Modelica.	50
3.10. Componente <code>Solow_Poblacion</code> en Modelica.	51
3.11. Componente <code>Solow_Inversion</code> en Modelica.	51
3.12. Modelo de Solow completo ensamblado en Modelica.	53
3.13. Componente <code>Solow_Circularidad</code> : Economía Circular integrada en el mo- delo de Solow.	56
4.1. Auto-lanzamiento de <code>OMEdit</code> : detección multiplataforma de la ruta de ins- talación mediante <code>ProcessBuilder</code>	70
4.2. Biblioteca de componentes IS-LM y Solow (patrón <code>Factory</code> en <code>inicializar()</code>).	77
4.3. <code>crearModeloISLMRapido()</code> : plantilla que instancia los 7 componentes IS- LM y sus 9 conexiones causales en un clic.	79
4.4. Estructura del archivo <code>.mo</code> generado por el exportador: definiciones de tipos, modelo <code>Main</code> e instrucciones <code>connect</code>	83
4.5. Transformación de coordenadas Java a Modelica: cambio de escala y origen. .	84
4.6. Código generado con datos históricos del Banco Mundial: bloque <code>TimeTable</code> con interpolación lineal.	86
4.7. Extracto de reglas semánticas IS-LM: <code>HashMap</code> de conexiones válidas con mensajes educativos.	92
4.8. Integración del validador en <code>CanvasModelica.intentarConectar()</code> : el usuario puede forzar incluso una conexión con <code>ERROR</code>	94
4.9. Flujo de datos del visor de código (pseudocódigo): del canvas al texto re- saltado.	98
4.10. Paso 1: <code>ComponenteModelica.java</code> — definición del componente <code>Impuestos</code> . .	100
4.11. Paso 2: <code>PaletaComponentes.java</code> — entrada en la paleta.	100
4.12. Paso 3: <code>ValidadorConectores.java</code> — regla semántica.	100
4.13. Paso 4: <code>ExportadorModelica.java</code> — ecuación Modelica del componente. . .	101

7.1. Componente <code>Solow_Circularidad</code> : tasa de circularidad en el modelo de Solow.	157
7.2. Componente <code>LimitePlanetario_Agua</code> : techo ecológico sobre la productividad Solow.	160
7.3. Componente <code>SueloSocial_Equidad</code> : equidad sobre la propensión a consumir IS-LM.	161
A.1. Ejemplo basico de modelo IS-LM generado por el editor.	170
A.2. Modelo IS-LM completo generado por el editor.	171
A.3. Modelo de Solow completo generado por el editor.	173

Capítulo 1

Introducción, Objetivos y Estructura

El presente Trabajo de Fin de Máster aborda el desarrollo de un **editor gráfico de modelos económicos descritos en lenguaje Modelica** implementado en Java, especializado en el modelado y simulación de sistemas económicos mediante el **lenguaje Modelica**. La aplicación desarrollada permite la creación visual de modelos macroeconómicos clásicos, concretamente el **modelo IS-LM** (Investment-Saving / Liquidity preference-Money supply) y el **modelo de crecimiento de Solow**, con capacidad de **exportación directa a OpenModelica** para su simulación y **análisis numérico detallado**.

OpenModelica (OpenModelica, 2026) es el entorno de referencia de **código abierto** para el lenguaje Modelica, desarrollado por el **Open Source Modelica Consortium** bajo coordinación de la Universidad de Linköping (Suecia). Su modelo de desarrollo abierto permite que investigadores, empresas y comunidades académicas contribuyan con nuevas librerías y extensiones, lo que ha dado lugar a un ecosistema diverso de componentes reutilizables para múltiples dominios físicos e ingenieriles. Esta filosofía open source resulta especialmente relevante en el ámbito académico, ya que facilita el acceso sin coste a herramientas de simulación de nivel profesional y favorece la reproducibilidad de resultados científicos.

El lenguaje Modelica se ha consolidado como un **estándar *de facto*** en el modelado de sistemas físicos complejos, particularmente en dominios como la ingeniería mecánica, eléctrica, térmica y de control. Su arquitectura orientada a objetos, su enfoque **acausal basado en ecuaciones** y su capacidad para manejar **sistemas híbridos continuos-discretos** lo convierten en una herramienta potente para la descripción de **sistemas dinámicos**. Sin embargo, su aplicación en el ámbito económico ha sido escasamente explorada, a pesar de que los modelos económicos comparten características estructurales similares a los sistemas físicos: ecuaciones diferenciales, relaciones causales, dinámicas temporales y comportamientos emergentes de la interacción de múltiples componentes.

Este trabajo pretende demostrar la viabilidad técnica y la utilidad práctica de extender el uso del lenguaje Modelica al dominio económico, tomando OpenModelica (OpenModelica, 2026) como entorno de referencia para la exportación y simulación. Los modelos generados son compatibles también con Dymola (Dassault Systèmes, 2024), cuya compatibilidad se verifica en la Sec. 5.11. La contribución principal consiste en el desarrollo de una aplicación

en Java que pueda generar archivos Modelica con componentes económicos reutilizables y un editor gráfico que reduce la **barrera de entrada** para estudiantes de economía o cualquier persona interesada en estos temas que desee aprovechar las capacidades de simulación de Modelica sin necesidad de dominar los aspectos técnicos del lenguaje.

1.1 Contexto y motivación

La sección se organiza en tres bloques: el valor de la simulación dinámica en ciencias sociales y los modelos IS-LM y Solow (Sec. 1.1.1), las limitaciones de las herramientas actualmente disponibles (Sec. 1.1.2), y las ventajas que aporta Modelica/OpenModelica frente a esas limitaciones (Sec. 1.1.3).

1.1.1. Simulación dinámica y modelos económicos clásicos

El desarrollo de simulaciones en ciencias sociales representa un avance fundamental para la comprensión y gestión de sistemas complejos como economías, organizaciones y sociedades. Las simulaciones dinámicas permiten capturar las interacciones no-lineales, retroalimentaciones y comportamientos emergentes que caracterizan a estos sistemas, superando las limitaciones de análisis estáticos tradicionales. Esta capacidad de modelar escenarios alternativos y probar políticas públicas o estrategias empresariales sin incurrir en costes reales proporciona a los responsables de la toma de decisiones una herramienta valiosa para la evaluación de riesgos y la exploración de futuros alternativos.

Los modelos macroeconómicos tradicionales, como el IS-LM desarrollado por John Hicks (Hicks, 1937) en 1937 como interpretación matemática de la Teoría General del Empleo, el Interés y el Dinero de John Maynard Keynes, y el modelo de crecimiento neoclásico de Solow presentado en 1956, constituyen herramientas fundamentales para el análisis de políticas económicas y la comprensión de fenómenos macroeconómicos. Estos modelos han sobrevivido décadas de evolución del pensamiento económico precisamente porque capturan los mecanismos esenciales del funcionamiento de las economías modernas.

El modelo IS-LM describe el equilibrio simultáneo en los mercados de bienes y servicios (**curva IS**, que representa combinaciones de producción y tipo de interés donde el ahorro iguala a la inversión) y en el mercado monetario (**curva LM**, que representa combinaciones donde la demanda de dinero iguala a la oferta monetaria controlada por el banco central). Este marco analítico permite estudiar los efectos de corto plazo de políticas fiscales (cambios en gasto público o impuestos) y políticas monetarias (cambios en la oferta

monetaria o tipos de interés) sobre variables agregadas como la producción nacional y los tipos de interés.

El modelo de Solow, por su parte, explica el crecimiento económico a largo plazo mediante la acumulación de capital físico (maquinaria, infraestructura, edificios productivos), el crecimiento de la fuerza laboral y el progreso tecnológico. El modelo predice convergencia de economías hacia estados estacionarios caracterizados por niveles de capital per cápita constantes, con el crecimiento sostenido de largo plazo dependiendo exclusivamente del progreso tecnológico exógeno. Este marco teórico ha sido fundamental para comprender por qué algunas economías crecen más rápidamente que otras y en qué condiciones ocurre convergencia de niveles de vida entre países.

1.1.2. Limitaciones de las herramientas actualmente disponibles

A pesar de su importancia pedagógica y analítica, estos modelos carecen frecuentemente de herramientas software especializadas que faciliten su construcción visual, parametrización con datos reales y simulación dinámica. Los economistas típicamente implementan estos modelos en hojas de cálculo (Excel), entornos estadísticos (R, Stata) o lenguajes de programación general (Python, MATLAB). Cada una de estas aproximaciones presenta limitaciones: las hojas de cálculo resultan propensas a errores y difíciles de mantener para modelos complejos, los entornos estadísticos requieren conocimientos de programación que muchos economistas no poseen, los lenguajes generales carecen de abstracciones específicas para modelado económico. Cada una de estas alternativas presenta ventajas parciales, pero ninguna integra todas las funcionalidades necesarias en un único entorno.

1.1.3. Ventajas de Modelica y OpenModelica

Frente a los entornos de propósito general, OpenModelica destaca porque automatiza los aspectos más técnicos de la simulación dinámica: la **integración numérica** (sin necesidad de implementar manualmente métodos como Runge-Kutta (Butcher, 2016)), la **resolución simultánea de sistemas de ecuaciones**, la **detección automática de estados estacionarios**, el **manejo de sistemas DAE** (ecuaciones diferenciales mezcladas con algebraicas) y la **generación de código C optimizado**. Además, soporta componentes reutilizables, conexiones acausales y jerarquías de modelos — todo ello con licencia de código abierto.

La extensión de Modelica al dominio económico requiere no solamente la definición de componentes apropiados que representen funciones de comportamiento económico (funciones

de consumo, inversión, demanda de dinero, producción agregada, etc.), sino también el desarrollo de herramientas que permita construir modelos sin necesidad de conocimientos profundos de la sintaxis del lenguaje Modelica o de los detalles de implementación de los solvers numéricos.

La Figura 1.1 resume el flujo de trabajo que sigue el editor propuesto, desde el modelo económico hasta su simulación y validación en OpenModelica, incorporando datos reales e iterando sobre los parámetros del modelo.

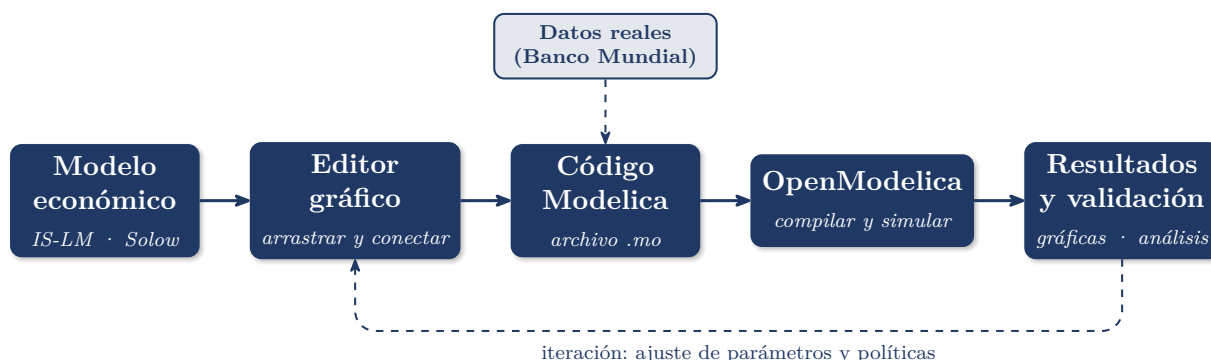


Figura 1.1: Flujo de trabajo del editor: del modelo económico a la simulación y validación en OpenModelica.

La motivación adicional para este trabajo surge de la observación de que la complejidad de los modelos económicos empleados en investigación y análisis de políticas ha aumentado en las últimas décadas. Los modelos dinámicos estocásticos de equilibrio general (DSGE), ampliamente utilizados por bancos centrales y organismos internacionales, involucran decenas de ecuaciones no lineales, múltiples sectores económicos, rigideces y fricciones financieras además de expectativas racionales de futuro. La construcción, calibración y simulación de estos modelos complejos demanda herramientas software sofisticadas. Aunque el presente trabajo se enfoca en modelos relativamente simples (IS-LM y Solow), la arquitectura desarrollada sienta las bases para futuras extensiones hacia modelos más complejos, demostrando la viabilidad del enfoque de emplear Modelica en la descripción de modelos económicos. Las secciones siguientes detallan los objetivos y requisitos (Sec. 1.2) y la organización del documento (Sec. 1.3).

1.2 Objetivos del trabajo y requisitos

El objetivo principal del presente trabajo es el diseño, implementación y validación de una aplicación Java que permita el **modelado visual** de sistemas económicos dinámicos utilizando componentes predefinidos basados en el lenguaje Modelica, con capacidad de exportación a formato estándar .mo para simulación en OpenModelica.

Requisitos iniciales del trabajo:

- Diseñar e implementar una librería en lenguaje Modelica para la descripción de modelos económicos (IS-LM y Solow), junto con un editor gráfico que permita crear modelos compuestos a partir de los componentes de dicha librería.
- Generar el código del modelo compuesto como archivo Modelica de extensión `.mo`, con la información de componentes, coordenadas y conexiones necesaria para que el archivo pueda abrirse en el entorno gratuito OpenModelica.

El editor debe permitir abrir y guardar archivos creados con el propio editor, y ofrecer como mínimo las siguientes capacidades:

- Zoom de acercamiento y alejamiento sobre el modelo del lienzo.
- Panel lateral derecho para visualizar y modificar los atributos, valores y descripción de los objetos seleccionados con el ratón.
- Panel lateral izquierdo con los componentes económicos disponibles: funciones individuales (consumo, inversión, gasto público, demanda monetaria) y modelos completos predefinidos (IS-LM y Solow), todos arrastrables al lienzo.
- Conexión de objetos y modelos entre sí para construir un modelo que describa un comportamiento económico determinado.
- Movimiento libre de los objetos en el lienzo, manteniendo sus conexiones.
- Registro o *log* en la parte inferior de la pantalla que confirme cada acción del usuario e indique si es correcta o ha producido un error.
- Plantillas de los modelos IS-LM y Solow, cargables desde la barra lateral derecha, ya conectadas y con valores genéricos modificables. Estas plantillas permiten además la descarga automática de series temporales reales desde la API del Banco Mundial: gasto público (% PIB) y oferta monetaria para el modelo IS-LM, e inversión y crecimiento poblacional para el modelo de Solow. Los datos descargados se incorporan directamente al archivo `.mo` generado.
- Compatibilidad del archivo generado con OpenModelica para su simulación, transformando los datos del Banco Mundial en gráficas por año de las funciones y variables incorporadas según el modelo precargado (IS-LM o Solow).

Para alcanzar este objetivo general, el trabajo se ha descompuesto en los siguientes requisitos específicos, organizados por el componente o fichero Java principal responsable de cada uno.

1.2.1. Diseño e implementación de la biblioteca de componentes económicos

Desarrollar una biblioteca de componentes Modelica que representen las ecuaciones fundamentales de los modelos IS-LM y Solow. Para el modelo IS-LM, esto incluye componentes que implementen la función de consumo keynesiana con consumo autónomo y propensión marginal a consumir, la función de inversión sensible al tipo de interés, el gasto público exógeno o endógeno, la demanda de dinero con componentes transaccional y especulativa, la oferta monetaria controlada por el banco central, y componentes agregadores que materialicen los equilibrios de mercado de bienes y mercado monetario.

Para el modelo de Solow, se requieren componentes que implementen la función de producción agregada (típicamente Cobb-Douglas con rendimientos constantes a escala), la ecuación de acumulación de capital que relacione inversión, depreciación y stock de capital mediante ecuaciones diferenciales, el crecimiento poblacional exponencial a tasa constante, y la función de ahorro que traduzca producción en inversión mediante la tasa de ahorro exógena. Cada componente debe definir claramente sus parámetros configurables, sus conectores de entrada y salida, y las ecuaciones matemáticas que gobiernan su comportamiento.

Además, se construirá de manera modular para que sea fácil y asequible poder añadir más funciones y componentes al sistema en un futuro.

1.2.2. Desarrollo del editor gráfico interactivo

Implementar un editor gráfico completo en Java utilizando Swing que permita la construcción visual de modelos económicos mediante interacción natural de **arrastrar y soltar**. El editor debe proporcionar un lienzo de trabajo donde el usuario pueda colocar componentes, moverlos libremente, ajustar el nivel de zoom sobre el lienzo, y crear conexiones visuales entre sus conectores. La interfaz debe incluir una paleta lateral que muestre todos los componentes disponibles organizados por bibliotecas (IS-LM, Solow), con iconos o texto descriptivos y etiquetas claras. El usuario debe poder seleccionar múltiples componentes simultáneamente para operaciones en bloque como abrir todo el conjunto del modelo IS-LM o el modelo Solow en un solo clic.

El proceso de creación de conexiones debe ser intuitivo: el usuario hace clic en un conector de salida de un componente, arrastra el cursor hasta un conector de entrada de otro componente, y suelta para crear la conexión. Durante el proceso de arrastre, una línea provisional debe seguir el cursor mostrando visualmente dónde quedará la conexión. El

sistema debe destacar visualmente los conectores compatibles cuando el usuario pasa el cursor sobre ellos, facilitando la identificación de conexiones válidas. Los conectores incompatibles deben indicarse claramente para prevenir errores de modelado.

1.2.3. Sistema de validación semántica

Desarrollar un **sistema robusto de validación de conexiones** que verifique no solamente la compatibilidad sintáctica de tipos entre conectores (RealInput debe conectarse con RealOutput), sino también la compatibilidad semántica según las reglas del dominio económico. El validador debe implementar conocimiento sobre la estructura correcta de los modelos IS-LM y Solow. Esto supone verificar, por ejemplo, que la salida de consumo se conecta a la entrada del mercado de bienes, que la demanda de dinero se vincula al mercado monetario, y que existen las realimentaciones necesarias entre mercados (el producto nacional Y debe retroalimentarse desde el mercado de bienes hacia las funciones de consumo y demanda de dinero, el tipo de interés r debe retroalimentarse desde el mercado monetario hacia la función de inversión).

El sistema de validación debe proporcionar tres niveles de mensajes: errores que impiden absolutamente la creación de la conexión (por ejemplo, conectar dos salidas entre sí), advertencias que señalan conexiones técnicamente válidas, pero económicamente sospechosas (por ejemplo, conectar la función de consumo directamente a la función de inversión sin pasar por el mercado de bienes), e información que confirma conexiones válidas y apropiadas. Los mensajes deben ser pedagógicamente valiosos, explicando no solamente qué está mal sino por qué y qué alternativa sería correcta, convirtiendo el proceso de validación en un proceso de aprendizaje.

1.2.4. Exportador de código Modelica

Implementar un **exportador de código Modelica** que genere archivos de texto en formato estándar (.mo) sintáctica y semánticamente correctos, que puedan abrirse directamente en OpenModelica para compilación y simulación. El código generado debe incluir todas las declaraciones de componentes con sus parámetros configurados según los valores especificados por el usuario en el editor, todas las ecuaciones de conexión mediante instrucciones connect de Modelica, y anotaciones visuales completas que preserven la disposición gráfica del modelo (posiciones de componentes, trazado de conexiones).

Las anotaciones visuales resultan críticas para la interoperabilidad con otros editores Modelica. Cuando un usuario exporta un modelo desde la aplicación y posteriormente lo

abre en OMEdit (el editor oficial de OpenModelica), debe ver el modelo con exactamente la misma disposición visual. Esto requiere transformaciones precisas entre el sistema de coordenadas del lienzo Java (origen en esquina superior izquierda, coordenadas en píxeles) y el sistema de coordenadas Modelica (origen típicamente centrado, coordenadas normalizadas en rango -100 a 100). El exportador debe calcular estas transformaciones automáticamente, incluyendo escalado y traslación de componentes.

1.2.5. Analizador sintáctico bidireccional

Desarrollar un **parseador bidireccional** que permita la lectura de archivos Modelica existentes y su representación visual en el editor, habilitando flujos de trabajo de edición iterativa. El parseador debe procesar el archivo línea por línea, identificando mediante expresiones regulares las diferentes construcciones sintácticas: declaraciones de paquetes e importaciones, declaraciones de componentes con sus tipos y parámetros, ecuaciones de conexión, y las anotaciones visuales de posicionamiento y trazado.

El parseador debe ser robusto frente a variaciones sintácticas permitidas por Modelica: comentarios en diferentes posiciones, orden arbitrario de declaraciones (componentes pueden declararse antes o después de las ecuaciones), espaciado variable (tabulaciones, múltiples espacios, saltos de línea), y presencia o ausencia de elementos opcionales. Debe manejar errores de parsing, proporcionando mensajes descriptivos que indiquen la línea y columna donde se detectó el problema y sugiriendo posibles correcciones. Para archivos que no contengan anotaciones visuales (por ejemplo, archivos escritos manualmente en un editor de texto), el parseador debe ser capaz de generar un layout visual automático razonable, distribuyendo componentes en una cuadrícula regular.

1.2.6. Integración con fuentes de datos económicos

Implementar **integración con la API REST del Banco Mundial** para obtener datos macroeconómicos reales (World Development Indicators), repositorio que almacena una gran cantidad de datos macroeconómicos, permitiendo la parametrización automática de modelos con información actualizada. El sistema debe proporcionar una interfaz donde el usuario pueda seleccionar país y según la disponibilidad de datos macro de dicho país se descargarán del Banco Mundial, indicadores macroeconómicos como: PIB, gasto público, oferta monetaria, ahorro nacional, crecimiento poblacional, etc, en un rango temporal, descargando automáticamente la serie histórica correspondiente.

Los datos descargados deben poder asignarse directamente a parámetros de componentes.

Por ejemplo, el usuario puede acceder al menú de descarga de datos, elegir un país entre los que hay disponibles del Banco Mundial y el componente **GastoPublico** u otros que haya disponibles, como tasa crecimiento poblacional, tasas de ahorro, etc, automáticamente se descargarán, si está disponible el dato Gasto Público G, el parámetro G del componente cargará los datos descargados y se configurará con los valores más recientes disponibles.

El sistema debe almacenar series temporales completas asociadas a componentes, permitiendo potencialmente simulaciones con parámetros variantes en el tiempo (aunque la versión básica de los modelos IS-LM y Solow emplea parámetros constantes).

La implementación debe incluir manejo de algunos errores de red (timeouts, servidores no disponibles), parsing tolerante a fallos de respuestas JSON (algunos registros pueden contener valores nulos o datos faltantes), y sistema de logging que informe al usuario del progreso de las descargas, del número de registros obtenidos y posibles advertencias sobre la calidad de datos.

1.2.7. Validación de los modelos

Validar **comprehensivamente la corrección** de los modelos generados mediante simulación en OpenModelica y verificación sistemática de que los resultados obtenidos coinciden cualitativamente y, cuando sea posible, cuantitativamente con las predicciones de la teoría económica establecida. Para el modelo IS-LM, esto implica verificar que un incremento del gasto público desplaza la curva IS hacia la derecha, aumentando tanto la producción como el tipo de interés de equilibrio (fenómeno de crowding out); que un incremento de la oferta monetaria desplaza la curva LM hacia la derecha, aumentando la producción y reduciendo el tipo de interés.

Para el modelo de Solow, la validación requiere verificar tres condiciones: (i) que el sistema converge asintóticamente hacia el estado estacionario predicho; (ii) que la velocidad de convergencia depende de los parámetros del modelo; y (iii) que el capital por trabajador en dicho estado estacionario crece con la tasa de ahorro y decrece con el crecimiento poblacional, sin afectar a la tasa de crecimiento de largo plazo en ausencia de progreso tecnológico (la expresión cerrada de este resultado se deriva formalmente en el Capítulo 3 y se contrasta numéricamente en el Capítulo 5). El modelo se ejercita además con distintos conjuntos de parámetros (datos reales de EE.UU. y la extensión de economía circular) para comprobar la robustez del comportamiento.

El conjunto de estos objetivos se concreta en la arquitectura del editor, que la Figura 1.2 resume en tres capas junto con su flujo de trabajo.

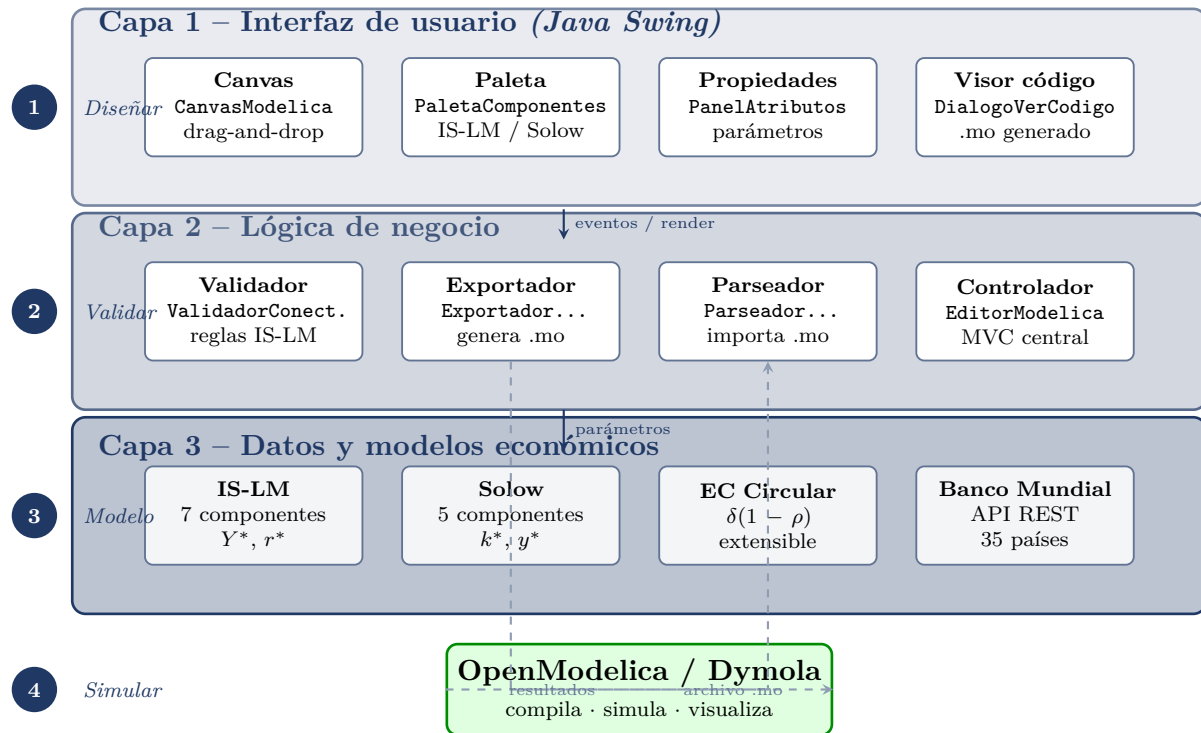


Figura 1.2: Visión general del editor: arquitectura en tres capas y flujo de trabajo.

1.3 Estructura del documento

El presente documento se estructura en siete capítulos principales y dos anexos que describen de forma sistemática y exhaustiva el desarrollo completo del proyecto, desde la concepción inicial hasta la validación final. El **Capítulo 1**, que el lector está actualmente leyendo, presenta la introducción general al trabajo, establece el contexto histórico y tecnológico, articula la motivación detrás del proyecto, enuncia los objetivos y requisitos principales y específicos con sus criterios de calidad mínima y describe la organización del documento.

El **Capítulo 2** ofrece una revisión exhaustiva y crítica del estado del arte en múltiples áreas relevantes al proyecto. Se analiza en profundidad el lenguaje Modelica, incluyendo su historia, filosofía de diseño, características principales, capacidades de modelado multi-dominio y ecosistema de herramientas. Se revisa la teoría económica subyacente a los modelos IS-LM y Solow, trazando su evolución histórica, formulación matemática, hipótesis fundamentales, predicciones principales y limitaciones conocidas. El capítulo incluye también una revisión del estado del arte en herramientas de simulación existentes y un análisis de los marcos alternativos de Economía Circular y modelo Doughnut como base teórica de las extensiones implementadas.

El **Capítulo 3** aborda el diseño e implementación de los modelos IS-LM y Solow en Mode-

lica. En primer lugar, presenta una introducción en el marco de la teoría macroeconómica, seguida de la descripción de los componentes y la formulación matemática completa de cada modelo. A continuación, desarrolla las derivaciones de las curvas IS-LM, la resolución del equilibrio simultáneo y la implementación en Modelica de ambos modelos, cerrando con la verificación y el análisis matemático.

En el **Capítulo 4**, el más extenso del documento, se examinan las tecnologías de desarrollo empleadas, particularmente Java y Swing, discutiendo sus ventajas y desventajas para el tipo de aplicación desarrollada. Se analiza la arquitectura software del sistema, los patrones de diseño aplicados y las decisiones de diseño fundamentales, se describe en profundidad el diseño detallado y la implementación concreta del sistema. Se presenta la arquitectura en capas del sistema, especificando responsabilidades de cada capa y protocolos de comunicación entre ellas. Se detallan las clases principales del sistema (`ComponenteModelica`, `ConexiónModelica`, `CanvasModelica`, `EditorModelicaMultiDominio`, `ExportadorModelica`, `ParseadorModelica`, `ValidadorConectores`, `ServicioDatosEconomicos`, `DialogoVerCodigo`) con sus atributos, métodos, relaciones y justificación de decisiones de diseño. Se explica detalladamente el proceso de generación de código Modelica, incluyendo las transformaciones de coordenadas necesarias para preservar el layout visual. Se describe el funcionamiento interno del parseador, mostrando ejemplos concretos de archivos Modelica y cómo se procesan. Se analiza el sistema de validación de conexiones con todas sus reglas implementadas y algunos otros temas a remarcar como el editor visual de código y el módulo de descarga de datos económicos.

El **Capítulo 5** presenta las pruebas realizadas para la validación y verificación del sistema. Se describen las pruebas de sistema y pruebas de aceptación (verificación de que el sistema satisface las necesidades de los usuarios). Se presentan casos de prueba con entradas, salidas esperadas y resultados obtenidos, incluyendo la validación económica mediante simulación en OpenModelica: choques de política fiscal y monetaria en IS-LM, convergencia al estado estacionario en Solow, y validación numérica del componente de Economía Circular para tres valores de ρ . El capítulo cierra con una prueba de carga del modelo en una versión reciente de Dymola.

El **Capítulo 6** documenta la planificación temporal y el presupuesto del proyecto. La planificación se articuló en **cinco fases** a lo largo de **26 semanas**: análisis de requisitos y diseño previo, diseño arquitectónico, implementación incremental en tres iteraciones, pruebas y validación extensiva, y optimización final con redacción de la memoria. La metodología combinó prácticas ágiles (Scrum) con el ciclo de vida en cascada, lo que permitió incorporar requisitos emergentes sin comprometer los plazos. El **presupuesto total asciende a 12.282 €**, desglosado en 12.000 € de personal (480 h a tarifa de

ingeniero junior), 102 € de amortización de hardware y 180 € de acceso a internet; el coste de software es nulo gracias al uso exclusivo de herramientas de código abierto (OpenJDK, OpenModelica, Git, TeXstudio, Overleaf).

El **Capítulo 7** presenta las conclusiones del trabajo y las líneas de trabajo futuro. La Sec. 7.2 recoge las limitaciones reconocidas; la Sec. 7.3 sintetiza las contribuciones técnicas y metodológicas; la Sec. 7.4 propone extensiones técnicas organizadas en cuatro bloques (variantes de los modelos IS-LM y Solow, nuevos componentes Modelica para Economía Circular y modelo Doughnut, mejoras de arquitectura software, e integración con OpenModelica y herramientas externas); y la Sec. 7.5 cierra el capítulo con las conclusiones integradoras del conjunto del trabajo.

Finalmente, el documento concluye con un apartado de referencias bibliográficas que recoge las fuentes consultadas durante el desarrollo del trabajo, incluyendo enlaces directos cuando están disponibles. Las referencias abarcan especificaciones técnicas del lenguaje Modelica, textos fundamentales de teoría económica, manuales de las herramientas empleadas, artículos sobre patrones de diseño software y publicaciones sobre simulación de sistemas económicos.

El documento incluye además dos **anexos**: el **Anexo A** presenta ejemplos completos de código Modelica generado por el editor para los modelos IS-LM y de Solow, ilustrando el formato y la estructura del archivo de salida; el **Anexo B** contiene una guía rápida de uso con los pasos esenciales de instalación, configuración y operación del editor.

1.4 Conclusiones

Este capítulo ha establecido el **para qué** del trabajo: existe una brecha entre la potencia formal de Modelica y la accesibilidad que necesitan los estudiantes y docentes de macroeconomía. Cerrar esa brecha mediante un editor gráfico de arrastrar y soltar, con datos reales integrados y exportación estándar a OpenModelica, es la contribución central de este TFM.

La elección de Modelica como lenguaje de destino no es arbitraria: su arquitectura acausal permite describir los modelos económicos con las mismas ecuaciones que aparecen en los libros de texto, sin requerir al usuario que los reformule en forma explícita. El editor hace de puente entre esa descripción declarativa y la simulación numérica que OpenModelica ejecuta automáticamente.

Lo que este TFM **no** pretende es construir un entorno de investigación de nivel profesional ni competir con Dymola o MATLAB: pretende demostrar que la arquitectura es viable y que la barrera de entrada puede reducirse de forma significativa con medios asequibles. Los capítulos siguientes documentan cómo se ha hecho.

Capítulo 2

Revisión del Estado del Arte

2.1 Introducción

El presente capítulo realiza una revisión del estado del arte en las áreas que fundamentan el desarrollo del editor gráfico de modelos económicos descritos en lenguaje Modelica. La revisión aborda progresivamente los modelos económicos específicos implementados (IS-LM y Solow), los fundamentos teóricos del lenguaje Modelica (Modelica Association, 2025) y sus capacidades de modelado multi-dominio, y el software disponible para la simulación de estos modelos, situando la contribución del presente trabajo en el contexto del conocimiento existente y justificando las decisiones de diseño adoptadas.

El capítulo se organiza como sigue: la Sec. 2.2 presenta el modelo IS-LM y su formulación matemática; la Sec. 2.3 hace lo propio con el modelo de crecimiento de Solow; la Sec. 2.4 introduce los fundamentos del lenguaje Modelica como entorno de modelado orientado a ecuaciones; la Sec. 2.5 compara las herramientas de simulación existentes, tanto genéricas como especializadas en macroeconomía; y la Sec. 2.6 cierra la revisión con los paradigmas económicos alternativos, Economía Circular y modelo Doughnut, que motivan las líneas de extensión futuras del editor.

2.2 Modelo IS-LM

El modelo IS-LM, desarrollado por John Hicks en 1937 a partir de las ideas de John Maynard Keynes (Hicks, 1937), es el marco de referencia para el análisis macroeconómico de corto plazo. Integra el mercado de bienes y el mercado monetario en un sistema de dos ecuaciones que determina simultáneamente el producto nacional Y y el tipo de interés r . Una implementación interactiva de referencia de este mismo modelo (con idénticas ecuaciones de equilibrio, multiplicador fiscal y efecto de *crowding out*) está disponible como simulación en R/Python en macrosimulation.org (Köhler, 2025a), y ha servido de contraste para la formulación adoptada en este trabajo.

El nombre IS-LM proviene de las dos curvas principales: **IS** representa el equilibrio en el mercado de bienes (*Investment = Saving*) y **LM** representa el equilibrio en el mercado de dinero (*Liquidity preference = Money supply*).

La **curva IS** relaciona producto nacional Y y tipo de interés r cuando el mercado de bienes está en equilibrio: es decir, cuando la demanda agregada iguala la producción:

$$Y = C(Y) + I(r) + G \quad (2.1)$$

donde $C(Y)$ es la función de consumo de los hogares, $I(r)$ la función de inversión (decreciente en el tipo de interés r) y G el gasto público exógeno.

La **curva LM** equilibra oferta y demanda de dinero, donde M_s es la oferta monetaria nominal y P el nivel general de precios. M_s/P representa la oferta monetaria real (saldos reales). En el modelo implementado en este TFM se asumen precios rígidos a corto plazo (P constante, normalizada a 1), por lo que la ecuación opera directamente con M_s :

$$M_s/P = L(Y, r) \quad (2.2)$$

donde M_s/P es la oferta monetaria real, $L(Y, r)$ la demanda de dinero creciente en Y y decreciente en r , P el nivel de precios (constante a corto plazo).

Descripción del Sistema Económico Modelado El modelo IS-LM representa una economía cerrada en el corto plazo, donde se determina simultáneamente el nivel de producción y el tipo de interés mediante el equilibrio en dos mercados interconectados. El sistema económico modelado incluye cuatro sectores principales y dos mercados que interactúan entre sí. A continuación se describen las principales variables:

Sectores de la Economía

El **sector de consumo** representa las decisiones de gasto de las familias. El consumo C depende positivamente del ingreso disponible: cuando las familias tienen más ingreso, consumen más. Esta relación se captura mediante la propensión marginal a consumir c , que indica qué fracción de cada euro adicional de ingreso se destina al consumo. Típicamente, c toma valores entre 0,6 y 0,9 en economías desarrolladas.

El **sector de inversión** refleja las decisiones de las empresas sobre compra de bienes de capital. La inversión I depende negativamente del tipo de interés r : cuando los tipos de interés son altos, los proyectos de inversión son menos rentables y las empresas invierten menos. La sensibilidad de la inversión al tipo de interés se representa mediante el

parámetro b .

El **sector público** comprende el gasto público G , que representa las compras de bienes y servicios por parte del gobierno. En el modelo básico, G se considera exógeno, es decir, determinado por decisiones políticas independientes del nivel de producción o del tipo de interés.

El **sector monetario** incluye la oferta monetaria M , controlada por el banco central, y la demanda de dinero. Las familias y empresas demandan dinero por tres motivos: para realizar transacciones (proporcional al ingreso Y), por precaución (también relacionado con Y), y por especulación (dependiendo inversamente del tipo de interés r).

El **mercado de bienes y servicios** es donde se determina la producción agregada. El equilibrio requiere que la producción Y sea igual a la demanda agregada, que es la suma de consumo, inversión y gasto público. Este equilibrio genera la curva IS , que muestra todas las combinaciones de producción y tipo de interés donde el mercado de bienes está en equilibrio.

El **mercado de dinero** es donde se determina el tipo de interés. El equilibrio requiere que la oferta monetaria real M/P (oferta nominal dividida por el nivel de precios) sea igual a la demanda de dinero. Este equilibrio genera la curva LM , que muestra todas las combinaciones de producción y tipo de interés donde el mercado de dinero está en equilibrio.

Interacción entre Mercados

La característica clave del modelo $IS-LM$ es la interdependencia entre los dos mercados. Un aumento en la producción incrementa la demanda de dinero (porque las familias necesitan más dinero para transacciones), lo que presiona al alza el tipo de interés en el mercado monetario. A su vez, un aumento en el tipo de interés reduce la inversión, lo que disminuye la demanda agregada en el mercado de bienes. Esta retroalimentación entre mercados es lo que hace al modelo $IS-LM$ rico en predicciones sobre política económica.

Formalizando con las funciones lineales adoptadas en este trabajo, la **curva IS** reducida queda:

$$Y = \frac{C_0 + I_0 + G}{1 - c} - \frac{b}{1 - c} \cdot r \quad (2.3)$$

donde $1/(1 - c)$ es el multiplicador keynesiano y la pendiente negativa $-b/(1 - c)$ refleja el efecto contractivo del tipo de interés sobre la inversión. La **curva LM** reducida es:

$$r = \frac{k}{h} \cdot Y - \frac{M_s}{h} \quad (2.4)$$

donde la pendiente positiva k/h indica que a mayor producción, mayor demanda de dinero y mayor tipo de interés de equilibrio. El **equilibrio IS-LM** (Y^*, r^*) resulta de resolver el sistema de las Ec. (2.3)–(2.4): es el punto de intersección que el editor calcula y visualiza, y que se valida analíticamente en el Cap. 5.

En su formulación lineal, el modelo IS-LM es un sistema de ecuaciones algebraicas que determina unívocamente los valores de equilibrio (Y^*, r^*) .

2.3 Modelo Solow

El modelo de Solow (Solow, 1956) analiza crecimiento económico de largo plazo. Este modelo explica cómo una economía crece en el largo plazo mediante la **acumulación de capital**, el crecimiento de la población y el progreso tecnológico. A diferencia de modelos más simples, el modelo de Solow incorpora rendimientos decrecientes del capital, lo que genera conclusiones importantes sobre la **convergencia** económica y el **estado estacionario**.

El modelo de Solow representa una economía cerrada que produce un único bien que puede utilizarse tanto para consumo como para inversión. Esta economía se caracteriza por los siguientes elementos fundamentales:

Variables Fundamentales

La producción agregada Y depende de tres factores: el stock de capital K , la fuerza laboral L y el nivel tecnológico A . El capital representa las máquinas, edificios e infraestructura que se utilizan en la producción. La fuerza laboral es el número de trabajadores empleados en la economía. La tecnología refleja el conocimiento y la eficiencia con que se combinan capital y trabajo para producir bienes.

Mediante una **función de producción Cobb-Douglas**:

$$Y = A \cdot K^\alpha \cdot L^{1-\alpha} \quad (2.5)$$

donde Y es la producción agregada, A el nivel tecnológico (productividad total de los factores), K el stock de capital, L la fuerza laboral y $\alpha \in (0, 1)$ la elasticidad del capital en la producción.

En términos per cápita, definimos:

$$y = Y/L \tag{2.6}$$

$$k = K/L \tag{2.7}$$

donde $y = Y/L$ es el producto por trabajador y $k = K/L$ el capital por trabajador.

Mecanismos del Modelo

El modelo opera a través de tres mecanismos principales. Primero, la inversión: una fracción s de la producción se ahorra e invierte en nuevo capital. Segundo, la **depreciación**: el capital existente se desgasta a una tasa δ cada período. Tercero, el **crecimiento poblacional**: la fuerza laboral crece a una tasa n constante. La interacción entre estos tres mecanismos determina la evolución del capital por trabajador y, por tanto, del producto por trabajador en el tiempo.

El modelo asume que los mercados funcionan competitivamente, que no hay gobierno ni comercio exterior, y que toda la producción que no se consume se invierte automáticamente. Estas simplificaciones permiten concentrarse en los determinantes fundamentales del crecimiento de largo plazo.

Ambos modelos son de naturaleza dinámica. La ecuación de acumulación de capital que rige la dinámica del modelo de Solow es:

$$\dot{K} = s \cdot Y - \delta \cdot K \tag{2.8}$$

donde $s \in (0,1)$ es la tasa de ahorro, Y la producción, δ la tasa de depreciación del capital y K el stock de capital. Esta sección se limita a la formulación teórica del modelo; su implementación completa como componentes Modelica se desarrolla en el Capítulo 3, y su simulación numérica en OpenModelica incluida la validación del estado estacionario y la convergencia se presenta en el Capítulo 5.

2.4 Modelica como entorno de modelado orientado a objetos

Modelica (Modelica Association, 2025) es un lenguaje de modelado **orientado a objetos** para sistemas físicos heterogéneos, enmarcado en el paradigma del modelado y la simulación basados en ecuaciones (Urquía Moraleda and Martín Villalba, 2014), desarrollado colaborativamente por la Modelica Association, un consorcio internacional de investigadores, desarrolladores de herramientas y usuarios industriales, desde su concepción inicial en 1996. La especificación del lenguaje, actualmente en su versión 3.6 (con la versión 3.7 en desarrollo), define rigurosamente la sintaxis mediante gramática BNF extendida, la semántica mediante reglas de traducción a sistemas de ecuaciones planas, y las bibliotecas estándar que proporcionan componentes reutilizables para múltiples dominios físicos (Fritzson, 2014).

El diseño de Modelica se fundamenta en varios principios arquitectónicos distintivos.

Primero, el enfoque declarativo basado en ecuaciones: los modelos se describen mediante ecuaciones matemáticas que expresan relaciones invariantes entre variables, sin especificar algoritmos procedurales de cálculo. El compilador determina automáticamente cómo resolver las ecuaciones mediante análisis simbólico.

Segundo, la orientación a objetos con herencia, encapsulación y polimorfismo: los componentes se organizan jerárquicamente en clases que pueden heredar comportamiento de clases base, encapsular detalles de implementación exponiendo únicamente interfaces públicas, y ser sustituidos polimórficamente cuando satisfacen interfaces compatibles.

Tercero, las conexiones acausales: los componentes se interconectan a través de conectores tipados. En dominios físicos, estos conectores no distinguen *a priori* entre entradas y salidas, sino entre **variables across** (de equilibrio o potencial, como la tensión entre dos puntos de un circuito eléctrico) y **variables through** (de flujo conservado, como la corriente eléctrica). En cambio, en los modelos económicos IS-LM y Solow, sí existe causalidad explícita entre variables de entrada y salida, definida por las ecuaciones estructurales del modelo. Por analogía con esa distinción física, las variables de equilibrio (Y , r en IS-LM; k , y en Solow) desempeñan aquí el papel de variables *across*: son las incógnitas que el sistema resuelve simultáneamente. Las variables de flujo (C , I , G , M en IS-LM; $s \cdot y$ el ahorro y $\delta \cdot k$ la depreciación, ambos per cápita, en Solow (versión per cápita de la Ec. (2.8), desarrollada en el Capítulo 3) desempeñan el papel de variables *through*: son magnitudes que distintos componentes producen y consumen.

Cuarto, la separación entre estructura y comportamiento: la topología del modelo indica

qué componentes existen y cómo se conectan, se especifica independientemente del comportamiento de cada componente (ecuaciones que gobiernan sus variables), facilitando la construcción modular de modelos complejos mediante composición de componentes más simples.

Los principales entornos de desarrollo Modelica disponibles incluyen OpenModelica (OpenModelica, 2026), Dymola (Dassault Systèmes, 2024), SystemModeler, MapleSim y SimulationX. OpenModelica, desarrollado por el Open Source Modelica Consortium liderado por la Universidad de Linköping en Suecia, constituye la implementación de código abierto más comprehensiva y madura. Incluye un compilador completo que traduce modelos Modelica a código C ejecutable, un entorno de simulación con múltiples solvers numéricos (DASSL, IDA, Radau), el editor gráfico OMEdit que permite modelado visual similar al desarrollado en este trabajo, herramientas de scripting y automatización mediante el lenguaje de scripting de OpenModelica, debuggers simbólicos que permiten inspeccionar ecuaciones y variables durante simulación, y optimizadores que pueden realizar ajuste de parámetros y diseño óptimo.

Dymola, producto comercial de Dassault Systèmes originalmente desarrollado por Dynasim AB, ofrece capacidades avanzadas particularmente valiosas para aplicaciones industriales: simulación en tiempo real con generación de código embebido para hardware específico, co-simulación mediante el estándar FMI (Functional Mockup Interface) permitiendo acoplar modelos Modelica con modelos desarrollados en otras herramientas, optimización multi-objetivo con algoritmos genéticos y métodos de gradiente, y análisis de sensibilidad e incertidumbre mediante técnicas de Monte Carlo y superficies de respuesta.

La Biblioteca Estándar de Modelica (Modelica Association, 2026) (MSL - **Modelica Standard Library**) proporciona más de 1600 componentes predefinidos organizados en paquetes por dominios físicos. El paquete Modelica.Electrical contiene modelos de componentes eléctricos básicos (resistencias, capacitores, inductores, fuentes de voltaje y corriente), semiconductores (diodos, transistores), máquinas eléctricas (motores DC, AC síncronos y asíncronos), convertidores de potencia y sensores. El paquete Modelica.Mechanics.Rotational modela sistemas mecánicos rotacionales con inercias, resortes torsionales, amortiguadores, embragues y frenos. Modelica.Thermal.HeatTransfer describe transferencia de calor por conducción, convección y radiación. Modelica.Fluid (Casella et al., 2005) implementa sistemas de fluidos incompresibles y compresibles con tuberías, válvulas, bombas e intercambiadores de calor. Cada componente de la MSL se define mediante: ecuaciones que describen su comportamiento físico derivadas de primeros principios (leyes de conservación, constitutivas), conectores que especifican las interfaces mediante las cuales el componente intercambia energía, información o materia con otros componentes, parámetros que permiten la configuración del componente para instancias específicas,

y anotaciones que proporcionan metadatos como documentación textual, representación gráfica (icono para diagramas), y layout visual. Esta estructura consistente facilita enormemente la composición de modelos complejos mediante instanciación y conexión de componentes de bibliotecas.

2.5 Herramientas existentes para la simulación de modelos IS-LM y Solow

Existen diversas herramientas para la simulación de los modelos IS-LM y Solow, que abarcan desde aplicaciones web interactivas hasta **lenguajes de propósito general** y entornos especializados.

La sección describe las herramientas web e interactivas (Sec. 2.5.1), los lenguajes de programación de uso general empleados en macroeconomía (Sec. 2.5.2), el software especializado (Sec. 2.5.3) y las implementaciones existentes en Modelica (Sec. 2.5.4).

2.5.1. Herramientas web e interactivas

Las herramientas web permiten explorar los modelos IS-LM y Solow sin instalación de software. Destacan simuladores online como los de la Universidad de Columbia y el MIT, que permiten ajustar parámetros en tiempo real y visualizar los efectos sobre el equilibrio macroeconómico. Su limitación principal es la falta de personalización y la imposibilidad de exportar modelos para su uso en entornos de simulación avanzados.

Wolfram Demonstrations Project ofrece una de las plataformas más completas para visualización dinámica de modelos económicos. Sus *Demonstrations* permiten ajustar parámetros en tiempo real y observar el desplazamiento de las curvas IS y LM, aunque no exportan modelos ni admiten datos reales. La Figura 2.1 muestra una simulación del modelo de Solow en Wolfram Demonstrations Project (Wolfram, 2026b).

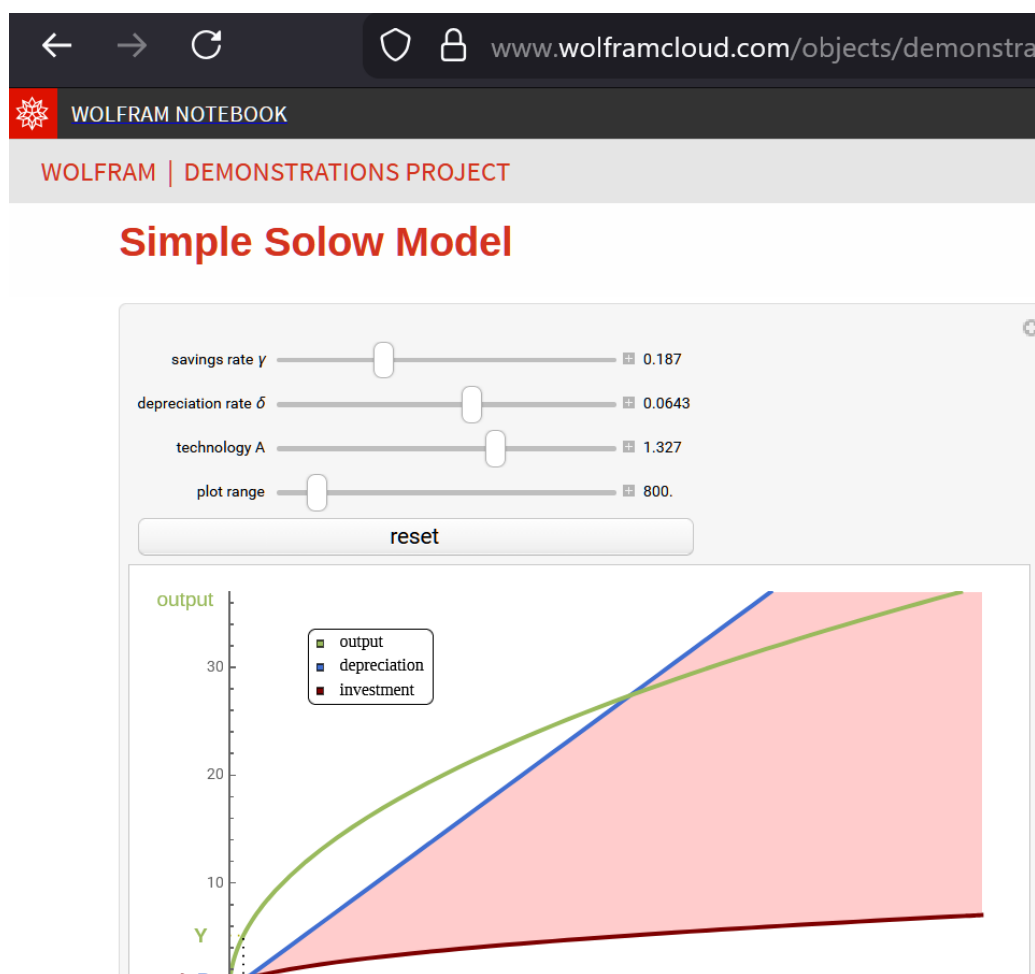


Figura 2.1: Bibliotecas de Wolfram Demonstration Project: simulación del modelo de Solow.

DIY Macroeconomic Model Simulation (Köhler, 2025b) es una plataforma educativa de acceso libre desarrollada por Karsten Köhler (Universidad de Leeds) y Franz Prante (Universidad Técnica de Chemnitz) que proporciona **implementaciones de referencia en R y Python** de una colección de modelos macroeconómicos. El sitio incluye, entre otros, el modelo IS-LM (Köhler, 2025a) (cap. 4) y el modelo de crecimiento de Solow (Köhler and Prante, 2025) (cap. 21), con código fuente completo, reproducible y de acceso abierto (licencia CC BY-NC 4.0). Cada implementación resuelve numéricamente las ecuaciones de equilibrio (multiplicador fiscal, política monetaria, estado estacionario de Solow) y permite el análisis de sensibilidad ante variaciones de parámetros mediante los propios entornos de cómputo de R y Python, sin necesidad de software adicional. Cubre, por tanto, las mismas formulaciones IS-LM y Solow adoptadas en este TFM y constituye una referencia de contraste para verificar que el código Modelica generado por el editor de este trabajo produce valores de equilibrio coherentes (Köhler, 2025a; Köhler and Prante, 2025). No obstante, la plataforma no permite construir modelos gráficamente ni exportarlos a entornos de simulación formales como Modelica u OMEdit, lo que delimita claramente su alcance respecto al editor aquí desarrollado. Las Figuras 2.2 y 2.3 ilustran

la estructura y las capacidades de la plataforma.

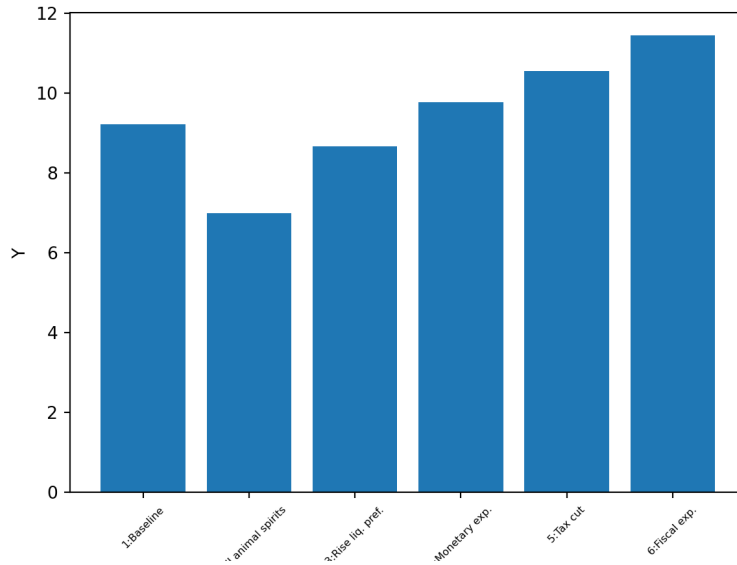


Figura 2.2: Diagrama causal del modelo IS-LM en macrosimulation.org: relaciones entre variables endógenas (Y, r, C, I, M_d) y exógenas (G_0, M_0, T_0, i_0, m_0). Fuente: Köhler (2025a), CC BY-NC 4.0.

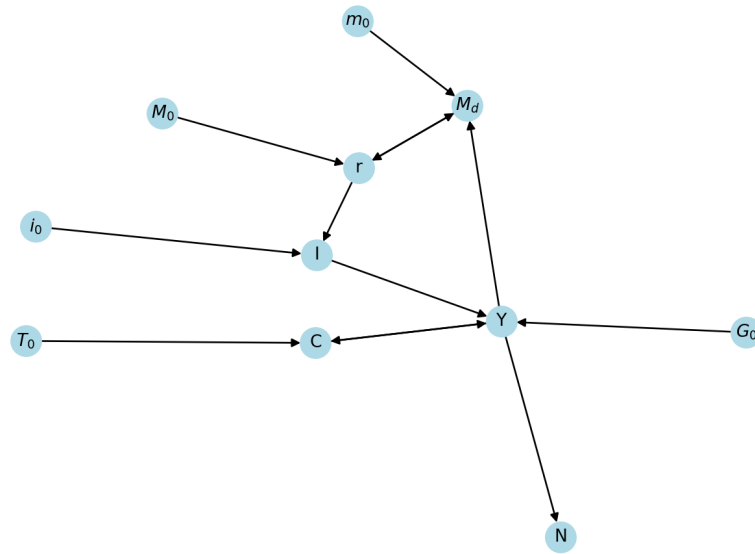


Figura 2.3: Producto de equilibrio Y bajo seis escenarios en macrosimulation.org: base, caída de expectativas, preferencia por liquidez, expansión monetaria, reducción fiscal y expansión fiscal. Fuente: Köhler (2025a), CC BY-NC 4.0.

Frente a la exploración sin instalación de las herramientas web, el siguiente apartado describe software especializado con capacidades de modelado más avanzadas.

2.5.2. Software de cálculo y lenguajes de programación

Microsoft Excel (con Solver): es la herramienta más accesible para la simulación estática de modelos IS-LM. Se pueden construir plantillas donde los parámetros son celdas editables y el equilibrio se calcula mediante Solver. Su principal limitación es la dificultad para modelar dinámicas temporales y la propensión a errores en modelos complejos.

MATLAB: ampliamente utilizado en entornos universitarios avanzados para simulaciones numéricas de modelos macroeconómicos. Ofrece potentes herramientas de análisis y visualización, pero requiere licencia comercial.

Python (librerías gráficas): se utiliza frecuentemente para crear simulaciones interactivas del modelo IS-LM mediante librerías como `matplotlib` (Hunter, 2007), `ipywidgets` (Kluyver et al., 2016) o `streamlit`. Es flexible y permite integrar datos reales, pero requiere programación y no genera modelos en formatos estándar de simulación dinámica.

R (Shiny Apps): existen numerosas aplicaciones desarrolladas por economistas en Shiny que permiten simular modelos IS-LM y Solow de forma interactiva en el navegador. Al igual que Python, requiere conocimientos de programación.

2.5.3. Software especializado en macroeconomía

Dynare: software gratuito que corre sobre MATLAB u Octave, diseñado específicamente para resolver modelos de equilibrio general dinámico y estocástico (DSGE). Es el estándar en macroeconomía cuantitativa avanzada, pero no dispone de editor visual de modelos y tiene una curva de aprendizaje elevada.

EViews: orientado a series temporales y modelos econométricos, ampliamente utilizado en análisis de política económica (IHS Markit, 2024).



Buy Now

[Home](#) [Products & Pricing](#) [User Community](#) [Learning Resources](#) [About EViews](#)

EViews 9.5 Overview: Presentation Output



EViews 9.5

[EViews 9.5 Home](#)
[Overview](#)
[Feature List](#)
[What's New in EViews 9.5](#)
[What's New Videos](#)
[System Requirements](#)

Part 4: Presentation Quality Output

EViews supports a wide range of basic graph types including line graphs, bar graphs, filled area graphs, pie charts, scatter diagrams, mixed line-bar graphs, high-low graphs, scatterplots, and boxplots. Any number of graphs can be combined in a single graph for presentation.



Figura 2.4: Interfaz de EViews 9.5: múltiples gráficos de series temporales. Estándar en econometría aplicada, pero sin editor visual de modelos estructurales (IS-LM, Solow).

Fuente: IHS Markit (2024), uso ilustrativo.

A continuación la Tabla 2.1 resume las más relevantes comparando sus ventajas y limitaciones frente al editor desarrollado en este trabajo.

Tabla 2.1: Comparativa de herramientas para la simulación de modelos IS-LM y Solow.

Categoría	Herramienta	Ventajas	Limitaciones
Web interactiva	Wolfram / macrosimulation.org	Sin instalación, visualización inmediata, pedagógica	No exporta modelos, sin datos reales, no extensible
Hoja de cálculo	Excel + Solver	Muy accesible, sin curva de aprendizaje técnica	Propenso a errores, difícil de mantener, no dinámico
Lenguaje general	Python/R/MATLAB	Flexible, con datos reales, reproducible	Requiere programación, sin editor visual
Software especializado	Dynare / EViews	Orientado a macroeconomía, calibración avanzada	Sin editor visual; Dynare requiere MATLAB/Octave, EViews licencia propia
Editor Modelica (este TFM)	Java + OpenModelica	Editor visual, datos reales, exportación .mo estándar	Modelos IS-LM y Solow (versión actual)

2.5.4. Implementación en Modelica y librerías específicas

Para los modelos IS-LM y Solow no existe una librería Modelica estándar. La **Business-Simulation Library** (GitHub (Modelica-3rdparty), 2025), aunque orientada a microeconomía empresarial, incluye bloques de dinámica de sistemas compatibles con modelos de crecimiento tipo Solow.

El proyecto más relevante es **MOSES** (*Modeling of Sustainable Economic Systems*) (Fritzson et al., 2025), desarrollado en el marco de OpenModelica, que implementa componentes para flujos de recursos naturales, stocks de capital natural y emisiones (Rockström et al., 2009). MOSES demuestra que Modelica es técnicamente apto para modelar sistemas económico-ecológicos, aunque no incluye modelos IS-LM ni Solow estándar ni editor gráfico visual, lo que justifica el presente trabajo.

Zimmer and Schlabe (2012) presentan un enfoque para modelado de sistemas económicos dinámicos usando Modelica, y Fritzson (2014) documenta la versatilidad del lenguaje para modelado en múltiples dominios. Estos trabajos validan la viabilidad técnica del enfoque adoptado en este TFM y demuestran que Modelica, aunque no sea el entorno habitual de los economistas, ofrece ventajas sustanciales para el modelado formal de sistemas dinámicos macroeconómicos.

En el ámbito del software especializado destacan herramientas directamente relacionadas con Modelica y la simulación económica:

Wolfram System Modeler permite simular modelos IS-LM y Solow mediante bibliotecas de componentes predefinidos. Ofrece integración con Mathematica para análisis numérico avanzado, aunque su coste de licencia limita su accesibilidad en entornos académicos (Wolfram, 2026a).

La relevancia de estas herramientas para el presente trabajo es doble: por un lado, justifican el diseño modular y extensible del editor, concebido para incorporar nuevos dominios sin reestructurar el código existente; por otro, señalan la dirección en que la herramienta puede evolucionar en trabajos futuros, alineándose con la demanda creciente de marcos que integren bienestar social y sostenibilidad ambiental, como la Economía Circular y el modelo Doughnut, cuyo estado del arte se revisa a continuación.

2.6 Economía Circular y modelo Doughnut: estado del arte

Los modelos macroeconómicos clásicos, como el IS-LM y el de Solow, asumen economías lineales: los recursos se extraen, se transforman en producción y se desechan. Esta linealidad ignora los límites físicos del planeta (Meadows et al., 1972) y los costes ecológicos de la actividad económica. En las últimas décadas han surgido marcos teóricos alternativos que intentan integrar la sostenibilidad en el análisis macroeconómico (Jackson, 2009), siendo los más relevantes para este trabajo la Economía Circular y el modelo Doughnut de Kate Raworth (Raworth, 2017). La sección describe ambos marcos, resume sus variables clave para la extensión del editor y justifica su idoneidad para Modelica mediante el consorcio MOSES.

Economía Circular. La Economía Circular (EC) propone sustituir el modelo lineal extraer–producir–desechar por sistemas cerrados donde los residuos de un proceso se convierten en recursos de otro, tal como ilustra la Figura 2.5. Sus variables clave incluyen la tasa de circularidad ρ (fracción de materiales que reentran al ciclo productivo), la tasa de reciclaje R , la vida útil media del capital $\bar{K}$, y las externalidades medioambientales asociadas a la extracción de recursos vírgenes. En el contexto del modelo de Solow, la circularidad modifica la función de depreciación efectiva: si una fracción ρ del capital depreciado se reutiliza, la depreciación neta se reduce a $\delta \cdot (1 - \rho)$, lo que altera la trayectoria hacia el estado estacionario. En el modelo IS-LM, la existencia de mercados de segunda vida y la reducción de costes de materias primas afectan a la **función de consumo** e inversión agregadas.

El ciclo de la Economía Circular

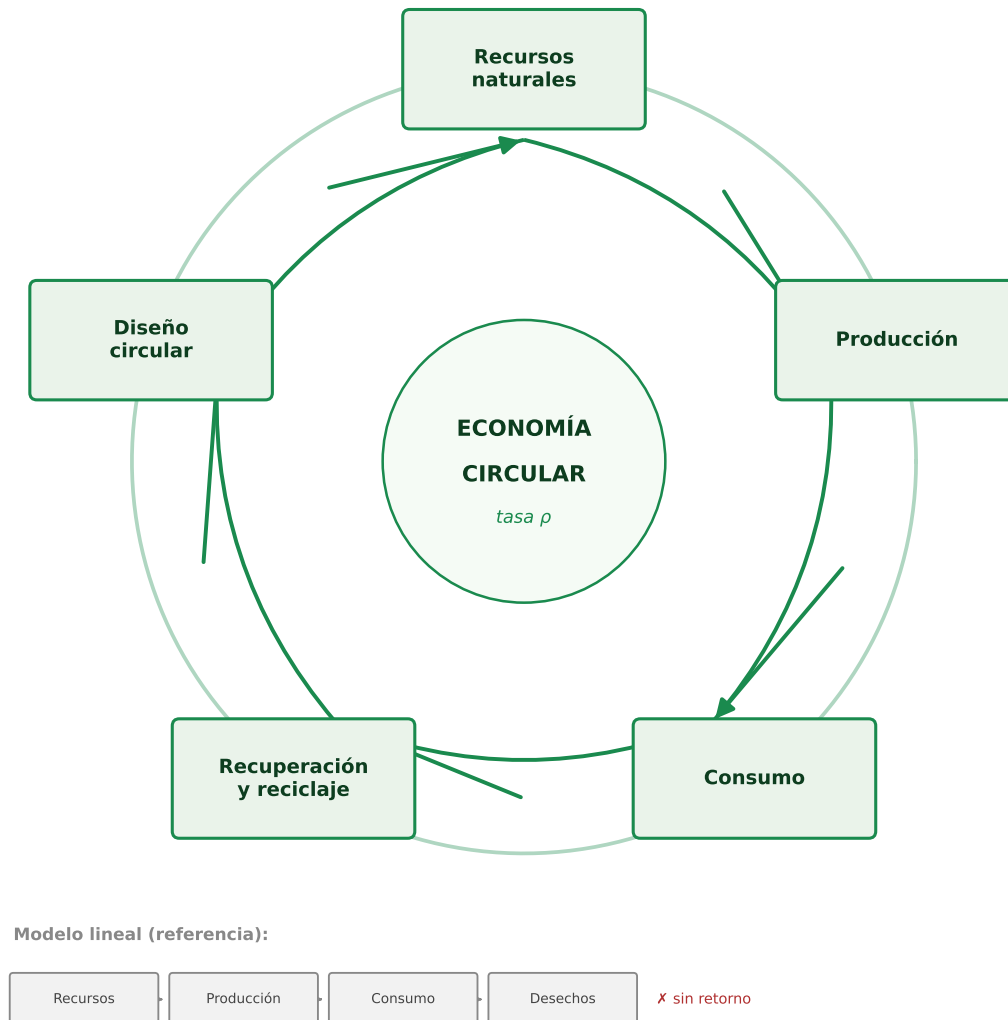


Figura 2.5: El ciclo de la Economía Circular: los materiales reentran al sistema en lugar de descartarse, frente al modelo lineal de referencia (inserto inferior). La tasa ρ cuantifica esta recirculación; su efecto concreto sobre el modelo de Solow se detalla como línea de trabajo futuro en el Capítulo 7.

Modelo Doughnut (Doughnut Economics Action Lab (DEAL), 2026; Raworth, 2017). El modelo Doughnut define un espacio económico seguro y justo delimitado por dos anillos: un *suelo social* (doce dimensiones de bienestar humano derivadas de los Objetivos de Desarrollo Sostenible de la ONU: agua, alimentación, salud, educación, ingresos y trabajo, paz y justicia, voz política, equidad social, igualdad de género, vivienda, redes y energía) y un *techo ecológico* (nueve **límites planetarios** identificados por Rockström

et al. (2009): cambio climático, acidificación oceánica, contaminación química, exceso de nitrógeno y fósforo, uso de agua dulce, cambio de uso del suelo, pérdida de biodiversidad, contaminación del aire y agotamiento de la capa de ozono). Las economías deben operar dentro de ese anillo intermedio: por encima del suelo social pero por debajo del techo ecológico. Desde el punto de vista de la simulación, el modelo Doughnut introduce dos tipos de restricciones sobre las variables macroeconómicas: restricciones inferiores sobre variables de bienestar (que no pueden caer por debajo de un umbral mínimo) y restricciones superiores sobre variables de presión ambiental (que no pueden superar un límite máximo). La Tabla 2.2 concreta esta idea con las tres variables del modelo Doughnut más relevantes para su integración con los modelos IS-LM y Solow:

Tabla 2.2: Variables del modelo Doughnut relevantes para la extensión de IS-LM y Solow.

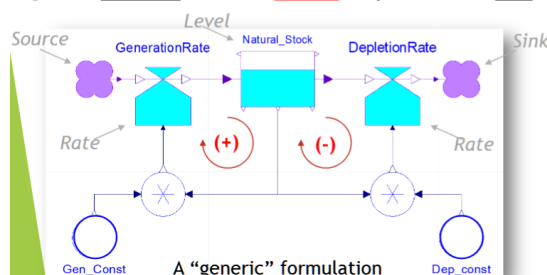
Dimensión	Variable (modelo)	Efecto
Suelo social: educación	Gasto educación G^e IS-LM, curva IS	Desplaza IS: mayor G^e eleva Y^* vía multiplicador
Suelo social: equidad	Índice de Gini γ IS-LM, consumo	Modifica propensión marginal $c = c(\gamma)$
Techo ecológico: agua	Uso agua dulce W Solow, producción	Penaliza el factor tecnológico A cuando el consumo de agua supera el límite planetario $W_{\text{máx}}$, reduciendo la producción por trabajador

Relevancia para el modelado en Modelica. La librería **MOSES** (*Modeling of Sustainable Economic Systems*) (Fritzson et al., 2025), desarrollada en el marco de OpenModelica, es el antecedente más directo de la integración de límites ecológicos en Modelica. MOSES implementa componentes para flujos de recursos naturales, stocks de capital natural y emisiones, demostrando que Modelica es técnicamente apto para modelar sistemas económico-ecológicos. Sin embargo, MOSES no incluye modelos macroeconómicos estándar (IS-LM, Solow) ni un editor gráfico visual orientado a usuarios no técnicos, lo que justifica la extensión propuesta en este trabajo.

Simple exponential generation/depletion model

- ▶ Example with two loops: 1 negative, 1 positive
- ▶ **Levels and Rates**
- ▶ **Sources and Sinks**
 - ▶ Provided for optical purposes only (System Dynamics modelers are used to them)
 - ▶ These models do not represent equations.

E.g.: The Modelica-based SD Library + OpenModelica Tool



E.g. The Stella SD-based Tool

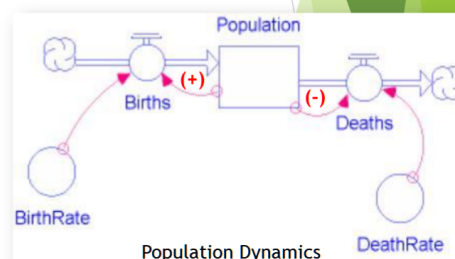


Figura 2.6: Proyecto MOSES en OpenModelica: World Models para transiciones sostenibles.

2.6.1. Investigaciones previas

Además de los trabajos de Zimmer and Schlabe (2012) y Fritzson (2014) ya citados, existen recursos educativos como DrControl y OMWebbook que utilizan Modelica en ciencias sociales, aunque sin orientación macroeconómica ni editor gráfico. El presente editor se distingue de todos ellos por combinar tres capacidades en un único entorno: **editor gráfico visual** de arrastrar y soltar, **integración de datos reales** del Banco Mundial y **exportación estándar a Modelica (.mo)** compatible con OpenModelica.

2.7 Conclusiones

La revisión realizada en este capítulo ha establecido los fundamentos teóricos de los modelos IS-LM y Solow, situado las herramientas existentes en su contexto y analizado los marcos de Economía Circular y Doughnut como extensiones relevantes lo cual permite extraer cuatro conclusiones que orientan el diseño del editor desarrollado en este TFM:

Vacío de herramienta. Las soluciones existentes cubren o bien la simulación técnica avanzada (Dymola, Wolfram System Modeler, MATLAB/Dynare) o bien la interactividad pedagógica superficial (simuladores web, hojas Excel), pero ninguna combina un **editor gráfico visual** con **exportación estándar a Modelica** y **datos reales** integrados. Este es el hueco que el presente trabajo ocupa.

Madurez de Modelica para economía. Los trabajos de Zimmer and Schlabe (2012) y

Fritzson (2014) confirman que el paradigma acausal de Modelica es técnicamente viable para modelar sistemas económicos dinámicos. La librería MOSES (Fritzson et al., 2025) demuestra que la comunidad ya explora esta dirección.

Demanda de extensiones sostenibles. El creciente interés académico e institucional en la Economía Circular (Ellen MacArthur Foundation, 2013) y el modelo Doughnut (Raworth, 2017) justifica diseñar el editor con una arquitectura extensible que permita incorporar estos marcos sin reescribir el núcleo del sistema.

Relevancia pedagógica. La ausencia de herramientas abiertas, visuales y en español para enseñar macroeconomía con simulación refuerza la orientación fundamentalmente pedagógica de este TFM, orientación que se detalla en el Resumen y en los objetivos del Cap. 1.

El siguiente capítulo aborda la traducción de ambos modelos al lenguaje Modelica: se presentan los componentes implementados, las ecuaciones generadas y las decisiones de diseño que han guiado el desarrollo del editor.

Capítulo 3

Diseño e Implementación de los Modelos IS-LM y Solow

3.1 Introducción

Este capítulo describe el proceso completo de diseño e implementación de los modelos económicos IS-LM y de Solow en el lenguaje Modelica. La traducción de modelos económicos tradicionales a un lenguaje de modelado orientado a ecuaciones requiere decisiones cuidadosas sobre la estructura de componentes, las interfaces entre ellos y la representación de las relaciones económicas.

Los objetivos específicos de este capítulo son: (1) presentar el diseño conceptual de ambos modelos identificando componentes, variables, relaciones, hipótesis y simplificaciones; (2) describir la implementación concreta en código Modelica de cada **componente**; y (3) documentar las decisiones de diseño tomadas y las alternativas consideradas.

El capítulo se estructura siguiendo el flujo de cada modelo, desde el diseño teórico hasta la implementación.

Para el modelo IS-LM: la Sec. 3.2 describe sus hipótesis y simplificaciones; la Sec. 3.3 desarrolla su formulación matemática completa (**mercado de bienes, mercado de dinero y equilibrio**); y la Sec. 3.4 presenta su implementación en Modelica.

Para el modelo de Solow: la Sec. 3.5 describe su diseño y la Sec. 3.6 detalla su implementación en Modelica.

Finalmente, la Sec. 3.7 presenta la extensión de economía circular del modelo de Solow.

Esta implementación constituye el núcleo funcional del editor desarrollado: proporciona la biblioteca de componentes económicos que los usuarios podrán conectar gráficamente tal como se describe en los objetivos del Capítulo 1.

Tanto para el modelo IS-LM como para el modelo de Solow se seguirá la misma metodología de diseño en cuatro fases:

1. Hipótesis y simplificaciones del modelo.
2. Descripción y formulación matemática completa.
3. Implementación en Modelica de los componentes.
4. Validación y verificación (desarrollada en el Capítulo 5).

3.2 IS-LM: hipótesis y simplificaciones

Antes de construir el código Modelica es necesario delimitar qué variante del modelo IS-LM se va a implementar: con o sin sector exterior, con o sin impuestos, con o sin ecuaciones lineales, etc. En este trabajo se ha optado por la versión más simple, que es la más habitual en los manuales de macroeconomía introductoria y facilita la validación analítica de los resultados. A continuación se presentan las variables y parámetros del modelo antes de enunciar las hipótesis.

Tabla 3.1: Variables y parámetros del modelo IS-LM.

Símbolo	Valor típico	Unidades	Significado
Y	800 – 1 200	Mill. €	Producto nacional / PIB
r	0.03 – 0.08	Fracción/año	Tipo de interés de equilibrio
C_0	100 – 300	Mill. €	Consumo autónomo
c	0.60 – 0.90	Adimensional	Propensión marginal a consumir
I_0	200 – 400	Mill. €	Inversión autónoma ($r = 0$)
b	20 – 100	Mill. €/punto %	Sensibilidad inversión al tipo de interés
G	100 – 300	Mill. €	Gasto público exógeno
M_s	400 – 800	Mill. €	Oferta monetaria
k	0.10 – 0.40	Adimensional	Sensibilidad M_d al ingreso
h	10 – 200	Mill. €/punto %	Sensibilidad M_d al tipo de interés
α	0.50 – 2.00	1/año	Velocidad de ajuste mercado de bienes
β	0.10 – 1.00	1/año	Velocidad de ajuste mercado monetario

La sección se estructura en dos apartados: las hipótesis sobre las que se construye el modelo (Sec. 3.2.1) y las simplificaciones adoptadas en la implementación (Sec. 3.2.2).

3.2.1. Hipótesis del modelo IS-LM

El modelo IS-LM básico se sustenta en cuatro hipótesis principales:

Primera hipótesis: precios rígidos en el corto plazo. El nivel de precios P permanece constante durante el período de análisis. Los ajustes ocurren mediante cambios en cantidades (producción, empleo) más que en precios. Esta hipótesis es razonable para períodos cortos, pero no para análisis de largo plazo.

Segunda hipótesis: economía cerrada sin comercio internacional. No existen exportaciones, importaciones ni flujos de capital con el resto del mundo. Esta simplificación permite concentrarse en los mecanismos internos, aunque limita la aplicabilidad a economías muy abiertas.

Tercera hipótesis: relaciones de comportamiento lineales. El consumo, la inversión y la **demanda de dinero** se modelan mediante relaciones lineales, lo que facilita el análisis algebraico y gráfico, aunque sacrifica algo de realismo.

Cuarta hipótesis: expectativas estáticas. Los agentes no anticipan cambios futuros cuando toman decisiones presentes, lo que simplifica el modelo pero ignora la dimensión intertemporal del comportamiento económico.

Las funciones lineales adoptadas bajo la tercera hipótesis son las siguientes:

$$C = C_0 + c \cdot Y, \quad \text{con } 0 < c < 1 \quad (3.1)$$

$$I = I_0 - b \cdot r, \quad \text{con } b > 0 \quad (3.2)$$

$$G = G_0 \quad (\text{exógeno}) \quad (3.3)$$

$$M_d = k \cdot Y - h \cdot r, \quad \text{con } k, h > 0 \quad (3.4)$$

donde $C_0 > 0$ es el consumo autónomo, $c \in (0, 1)$ la propensión marginal a consumir, $I_0 > 0$ la inversión autónoma, $b > 0$ la sensibilidad de la inversión al tipo de interés r , G_0 el gasto público exógeno (fijado por política fiscal), M_d la demanda de dinero, $k > 0$ la sensibilidad de la demanda de dinero a la renta y $h > 0$ la sensibilidad de la demanda de dinero al tipo de interés.

Establecidas las cuatro hipótesis, el siguiente apartado delimita las simplificaciones que permiten mantener el modelo matemáticamente tratable.

3.2.2. Simplificaciones del modelo IS-LM

Además de las hipótesis anteriores, el modelo básico incorpora las siguientes simplificaciones:

Sin impuestos ni sector bancario. El modelo no incluye impuestos T , lo que simplifica las ecuaciones pero omite un instrumento importante de **política fiscal**. Tampoco considera el sector bancario ni el mecanismo de creación endógena de dinero: se asume que el banco central controla directamente la **oferta monetaria** M_s .

Sin formación de capital. La inversión I se incorpora a la demanda agregada como gasto corriente, pero no se acumula en el stock de capital ni afecta a la capacidad productiva futura. Esta simplificación es apropiada para el análisis de corto plazo, pero es inconsistente con el modelo de Solow, que se analiza en la Sec. 3.5.

Sin inflación ni expectativas inflacionarias. El nivel general de precios P se asume constante, por lo que los tipos de interés nominal y real son idénticos en este marco. Esta limitación es importante porque en economías con inflación significativa la distinción entre ambos tipos es crucial para entender el comportamiento de la inversión.

Estas simplificaciones son deliberadas: permiten centrarse en los mecanismos esenciales del equilibrio simultáneo en dos mercados. Las extensiones posibles (economía abierta, impuestos proporcionales, expectativas racionales) se recogen en la sección de trabajos futuros del Capítulo 7.

3.3 IS-LM: descripción y formulación matemática

Bajo las hipótesis anteriores, el modelo IS-LM queda definido por siete componentes que representan las ecuaciones fundamentales del equilibrio macroeconómico de corto plazo. Esta sección presenta la formulación matemática completa antes de pasar a su implementación en Modelica en la Sec. 3.4. La sección se estructura en tres apartados: las ecuaciones del mercado de bienes y la curva IS (Sec. 3.3.1), las ecuaciones del mercado monetario y la curva LM (Sec. 3.3.2) y el equilibrio simultáneo de ambos mercados (Sec. 3.3.3).

3.3.1. Ecuaciones del mercado de bienes (curva IS)

El mercado de bienes determina el nivel de producción Y dado el tipo de interés r . El equilibrio requiere que la producción sea igual a la demanda agregada DA :

$$DA = C + I + G \quad (3.5)$$

donde DA es la demanda agregada, C el consumo privado de los hogares, I la inversión empresarial y G el gasto público exógeno.

$$Y = C(Y) + I(r) + G \quad (3.6)$$

donde $C(Y)$ es la función de consumo de los hogares, $I(r)$ la función de inversión empresarial (decreciente en r) y G el gasto público exógeno.

Sustituyendo las funciones lineales Ecs. (3.1), (3.2) y (3.3) en la Ec. (3.6) se obtiene la ecuación reducida de la curva IS:

$$Y = (C_0 + I_0 + G)/(1 - c) - (b/(1 - c)) \cdot r \quad (3.7)$$

donde C_0 e I_0 son el consumo y la inversión autónomos, $c \in (0, 1)$ la propensión marginal a consumir y $b > 0$ la sensibilidad de la inversión al tipo de interés r . Esta expresión define la curva IS en el plano (Y, r) .

Dado que los modelos económicos rara vez alcanzan el equilibrio instantáneamente, la implementación dinámica en Modelica modela el ajuste gradual de la producción mediante una **ecuación diferencial** ordinaria:

$$\dot{Y} = \alpha \cdot (DA - Y) \quad (3.8)$$

donde $\alpha > 0$ es el parámetro de velocidad de ajuste del mercado de bienes. El sistema converge al equilibrio estático de la Ec. (3.6) cuando $\dot{Y} = 0$. El mercado de bienes queda así caracterizado por la curva IS. El siguiente apartado formula el mercado de dinero, cuyo equilibrio determina el tipo de interés.

3.3.2. Ecuaciones del mercado de dinero (curva LM)

El **mercado monetario** determina el tipo de interés r dado el nivel de producción Y . La demanda de dinero depende positivamente del ingreso (motivo transaccional) y negativamente del tipo de interés (coste de oportunidad de mantener dinero líquido):

$$M_d = k \cdot Y - h \cdot r \quad (3.9)$$

donde $k > 0$ es la sensibilidad de la demanda de dinero al nivel de renta Y y $h > 0$ su sensibilidad al tipo de interés r .

De forma análoga al mercado de bienes, la dinámica de ajuste del tipo de interés se modela con:

$$\dot{r} = \beta \cdot (M_d - M_s) \quad (3.10)$$

donde $\beta > 0$ es la velocidad de ajuste del tipo de interés y $\dot{r} = 0$ cuando se alcanza el equilibrio $M_s = M_d$.

Con las curvas IS y LM formuladas, el siguiente apartado calcula el punto de equilibrio simultáneo en ambos mercados.

3.3.3. Equilibrio general IS-LM

El equilibrio general del modelo requiere la satisfacción simultánea de las condiciones de ambos mercados. Las dos variables endógenas de equilibrio son Y^* y r^* :

$$Y^* = C(Y^*) + I(r^*) + G \quad (3.11)$$

$$M_s = k \cdot Y^* - h \cdot r^* \quad (3.12)$$

donde M_s es la oferta monetaria (fijada por política monetaria), k y h son los parámetros de sensibilidad de la Ec. (3.4), e Y^* y r^* los valores de producto y tipo de interés en el punto de equilibrio simultáneo IS-LM. Resolviendo analíticamente el sistema (3.11)–(3.12) con las funciones lineales Ecs. (3.1)–(3.4), el equilibrio queda determinado de forma cerrada:

$$Y^* = \frac{h(C_0 + I_0 + G) + b \cdot M_s}{h(1 - c) + b \cdot k} \quad (3.13)$$

$$r^* = \frac{k(C_0 + I_0 + G) - (1 - c) \cdot M_s}{h(1 - c) + b \cdot k} \quad (3.14)$$

donde C_0 , I_0 y G son los componentes autónomos de la demanda y M_s la oferta monetaria. Los parámetros clave son:

- $b > 0$: sensibilidad de la inversión al tipo de interés (Ec. (3.2): $I = I_0 - b \cdot r$). Mayor b implica curva IS más plana.
- $k > 0$: sensibilidad de la demanda de dinero a la renta (Ec. (3.4): $M_d = kY - hr$). Mayor k implica curva LM más empinada.
- $h > 0$: sensibilidad de la demanda de dinero al tipo de interés (Ec. (3.4): $M_d = kY - hr$). Mayor h implica curva LM más plana.

Puede verificarse que $\partial r^* / \partial M_s < 0$ (expansión monetaria reduce el tipo de interés) y $\partial Y^* / \partial G > 0$ (gasto público eleva el producto), ambos coherentes con la teoría estándar IS-LM (Blanchard, 2017).

Estas expresiones analíticas se usarán en el Capítulo 5 para validar numéricamente los resultados de la simulación en OpenModelica.

3.4 IS-LM: implementación en Modelica y componentes

Con la formulación matemática del apartado anterior bien establecida, en esta sección se presenta la implementación directa de cada ecuación en código Modelica. El modelo adopta una **arquitectura modular**: cada relación económica se encapsula en un componente independiente con conectores RealInput/RealOutput, y el ensamblaje final se realiza mediante instrucciones connect. La sección se estructura en tres apartados: la implementación del mercado de bienes (Sec. 3.4.1), la del mercado monetario (Sec. 3.4.2) y el ensamblaje completo del modelo (Sec. 3.4.3).

3.4.1. Implementación en Modelica del mercado de bienes

Los tres componentes de oferta del mercado de bienes se definen a continuación.

Componente FuncionConsumo

Implementa la Ec. (3.1). El parámetro $C_0 = 200$ representa el consumo autónomo y $c = 0.8$ la **propensión marginal a consumir**. Por cada euro adicional de ingreso, las familias destinan 0.80 € al consumo. El Código 3.1 lo implementa:

Código 3.1: Función de consumo en Modelica (FuncionConsumo).

```

1 model FuncionConsumo
2   parameter Real C0 = 200; // Consumo autónomo
3   parameter Real c = 0.8; // Propensión marginal a consumir (0 < c < 1)
4   RealInput Y_in; // Producción / ingreso (entrada)
5   RealOutput C_out; // Consumo agregado (salida)
6   equation
7     C_out = C0 + c * Y_in; // Ec. (3.1): C = C0 + c*Y
8 end FuncionConsumo;

```

Junto al consumo, la inversión es el segundo componente de la demanda privada.

Componente FuncionInversion

Implementa la Ec. (3.2). El parámetro $I_0 = 300$ es la inversión con tipo de interés nulo y $b = 50$ indica que cada punto porcentual adicional de r reduce la inversión en 50 unidades, capturando la pérdida de viabilidad de proyectos marginales cuando sube el coste de financiación. El Código 3.2 lo implementa:

Código 3.2: Función de inversión en Modelica (FuncionInversion).

```

1 model FuncionInversion
2   parameter Real I0 = 300; // Inversión autónoma
3   parameter Real b = 50; // Sensibilidad al tipo de interés (b > 0)
4   RealInput r_in; // Tipo de interés (entrada)
5   RealOutput I_out; // Inversión agregada (salida)
6   equation
7     I_out = I0 - b * r_in; // Ec. (3.2): I = I0 - b*r
8 end FuncionInversion;

```

El tercer y último componente de la demanda agregada es de naturaleza pública, no privada.

Componente GastoPublico

Implementa la Ec. (3.3). El **gasto público** G es una variable de política fiscal exógena que forma parte de la demanda agregada. Al ser un parámetro, el usuario puede simular expansiones o contracciones fiscales cambiando su valor directamente. El Código 3.3 lo implementa:

Código 3.3: Gasto público en Modelica (GastoPublico).

```

1 model GastoPublico
2   parameter Real G = 200; // Gasto público exógeno (decisión del gobierno)
3   RealOutput G_out; // Gasto público (salida)
4 equation
5   G_out = G; // Ec. (3.3):  $G = G_0$ 
6 end GastoPublico;
```

Componente MercadoBienes (IS)

Es el componente central del mercado de bienes. Recibe las tres entradas de demanda (C , I , G), calcula la demanda agregada DA y, mediante la ecuación diferencial Ec. (3.8), hace converger la producción Y hacia el equilibrio. El parámetro $\alpha = 1.0$ controla la velocidad de ese ajuste. El Código 3.4 lo implementa; el siguiente apartado describe el mercado monetario.

Código 3.4: Mercado de bienes IS en Modelica (MercadoBienes).

```

1 model MercadoBienes
2   parameter Real alpha = 1.0; // Velocidad de ajuste ( $\alpha > 0$ )
3   RealInput C_in; // Consumo (entrada)
4   RealInput I_in; // Inversión (entrada)
5   RealInput G_in; // Gasto público (entrada)
6   RealOutput Y_out(start=1000, fixed=true); // Producción de equilibrio (
  ↪ salida)
7   Real DA;
8 equation
9   DA = C_in + I_in + G_in; // Ec. (3.5):  $DA = C + I + G$ 
10  der(Y_out) = alpha * (DA - Y_out); // Ec. (3.8):  $dY/dt = \alpha * (DA - Y$ 
  ↪ )
11 end MercadoBienes;
```

La ecuación diferencial Ec. (3.8) merece una nota de interpretación: si la demanda supera la producción ($DA > Y$), las empresas aumentan su producción; si ocurre lo contrario, la reducen. El sistema converge numéricamente al equilibrio $Y^* = DA$ mediante integración

automática de OpenModelica (método Runge-Kutta por defecto).

3.4.2. Implementación en Modelica del mercado monetario

Los dos componentes que definen el mercado de dinero capturan la oferta y la demanda monetaria. El equilibrio dinámico del tipo de interés sigue la misma lógica de ajuste gradual que el mercado de bienes.

Componentes `OfertaMonetaria` y `DemandaDinero`

Implementan respectivamente las Ecs. (3.9) y (3.4): la oferta monetaria es exógena (fijada por el banco central) y la demanda depende positivamente del ingreso y negativamente del tipo de interés. El Código 3.5 los implementa:

Código 3.5: Oferta monetaria y demanda de dinero en Modelica.

```

1 model OfertaMonetaria
2   parameter Real Ms = 500; // Oferta monetaria (controlada por el banco
   ↪ central)
3   RealOutput Ms_out; // Oferta monetaria (salida)
4 equation
5   Ms_out = Ms; // Ec. (3.9):  $M_s = M_{s0}$ 
6 end OfertaMonetaria;
7
8 model DemandaDinero
9   parameter Real k = 0.2; // Sensibilidad Md al ingreso ( $k > 0$ )
10  parameter Real h = 10; // Sensibilidad Md al tipo de interés ( $h > 0$ )
11  RealInput Y_in; // Producción / ingreso (entrada)
12  RealInput r_in; // Tipo de interés (entrada)
13  RealOutput Md_out; // Demanda de dinero (salida)
14 equation
15  Md_out = k * Y_in - h * r_in; // Ec. (3.4):  $M_d = k*Y - h*r$ 
16 end DemandaDinero;

```

Con la oferta y la demanda de dinero ya definidas, el componente central del mercado monetario las recibe como entradas para calcular el tipo de interés de equilibrio.

Componente MercadoMonetario (LM)

El mercado monetario implementa la ecuación diferencial Ec. (3.10). Cuando hay exceso de demanda de dinero ($M_d > M_s$), el tipo de interés sube; cuando hay exceso de oferta, baja. El parámetro $\beta = 0,5$ regula la velocidad de este ajuste. El Código 3.6 lo implementa:

Código 3.6: Mercado monetario LM en Modelica (MercadoMonetario).

```

1 model MercadoMonetario
2   parameter Real beta = 0.5; // Velocidad de ajuste (beta > 0)
3   RealInput Md_in; // Demanda de dinero (entrada)
4   RealInput Ms_in; // Oferta monetaria (entrada)
5   RealOutput r_out(start=0.05, fixed=true); // Tipo de interés de
   ↪ equilibrio (salida)
6 equation
7   der(r_out) = beta * (Md_in - Ms_in); // Ec. (3.10): dr/dt = beta*(Md - Ms
   ↪ )
8 end MercadoMonetario;
```

Esta arquitectura modular permite simular fácilmente choques de política económica: un incremento del gasto público se implementa modificando únicamente el parámetro G del componente `GastoPublico`, y el sistema converge automáticamente al nuevo equilibrio siguiendo la dinámica de las Ecs. (3.8)–(3.10). Con los componentes de ambos mercados definidos, el siguiente apartado ensambla el modelo completo.

3.4.3. Ensamblaje y ecuaciones de equilibrio

El modelo completo conecta los siete componentes anteriores creando el sistema de retroalimentación entre mercados. Las instrucciones `connect` establecen el flujo de información entre componentes; OpenModelica resuelve automáticamente el sistema diferencial-algebraico resultante (Hairer et al., 1993).

Cada instrucción `connect` no es una metáfora de «cableado»: Modelica la traduce internamente en una **ecuación de igualdad** entre las dos variables conectadas. Por ejemplo, `connect(Consumo.C_out, IS.C_in)` genera la ecuación $IS.C_{in} = Consumo.C_{out}$ dentro del sistema que resuelve el simulador: exactamente el paso de sustituir la Ec. (3.1) en la Ec. (3.6) que se realizó algebraicamente a mano en la Sec. 3.3.1. Las nueve conexiones del modelo reconstruyen así, ecuación a ecuación, la misma sustitución que produjo la curva IS reducida (Ec. (3.7)) y el sistema LM (Ecs. (3.9)–(3.10)): la diferencia es que en Modelica cada paso queda como una ecuación independiente y verificable en lugar de una única expresión combinada, lo que permite a OpenModelica comprobar automáticamente

te que el sistema está bien determinado (mismo número de ecuaciones que de variables) antes de simular.

La elección de cada **connect** no es arbitraria: se deriva del **análisis de dependencia de variables** sobre el sistema de ecuaciones. Para cada variable se identificó la ecuación que la *define* (componente productor) y las que la *consumen* (componentes consumidores); cada par productor–consumidor se traduce en una instrucción **connect**. La Tabla 3.2 recoge este análisis; el diagrama causal de la Fig. 2.2 sirvió como validación cruzada.

Tabla 3.2: Análisis de dependencia de variables IS-LM: de las ecuaciones del modelo a las instrucciones **connect** de Modelica.

Var.	Produce	Consume	Instrucción connect
C	Consumo $C = C_0 + c \cdot Y$	IS $Y = C + I + G$	<code>connect(Consumo.C_out, IS.C_in)</code>
I	Inversion $I = I_0 - b \cdot r$	IS $Y = C + I + G$	<code>connect(Inversion.I_out, IS.I_in)</code>
G	Gasto $G = G_0$	IS $Y = C + I + G$	<code>connect(Gasto.G_out, IS.G_in)</code>
Y	IS $\dot{Y} = \alpha(C + I + G - Y)$	Consumo $C = C_0 + c \cdot Y$	<code>connect(IS.Y_out, Consumo.Y_in)</code>
Y	IS $\dot{Y} = \alpha(\dots)$	DemandaDin $M_d = kY - hr$	<code>connect(IS.Y_out, DemandaDin.Y_in)</code>
M_d	DemandaDin $M_d = kY - hr$	LM $\dot{r} = \beta(M_s - M_d)$	<code>connect(DemandaDin.Md_out, LM.Md_in)</code>
M_s	OfertaMon $M_s = M_{s0}$	LM $\dot{r} = \beta(M_s - M_d)$	<code>connect(OfertaMon.Ms_out, LM.Ms_in)</code>
r	LM $\dot{r} = \beta(M_s - M_d)$	Inversion $I = I_0 - b \cdot r$	<code>connect(LM.r_out, Inversion.r_in)</code>
r	LM $\dot{r} = \beta(\dots)$	DemandaDin $M_d = kY - hr$	<code>connect(LM.r_out, DemandaDin.r_in)</code>

A partir de este análisis, el Código 3.7 ensambla los siete componentes y establece las nueve conexiones causales que materializan el modelo IS-LM completo:

Código 3.7: Modelo IS-LM completo en Modelica (anotaciones visuales omitidas).

```

1 package ISLM_
2
3 import Modelica.Blocks.Interfaces.*;
4 import Modelica.Blocks.Sources.*;
5 // ... definiciones de modelos (Codigos 3.1 -- 3.6) ...
6
7 model Main "Modelo IS-LM basico"
8   FuncionConsumo    Consumo    (c=0.8,    C0=200);
9   FuncionInversion  Inversion  (b=50,    I0=300);
10  GastoPublico      Gasto      (G=200);
11  MercadoBienes     IS        (alpha=1.0);
12  DemandaDinero     DemandaDin (k=0.2,    h=10);
13  OfertaMonetaria   OfertaMon  (Ms=500);
14  MercadoMonetario  LM        (beta=0.5);
15 equation
16   // Flujos hacia el mercado de bienes (IS)
17   connect(Consumo.C_out,    IS.C_in);           // C --> IS
18   connect(Inversion.I_out, IS.I_in);           // I --> IS
19   connect(Gasto.G_out,     IS.G_in);           // G --> IS
20   // Retroalimentacion: Y determina C y Md
21   connect(IS.Y_out, Consumo.Y_in);             // IS --> C (Y -> C)
22   connect(IS.Y_out, DemandaDin.Y_in);         // IS --> Md (Y -> Md)
23   // Flujos hacia el mercado de dinero (LM)
24   connect(DemandaDin.Md_out, LM.Md_in);        // Md --> LM
25   connect(OfertaMon.Ms_out, LM.Ms_in);        // Ms --> LM
26   // Retroalimentacion: r determina I y Md
27   connect(LM.r_out, Inversion.r_in);          // LM --> I (r -> I)
28   connect(LM.r_out, DemandaDin.r_in);        // LM --> Md (r -> Md)
29 end Main;
30 end ISLM_;

```

Esta implementación cierra el modelo IS-LM en Modelica. La siguiente sección presenta el modelo de Solow, que extiende el análisis del corto al largo plazo incorporando la acumulación de capital y el crecimiento poblacional. En ambos casos el sistema converge automáticamente al nuevo equilibrio (Y^*, r^*).

3.5 Modelo de Solow: hipótesis y simplificaciones

A diferencia del modelo IS-LM, orientado al análisis de corto plazo, el modelo de Solow explica la dinámica de crecimiento económico en el largo plazo. Esta sección presenta las variables del modelo y las hipótesis que delimitan la variante implementada.

Tabla 3.3: Variables y parámetros del modelo de Solow.

Símbolo	Valor típico	Unidades	Significado
s	0.150 – 0.350	Adimensional	Tasa de ahorro
n	0.005 – 0.020	Fracción/año	Tasa de crecimiento poblacional
δ	0.030 – 0.070	Fracción/año	Tasa de depreciación del capital
α	0.250 – 0.400	Adimensional	Elasticidad del producto respecto al capital
A	1.000 – 2.000	Adimensional	Nivel de tecnología, constante en la versión básica (Hipótesis 5)
K_0	50 – 500	Miles €/trabajador	Capital inicial por trabajador
L_0	50 – 500	Miles personas	Población inicial

La sección se estructura en dos apartados: las hipótesis sobre las que se construye el modelo (Sec. 3.5.1) y las simplificaciones adoptadas en su implementación (Sec. 3.5.2).

3.5.1. Hipótesis del modelo de Solow

El modelo de Solow básico se asienta en cinco hipótesis principales:

Primera hipótesis: la **tasa de ahorro** s es constante y exógena: las familias ahorran una fracción fija de su ingreso con independencia del nivel de renta u otras variables.

Segunda hipótesis: la **tasa de depreciación** δ es constante: el capital se desgasta a un ritmo uniforme cada período.

Tercera hipótesis: el **crecimiento poblacional** n es constante y exógeno.

Cuarta hipótesis: la **función de producción** presenta rendimientos constantes a escala. Si se duplican todos los factores de producción (K y L), la producción se duplica. No obstante, existe rendimiento decreciente en cada factor por separado: aumentar el capital manteniendo constante el trabajo incrementa la producción, pero cada vez menos.

Quinta hipótesis: economía de un solo sector. Se asume un único bien homogéneo que sirve tanto para consumo como para inversión, y se abstrae el **progreso tecnológico** en la versión básica aquí implementada.

3.5.2. Simplificaciones del modelo de Solow

En la versión básica implementada, el modelo prescinde de los siguientes elementos:

No existe gobierno: no hay impuestos ni gasto público.

La economía es cerrada: no hay comercio internacional.

Las decisiones de ahorro/consumo de las familias no se modelan explícitamente: se asume una fracción constante s de ahorro.

No hay desempleo: toda la fuerza laboral está siempre empleada.

El progreso tecnológico A se fija como parámetro constante ($A = 1$ en la versión básica).

Estas simplificaciones permiten concentrarse en los mecanismos esenciales del crecimiento: **acumulación de capital**, depreciación y crecimiento poblacional. A pesar de su sencillez, el modelo reproduce patrones observados en datos reales, como la convergencia entre países con diferente nivel de desarrollo.

3.6 Modelo de Solow: descripción y formulación matemática

Con las hipótesis anteriores, el modelo de Solow queda caracterizado por cinco ecuaciones fundamentales que se implementarán como componentes Modelica en la Sec. 3.6.1. Se presentan en orden lógico, desde la función de producción hasta el **estado estacionario** de largo plazo. La sección se estructura en siete apartados: función de producción Cobb-Douglas (Sec. 3.6.1), ecuación de acumulación de capital (Sec. 3.6.2), crecimiento poblacional (Sec. 3.6.3), inversión (Sec. 3.6.4), estado estacionario (Sec. 3.6.5), variables per cápita (Sec. 3.6.6) e implementación en Modelica (Sec. 3.6.7).

3.6.1. Función de producción Cobb-Douglas

La producción agregada se describe mediante una función de tipo Cobb-Douglas con rendimientos decrecientes del capital:

$$Y = A \cdot K^\alpha \cdot L^{1-\alpha} \quad (3.15)$$

donde K es el stock de capital, L la fuerza laboral, $A > 0$ el nivel de tecnología y $\alpha \in (0, 1)$ la elasticidad del producto respecto al capital (fracción del producto atribuible al factor capital). En términos per cápita (dividiendo por L):

$$y = A \cdot k^\alpha \quad (3.16)$$

donde $y = Y/L$ es el producto por trabajador y $k = K/L$ el **capital por trabajador**. El parámetro $\alpha \approx 0,33$ implica que el capital explica aproximadamente un tercio del producto, un valor empírico estándar en la literatura. Definida la función de producción, el siguiente apartado modela la dinámica de acumulación del capital.

3.6.2. Ecuación de acumulación de capital

El stock de capital varía según la diferencia entre la inversión bruta y la depreciación:

$$\dot{K} = s \cdot Y - \delta \cdot K \quad (3.17)$$

donde $s \in (0, 1)$ es la tasa de ahorro, de modo que $I = s \cdot Y$ es la inversión total y $\delta \cdot K$ la depreciación del stock de capital. En términos per cápita:

$$\dot{k} = s \cdot y - (\delta + n) \cdot k \quad (3.18)$$

donde $k = K/L$ es el capital por trabajador, $y = Y/L$ el producto por trabajador, $s \in (0, 1)$ la tasa de ahorro, δ la depreciación del capital y n la tasa de crecimiento poblacional. El término $s \cdot y$ representa la inversión por trabajador y $(\delta + n) \cdot k$ el capital necesario para mantener k constante: $\delta \cdot k$ repone el capital depreciado y $n \cdot k$ equipa a los nuevos trabajadores. La ecuación incluye n como parámetro exógeno; el siguiente apartado caracteriza la dinámica de la fuerza laboral.

3.6.3. Crecimiento poblacional

La fuerza laboral crece a una tasa constante n según la siguiente ecuación diferencial lineal:

$$\dot{L} = n \cdot L \quad (3.19)$$

donde L es el tamaño de la fuerza laboral y $n > 0$ su tasa de crecimiento poblacional exógena y constante. La ecuación tiene solución analítica $L(t) = L_0 \cdot e^{nt}$, donde L_0 es la población inicial. Con la dinámica de capital y población establecidas, el siguiente apartado especifica cómo se determina la inversión.

3.6.4. Inversión

La fracción constante s de la producción que se destina a la formación de nuevo capital es:

$$I = s \cdot Y \quad (3.20)$$

donde I es la inversión total bruta, $s \in (0, 1)$ la tasa de ahorro (fracción constante de la producción que no se consume) e Y la producción agregada. Una fracción fija s de la producción se invierte en nuevo capital, cerrando el bucle de retroalimentación $K \rightarrow Y \rightarrow I \rightarrow K$ (capital, producción e inversión, respectivamente). Establecidas las cuatro ecuaciones dinámicas del modelo, el siguiente apartado calcula el punto de equilibrio de largo plazo.

3.6.5. Estado estacionario

El estado estacionario se alcanza cuando $\dot{k} = 0$ en la Ec. (3.18), es decir, cuando la inversión por trabajador iguala exactamente el capital de mantenimiento:

$$s \cdot y^* = (\delta + n) \cdot k^* \quad (3.21)$$

Sustituyendo $y^* = (k^*)^\alpha$ de la Ec. (3.16):

$$s \cdot (k^*)^\alpha = (\delta + n) \cdot k^* \quad (3.22)$$

Resolviendo para k^* :

$$k^* = \left[\frac{s}{\delta + n} \right]^{\frac{1}{1-\alpha}} \quad (3.23)$$

donde s es la tasa de ahorro, δ la tasa de depreciación, n la tasa de crecimiento poblacional y α la elasticidad del capital en la función Cobb-Douglas.

El producto por trabajador en estado estacionario es:

$$y^* = \left[\frac{s}{\delta + n} \right]^{\frac{\alpha}{1-\alpha}} \quad (3.24)$$

donde $y^* = Y^*/L$ es el producto por trabajador en el estado estacionario. Las Ec. (3.23) y (3.24) muestran que el ingreso de largo plazo depende positivamente de la tasa de ahorro s y negativamente de la depreciación δ y el crecimiento poblacional n . Estas fórmulas se utilizarán en el Capítulo 5 para validar analíticamente los resultados de la simulación.

3.6.6. Variables per cápita

Para facilitar la visualización del estado estacionario en OpenModelica, el modelo calcula las cuatro variables per cápita como cocientes de la variable agregada entre la fuerza laboral L :

$$k = \frac{K}{L} \quad (\text{capital per cápita}) \quad (3.25)$$

El producto per cápita $y = Y/L$ ya se estableció en la Ec. (3.16) ($y = A \cdot k^\alpha$). A partir de estas dos variables fundamentales se derivan la inversión y el consumo per cápita:

$$i_{pc} = I/L = s \cdot y \quad (3.26)$$

donde s es la tasa de ahorro e y el producto per cápita definido en (3.16).

$$c_{pc} = (Y - I)/L = (1 - s) \cdot y \quad (3.27)$$

donde $(1 - s)$ es la propensión media al consumo e y el producto per cápita. Con las cuatro variables per cápita definidas, el siguiente apartado presenta su traducción directa al lenguaje Modelica.

3.6.7. Implementación en Modelica del modelo de Solow

Partiendo de la formulación anterior, esta sección presenta la traducción directa de cada ecuación al código Modelica correspondiente. El modelo Solow implementado sigue la misma filosofía modular que el IS-LM: cada ecuación dinámica se encapsula en un componente con conectores tipados RealInput/RealOutput.

Componente Solow_Capital

Implementa la Ec. (3.17). Los parámetros por defecto son $K_0 = 100$ (capital inicial) y $\delta = 0,05$ (depreciación del 5% anual). Recibe como entrada la inversión I y calcula la evolución del stock de capital K . El Código 3.8 lo implementa:

Código 3.8: Componente Solow_Capital en Modelica.

```

1 model Solow_Capital
2   parameter Real K0    = 100; // Capital inicial
3   parameter Real delta = 0.05; // Tasa de depreciación (delta = 5% anual)
4   RealInput  I_in; // Inversión (entrada)
5   RealOutput K_out(start=K0, fixed=true); // Stock de capital (salida)
6 equation
7   der(K_out) = I_in - delta * K_out; // Ec. (3.20):  $dK/dt = I - \delta K$ 
8 end Solow_Capital;
```

El stock de capital acumulado alimenta directamente la función de producción, definida a continuación mediante el Componente Solow_Producción

Implementa la función de producción Cobb-Douglas (Ec. (3.15)) con $A = 1$ (parámetro implícito en la versión básica). El parámetro $\alpha = 0.33$ representa la elasticidad del producto respecto al capital. El Código 3.9 lo implementa:

Código 3.9: Componente Solow_Produccion en Modelica.

```

1 model Solow_Produccion
2   parameter Real alpha = 0.33; // Elasticidad del producto respecto al
   ↪ capital
3   RealInput  K_in; // Stock de capital (entrada)
4   RealInput  L_in; // Fuerza laboral (entrada)
5   RealOutput Y_out; // Producción agregada (salida)
6 equation
7   Y_out = K_in^alpha * L_in^(1 - alpha); // Ec. (3.18):  $Y = K^\alpha * L^{1-}$ 
   ↪ alpha)
8 end Solow_Produccion;
```

La producción depende también del segundo factor productivo, la fuerza laboral, cuya dinámica se modela a continuación.

Componente Solow_Poblacion

Implementa la ecuación diferencial Ec. (3.19). Si se dispone de datos reales de crecimiento poblacional (**conector n_tasa**), el modelo los utiliza; en caso contrario, usa el parámetro constante n . Esta lógica permite parametrizar el modelo con series históricas del Banco Mundial. El Código 3.10 lo implementa:

Código 3.10: Componente Solow_Poblacion en Modelica.

```

1 model Solow_Poblacion
2   parameter Real n = 0.02; // Tasa de crecimiento poblacional (2% anual)
3   parameter Real L0 = 100; // Población inicial
4   RealInput n_tasa; // Tasa real (si disponible, p. ej. del Banco
   ↪ Mundial)
5   RealOutput L_out(start=L0, fixed=true); // Fuerza laboral (salida)
6 equation
7   // Si hay datos reales (|n_tasa| > 0.0001), se usan; si no, el parámetro
   ↪ n
8   der(L_out) = if abs(n_tasa) > 0.0001
9                 then n_tasa * L_out
10                else n * L_out;
11 end Solow_Poblacion;

```

Cerrando el ciclo de crecimiento, la inversión traduce una fracción de la producción en nuevo capital. El componente Solow_Inversion implementa la Ec. (3.20). Al igual que la población, acepta una tasa de ahorro real procedente del Banco Mundial (**s_tasa**) o usa el parámetro por defecto $s = 0.25$. El Código 3.11 lo implementa:

Código 3.11: Componente Solow_Inversion en Modelica.

```

1 model Solow_Inversion
2   parameter Real s = 0.25; // Tasa de ahorro por defecto (25%)
3   RealInput Y_in; // Producción agregada (entrada)
4   RealInput s_tasa; // Tasa real (si disponible)
5   RealOutput I_out; // Inversión (salida)
6 equation
7   I_out = if s_tasa > 0.001
8           then s_tasa * Y_in
9           else s * Y_in;
10 end Solow_Inversion;

```

El modelo completo junto con las conexiones conecta los cuatro componentes anteriores formando el bucle de retroalimentación K (capital) $\rightarrow Y$ (producción) $\rightarrow I$ (inversión) $\rightarrow K$, al que se añade L (fuerza laboral) $\rightarrow Y$ y las **variables per cápita** (Ecs. (3.16) y (3.25)–(3.27)).

La Tabla 3.4 muestra el análisis de dependencia de variables del modelo Solow. El bucle $K \rightarrow Y \rightarrow I \rightarrow K$ emerge directamente de las ecuaciones: Y es salida de **Produccion** y entrada de **Inversion**; I es salida de **Inversion** y entrada de **Capital**.

Tabla 3.4: Análisis de dependencia de variables Solow: de las ecuaciones del modelo a las instrucciones `connect` de Modelica.

Var.	Produce	Consume	Instrucción connect
s	tasa_s API Banco Mundial	Inversion $I = s \cdot Y$	<code>connect(tasa_s.out, Inversion.s_tasa)</code>
n	tasa_n API Banco Mundial	Poblacion $\dot{L} = n \cdot L$	<code>connect(tasa_n.out, Poblacion.n_tasa)</code>
I	Inversion $I = s \cdot Y$	Capital $\dot{K} = I - \delta K$	<code>connect(Inversion.I_out, Capital.I_in)</code>
K	Capital $\dot{K} = I - \delta K$	Produccion $Y = AK^\alpha L^{1-\alpha}$	<code>connect(Capital.K_out, Produccion.K_in)</code>
L	Poblacion $\dot{L} = n \cdot L$	Produccion $Y = AK^\alpha L^{1-\alpha}$	<code>connect(Poblacion.L_out, Produccion.L_in)</code>
Y	Produccion $Y = AK^\alpha L^{1-\alpha}$	Inversion $I = s \cdot Y$	<code>connect(Produccion.Y_out, Inversion.Y_in)</code>

A partir de este análisis, el Código 3.12 ensambla los cuatro componentes y establece las cinco conexiones causales que cierran el ciclo de crecimiento del modelo de Solow:

Código 3.12: Modelo de Solow completo ensamblado en Modelica.

```

1 package Solow_Model
2
3 // ... definiciones de modelos (Códigos 3.8 -- 3.11) ...
4
5 model Main "Modelo de Solow básico"
6   Solow_Capital    Capital    (K0=100, delta=0.05);
7   Solow_Produccion Produccion(alpha=0.33);
8   Solow_Poblacion  Poblacion  (n=0.02, L0=100);
9   Solow_Inversion  Inversion  (s=0.25);
10
11 // Fuentes de tasas (valor 0 -> se usa el parámetro interno)
12 Constant tasa_s(k=0);
13 Constant tasa_n(k=0);
14
15 // Variables per capita (Ecs. 3.19, 3.28 -- 3.30)
16 Real k_percapita;
17 Real y_percapita;
18 Real i_percapita;
19 Real c_percapita;
20
21 equation
22   connect(tasa_s.out, Inversion.s_tasa);      // s_real --> I
23   connect(tasa_n.out, Poblacion.n_tasa);      // n_real --> L
24   connect(Inversion.I_out, Capital.I_in);     // I --> K
25   connect(Capital.K_out, Produccion.K_in);    // K --> Y
26   connect(Poblacion.L_out, Produccion.L_in);  // L --> Y
27   connect(Produccion.Y_out, Inversion.Y_in);  // Y --> I
28
29 // Variables per cápita
30 k_percapita = Capital.K_out / Poblacion.L_out; //
31 ↪ Ec. (3.28)
32 y_percapita = Produccion.Y_out / Poblacion.L_out; //
33 ↪ Ec. (3.19)
34 i_percapita = Inversion.I_out / Poblacion.L_out; //
35 ↪ Ec. (3.29)
36 c_percapita = (Produccion.Y_out - Inversion.I_out) / Poblacion.L_out; //
37 ↪ Ec. (3.30)
38 end Main;
39
40 end Solow_Model;

```

Con este bloque se cierra la implementación en Modelica de ambos modelos. El IS-LM queda expresado en siete componentes y el de Solow en cuatro, todos ellos con conectores tipados que garantizan la coherencia de las conexiones. La estructura modular permite sustituir cualquier componente (por ejemplo, `Solow_Produccion` por una función CES o con progreso técnico $A(t)$, o `Solow_Poblacion` con datos reales del Banco Mundial) sin modificar el resto del sistema.

3.7 Extensión del modelo Solow: Economía Circular

Esta sección presenta la extensión de Economía Circular del modelo Solow, que introduce un parámetro de circularidad ρ para modelar la reutilización del capital depreciado. Se describe en dos apartados: la formulación matemática y justificación económica (Sec. 3.7.1) y la implementación en Modelica como componente `Solow_Circularidad` (Sec. 3.7.2).

3.7.1. Motivación y formulación

El modelo de Solow estándar asume que el capital se deprecia a una tasa constante δ independientemente de si los bienes depreciados se desechan o se reutilizan. La **Economía Circular** (Daly, 1996) (Ellen MacArthur Foundation, 2013) propone que una fracción $\rho \in [0, 1]$ del capital depreciado puede reincorporarse al proceso productivo (mediante reciclaje, reparación o reutilización), reduciendo la depreciación *neta* efectiva:

$$\delta_{\text{ef}} = \delta \cdot (1 - \rho) \quad (3.28)$$

donde $\rho = 0$ recupera el modelo lineal estándar y $\rho = 1$ representaría circularidad perfecta (sin pérdida de capital). Con esta modificación la ecuación de acumulación de capital per cápita queda:

$$\dot{k} = s \cdot f(k) - [\delta \cdot (1 - \rho) + n] \cdot k \quad (3.29)$$

donde $f(k) = A \cdot k^\alpha$ es la función de producción Cobb-Douglas per cápita, $\rho \in [0, 1]$ la tasa de circularidad y n la tasa de crecimiento poblacional.

y el estado estacionario k^* se desplaza hacia un valor mayor:

$$k^* = \left[\frac{s}{\delta(1 - \rho) + n} \right]^{\frac{1}{1-\alpha}} \quad (3.30)$$

donde los parámetros tienen el mismo significado que en (3.23), siendo ρ la tasa de circularidad que reduce la depreciación efectiva.

La interpretación económica es directa: mayor circularidad ρ reduce la depreciación efectiva, por lo que se necesita menos inversión para mantener el stock de capital, elevando el estado estacionario.

3.7.2. Implementación en Modelica: componente Solow_Circularidad

El componente añade un único parámetro `rho` a la ecuación de acumulación de capital. El resto de la interfaz es idéntica a `Solow_Capital`, lo que garantiza compatibilidad total con el modelo base.

El componente `Solow_Circularidad` está integrado en el editor como **plantilla nativa**: la opción *Modelo Circular Base* de la paleta lateral (y del menú de plantillas) genera automáticamente el modelo de Solow completo sustituyendo el bloque de capital por su variante circular, listo para exportarse y simularse en OpenModelica (véase la Sec. 4.5.4).

Código 3.13: Componente Solow_Circularidad: Economía Circular integrada en el modelo de Solow.

```

1  model Solow_Circularidad
2    "Acumulacion de capital con tasa de circularidad rho"
3
4    parameter Real s      = 0.25  "Tasa de ahorro";
5    parameter Real delta = 0.05  "Tasa de depreciacion base";
6    parameter Real n      = 0.02  "Crecimiento poblacional";
7    parameter Real alpha = 0.33  "Elasticidad del capital";
8    parameter Real rho    = 0.0   "Tasa de circularidad [0,1]";
9
10   Real k(start = 1.0)          "Capital por trabajador";
11   Real y                      "Produccion per capita";
12   Real i                      "Inversion per capita";
13   Real c                      "Consumo per capita";
14   Real delta_ef               "Depreciacion efectiva";
15
16  equation
17    // Depreciacion neta segun circularidad
18    delta_ef = delta * (1 - rho);
19
20    // Funcion de produccion Cobb-Douglas
21    y = k ^ alpha;
22
23    // Identidad de inversion
24    i = s * y;
25
26    // Consumo per capita
27    c = (1 - s) * y;
28
29    // Acumulacion de capital con economia circular
30    der(k) = i - (delta_ef + n) * k;
31
32  end Solow_Circularidad;

```

3.8 Conclusiones

Este capítulo ha completado el diseño e implementación de los modelos IS-LM y Solow en el lenguaje Modelica, desde las hipótesis teóricas hasta el código funcional de cada componente.

Para el **modelo IS-LM**, se han implementado siete componentes econométricos reutilizables (funciones de consumo, inversión, gasto público, demanda de dinero, oferta monetaria y los mercados de bienes y monetario), formulados como sistemas de ecuaciones diferenciales en Modelica que convergen al equilibrio simultáneo (Y^*, r^*) mediante velocidades de ajuste α y β .

Para el **modelo de Solow**, se han implementado cuatro componentes (capital, población, producción Cobb-Douglas e inversión), cuya dinámica conjunta reproduce la convergencia al estado estacionario k^* e y^* para parámetros calibrados con datos reales del Banco Mundial.

La **extensión de Economía Circular** (`Solow_Circularidad`) demuestra la extensibilidad de la arquitectura: basta añadir el parámetro $\rho \in [0, 1]$ y modificar la ecuación de depreciación efectiva para obtener un nuevo estado estacionario sustancialmente superior, sin alterar ningún otro componente del sistema.

Los modelos quedan listos para ser manipulados visualmente mediante el editor gráfico. El siguiente capítulo describe la arquitectura software de dicho editor: los patrones de diseño empleados, las clases principales y el mecanismo que conecta la interfaz gráfica con la generación de código Modelica.

Capítulo 4

Patrones de Diseño y Arquitectura Software

4.1 Introducción

Este capítulo describe la arquitectura software del editor gráfico Modelica desarrollado. Se detallan los patrones de diseño empleados, la estructura de las clases principales y sus listeners, el funcionamiento del exportador e importador de código Modelica, la integración con fuentes de datos económicos externos y el visor de código integrado. Gracias al diseño modular y extensible, el sistema incorpora capacidad de crecimiento para nuevos dominios de modelado sin necesidad de reestructurar el código existente.

Los componentes descritos en este capítulo implementan la funcionalidad del editor que permite manipular gráficamente los modelos económicos desarrollados en el Capítulo 3, proporcionando una interfaz intuitiva que elimina la necesidad de conocer la sintaxis Modelica: el usuario trabaja con objetos visuales en el lienzo y el editor genera automáticamente el código correcto.

El capítulo se estructura en trece secciones. Tras esta introducción (Sec. 4.1), la Sec. 4.2 presenta la arquitectura general basada en el patrón Modelo-Vista-Controlador (MVC). Las secciones 4.3 y 4.4 detallan las clases de la interfaz gráfica (capa Vista): la clase principal `EditorModelicaMultiDominio` (Sec. 4.3) y el lienzo de dibujo `CanvasModelica` (Sec. 4.4). Las secciones 4.5 y 4.6 describen las clases de datos del modelo (capa Modelo): la representación de los componentes (Sec. 4.5) y la de las conexiones (Sec. 4.6). La Sec. 4.7 describe el exportador de código Modelica (`ExportadorModelica`). La Sec. 4.8 describe el `ParseadorModelica`, que reconstruye los modelos a partir de archivos `.mo`. Las secciones 4.9 y 4.10 cubren, respectivamente, el sistema de validación de conexiones y la integración con fuentes de datos económicos externas. La Sec. 4.11 presenta el visor de código integrado. La Sec. 4.12 explica el mecanismo de extensibilidad y la Sec. 4.13 recoge las conclusiones del capítulo.

4.2 Arquitectura general

La arquitectura general del editor se fundamenta en patrones de diseño de ingeniería del software documentados en la literatura clásica, en particular en el libro *Design Patterns* de Gamma et al. (Gamma et al., 1994) (la conocida «Gang of Four»), estudiados también en la asignatura de Diseño de Aplicaciones Orientadas a Objetos del Grado en Ingeniería en Tecnologías de la Información (Departamento de Lenguajes y Sistemas Informáticos. UNED, 2025), que proporcionan soluciones probadas a problemas recurrentes de diseño. El sistema implementa una arquitectura de tres capas basada en el patrón Modelo-Vista-Controlador (MVC).

La Tabla 4.1 ofrece una vista de conjunto de los patrones y principios aplicados, antes de entrar en el detalle de cada uno:

Tabla 4.1: Patrones de diseño y principios SOLID aplicados: problema que resuelven y dónde se usan.

Patrón / Principio	Problema que resuelve	Dónde se usa en el editor
MVC	Que la interfaz gráfica, los datos y la lógica de negocio queden entrelazados, dificultando el mantenimiento	Arquitectura general (Sec. 4.2); Vista en Secs. 4.3–4.4, Modelo en Secs. 4.5–4.6
Factory	Que crear cada tipo de componente requiera código disperso y repetido por todo el programa	Método <code>inicializar()</code> de <code>ComponenteModelica</code> (Sec. 4.5.3)
SRP	Clases que hacen demasiadas cosas a la vez, volviéndose frágiles ante cualquier cambio	Cada clase del editor tiene una única responsabilidad (Secs. 4.5–4.12)
OCP	Que añadir una funcionalidad nueva obligue a modificar — y arriesgar romper — código que ya funciona	Mecanismo de extensibilidad: nuevo componente en 4 ficheros sin tocar el resto (Sec. 4.12)
LSP	Que una subclase se comporte de forma incompatible con su clase base, rompiendo el código que la usa	Jerarquía de componentes Swing (paneles, diálogos)
ISP	Que una clase dependa de métodos que no necesita, solo por compartir una interfaz demasiado genérica	Conectores <code>RealInput/RealOutput</code> específicos en <code>Conector</code> (Sec. 4.5)
DIP	Que los módulos principales queden atados a detalles de implementación concretos, dificultando cambios futuros	Validación de conexiones sobre abstracciones, no implementaciones (Sec. 4.9)

La aplicación ha sido desarrollada en Java Swing, el toolkit de interfaces gráficas de usua-

rio incluido en el JDK estándar desde 1998. Swing ofrece más de 25 años de estabilidad y madurez: su API está completa y bien documentada, no requiere dependencias externas (basta con importar `javax.swing.*`) y es independiente de plataforma, ejecutándose idénticamente en Windows, macOS y Linux. Para este proyecto académico, estas características pesan más que las ventajas estéticas de JavaFX.

El sistema se estructura en once módulos agrupados según su función en el patrón MVC:

Modelo (datos): `ComponenteModelica`, datos y lógica de cada bloque económico (tipo, parámetros, conectores); `ConexiónModelica`, relaciones direccionales entre componentes.

Controladores: `ValidadorConectores`, con las reglas de validación sintáctica y semántica; `ExportadorModelica`, que genera código `.mo` a partir del modelo gráfico; `ParseadorModelica`, que lee archivos `.mo` y reconstruye el modelo en el editor; `ServicioDatosEconomicos`, integración con la API del Banco Mundial.

Vista (interfaz gráfica): `EditorModelicaMultiDominio`, ventana principal y coordinador de eventos; `CanvasModelica`, lienzo de dibujo interactivo; `PaletaComponentes`, panel lateral de bloques disponibles; `PanelAtributosDescriptivo`, inspector de propiedades del componente seleccionado; `DialogoVerCodigo`, visor de código Modelica generado.

Tabla 4.2: Mapa de clases del editor: capa MVC, responsabilidad y sección de referencia.

Capa	Clase	Responsabilidad	Sec.
Modelo	<code>ComponenteModelica</code>	Datos y lógica de cada bloque económico	4.5
Modelo	<code>ConexiónModelica</code>	Relación direccional entre componentes	4.6
Controlador	<code>ValidadorConectores</code>	Reglas de validación sintáctica y semántica	4.9
Controlador	<code>ExportadorModelica</code>	Genera código <code>.mo</code> desde el modelo gráfico	4.7
Controlador	<code>ParseadorModelica</code>	Lee <code>.mo</code> y reconstruye el modelo	4.8
Controlador	<code>ServicioDatosEconomicos</code>	Integración con API del Banco Mundial	4.10
Vista	<code>EditorModelicaMultiDominio</code>	Ventana principal, coordinador de eventos	4.3
Vista	<code>CanvasModelica</code>	Lienzo de dibujo interactivo	4.4
Vista	<code>PaletaComponentes</code>	Panel lateral de bloques disponibles	4.3
Vista	<code>PanelAtributosDescriptivo</code>	Inspector de propiedades	4.3
Vista	<code>DialogoVerCodigo</code>	Visor de código Modelica generado	4.11

La Tabla 4.2 resume esta clasificación junto con la sección del capítulo donde se describe cada clase, sirviendo como mapa de referencia rápida. El diseño del editor aplica cuatro patrones clásicos de forma coherente y justificada:

MVC (Model-View-Controller). Separa la lógica de negocio (`ComponenteModelica`, `ConexiónModelica`), la presentación gráfica (`CanvasModelica`, `PaletaComponentes`) y el control (`EditorModelicaMultiDominio`). Esta separación permite cambios en la interfaz sin afectar al modelo de datos subyacente.

Observer. El canvas observa cambios en la selección de componentes y notifica al panel de atributos mediante eventos `PropertyChange`. Esto mantiene la consistencia entre vista y modelo: cuando el usuario selecciona un bloque en el lienzo, el inspector de la derecha se actualiza automáticamente.

Strategy. `ValidadorConectores` aplica estrategias de validación distintas según el modelo económico (IS-LM o Solow), agrupando las reglas por dominio. Nuevos dominios pueden añadirse sin modificar las estrategias existentes.

Factory. `ComponenteModelica` actúa como factoría: el método `inicializar()` configura el tipo correcto de conectores y parámetros según el tipo y la biblioteca del componente, centralizando la lógica de creación y facilitando la adición de nuevos tipos.

Además, el diseño respeta los cinco principios SOLID de la programación orientada a objetos (Martin, 2000), resumidos ya en la Tabla 4.1 y detallados aquí:

- **SRP** — evita clases que hacen demasiadas cosas a la vez y se vuelven frágiles ante cualquier cambio: en el editor, cada clase tiene una única responsabilidad bien definida (Secs. 4.5–4.12).
- **OCP** — evita que añadir una funcionalidad obligue a modificar código que ya funciona, con el riesgo de romperlo: el editor está abierto a extensión sin modificar código existente (Sec. 4.12).
- **LSP** — garantiza que una subclase pueda sustituir a su clase base sin comportamientos sorpresa: las subclases son intercambiables con su tipo base, aplicado de forma general en la jerarquía de componentes Swing.
- **ISP** — evita que una clase dependa de métodos que no necesita solo por compartir una interfaz demasiado genérica: el editor usa interfaces específicas `RealInput/RealOutput` en la clase `Conector` (Sec. 4.5), sin dependencias innecesarias.
- **DIP** — evita que los módulos de alto nivel queden atados a detalles de implementación concretos, dificultando cambios futuros: dependen de abstracciones (Sec. 4.9).

Estos principios se aplican de forma transversal en los apartados 4.5 a 4.12.

La combinación de los cuatro patrones de diseño y los cinco principios SOLID descritos son un conjunto que se ha tratado de mantener coherentemente: el patrón MVC define los tres bloques principales (Modelo, Vista, Controlador); Factory y Strategy resuelven, respectivamente, cómo se crean los componentes y cómo se validan sus conexiones dentro de esos bloques; y los principios SOLID garantizan que esa estructura siga siendo mantenible y extensible a medida que el editor crece.

Esta decisión arquitectónica, de usar un controlador implícito en lugar de clases dedicadas tiene una contrapartida que conviene reconocer: las clases Vista terminan acumulando tanto lógica de presentación como lógica de control, lo que en proyectos de mayor escala podría dificultar las pruebas unitarias del comportamiento de control de forma aislada de la interfaz gráfica. Para un editor de la envergadura de este TFM, con una única ventana principal y un flujo de interacción relativamente lineal, este acoplamiento resulta manejable y evita la sobreingeniería de introducir una capa de Controlador explícita que, en la práctica, se limitaría a redirigir eventos de Swing sin añadir lógica propia. Frameworks más maduros para aplicaciones de escritorio Java, como JavaFX con su patrón MVVM (*Model-View-ViewModel*), sí separan explícitamente esta responsabilidad mediante propiedades observables, a costa de una curva de aprendizaje mayor y de las dependencias adicionales que Swing evita.

Esta arquitectura de tres capas ha demostrado su flexibilidad durante el propio desarrollo del editor: la incorporación de los tres dominios económicos soportados (IS-LM, Solow y, de forma más limitada, Doughnut) no requirió modificar ni una sola clase Vista o Controlador — únicamente `ComponenteModelica` creció con nuevos casos en su método `inicializar()` (Sec. 4.5.3), y `ValidadorConectores` incorporó las reglas semánticas correspondientes (Sec. 4.9). Esta estabilidad de la capa de presentación frente a cambios en el dominio económico es precisamente el resultado esperado de una separación MVC bien aplicada, y confirma en la práctica lo que la Tabla 4.1 enuncia de forma teórica.

La Figura 4.1 representa esta arquitectura completa, mostrando los archivos concretos que implementan cada capa y el volumen de *listeners* que materializan el Controlador implícito.

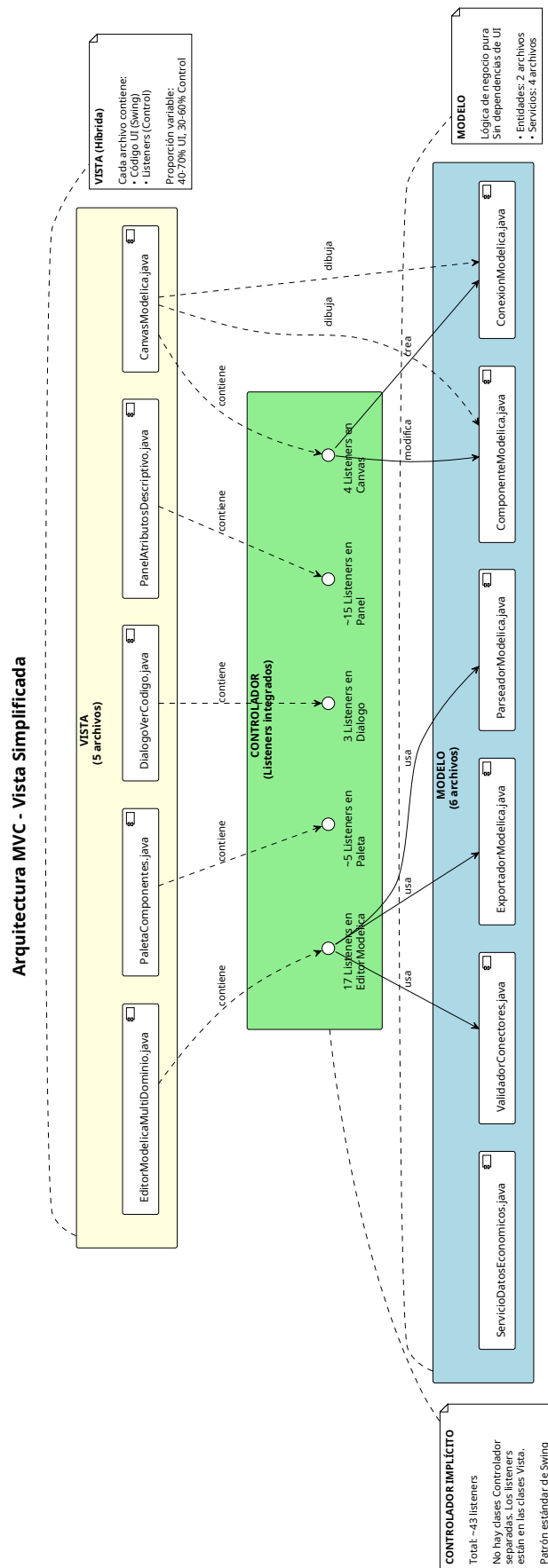


Figura 4.1: Diagrama detallado de la arquitectura MVC del editor gráfico: MODELO (6 archivos de lógica pura), VISTA (5 archivos Swing) y CONTROLADOR implícito (43 listeners integrados en las clases Vista, patrón estándar de Swing).

El diagrama destaca una decisión de diseño relevante: el editor no implementa clases Controlador independientes, sino que integra esa responsabilidad directamente en las clases Vista mediante *listeners* de Swing (`ActionListener`, `MouseListener` y similares). Esta es la forma habitual de aplicar MVC en aplicaciones de escritorio Java, donde el propio framework gráfico ya proporciona el mecanismo de eventos: separar el Controlador en clases aparte añadiría una capa de indirección sin beneficio real, dado que Swing no permite desacoplar completamente el manejo de eventos de los componentes que los generan. El flujo típico de trabajo del usuario recorre los módulos en el siguiente orden, ilustrando cómo colaboran las capas MVC en la práctica:

1. El usuario hace doble clic en un elemento de la paleta.
2. `PaletaComponentes` dispara un `PropertyChangeEvent` con el tipo seleccionado.
3. `EditorModelicaMultiDominio` recibe el evento y crea una instancia de `ComponenteModelica` con el tipo y biblioteca correspondientes.
4. `CanvasModelica` añade el componente a su lista interna y llama a `repaint()`, que activa `paintComponent()`.
5. El usuario activa el modo conexión y arrastra entre dos conectores.
6. `CanvasModelica` detecta el clic de origen (`mousePressed`) y actualiza la línea provisional en tiempo real (`mouseDragged`).
7. Al soltar (`mouseReleased`), se invoca `ValidadorConectores.validarConexión()`. Si la validación es exitosa, se crea un objeto `ConexiónModelica` y se añade a la lista; en caso contrario se muestra un diálogo de error.

Nótese que los pasos 1 a 4 corresponden a la creación de componentes, mientras que los pasos 5 a 7 cubren el establecimiento de conexiones entre ellos: ambos flujos comparten el mismo patrón de validar antes de modificar el estado del modelo, evitando que el canvas quede en un estado inconsistente ante una entrada inválida del usuario.

Este flujo de interacción se apoya en un conjunto de clases especializadas que las secciones siguientes describen una a una, comenzando por la clase raíz que coordina todo el sistema: `EditorModelicaMultiDominio`.

La Figura 4.2 detalla las clases de la capa Vista, y la Figura 4.3 las clases de las capas Modelo y Controlador, con sus atributos y métodos más relevantes, y las relaciones reales entre ellas (composición, asociación y dependencia, con sus multiplicidades), generadas con PlantUML a partir del código fuente verificado.

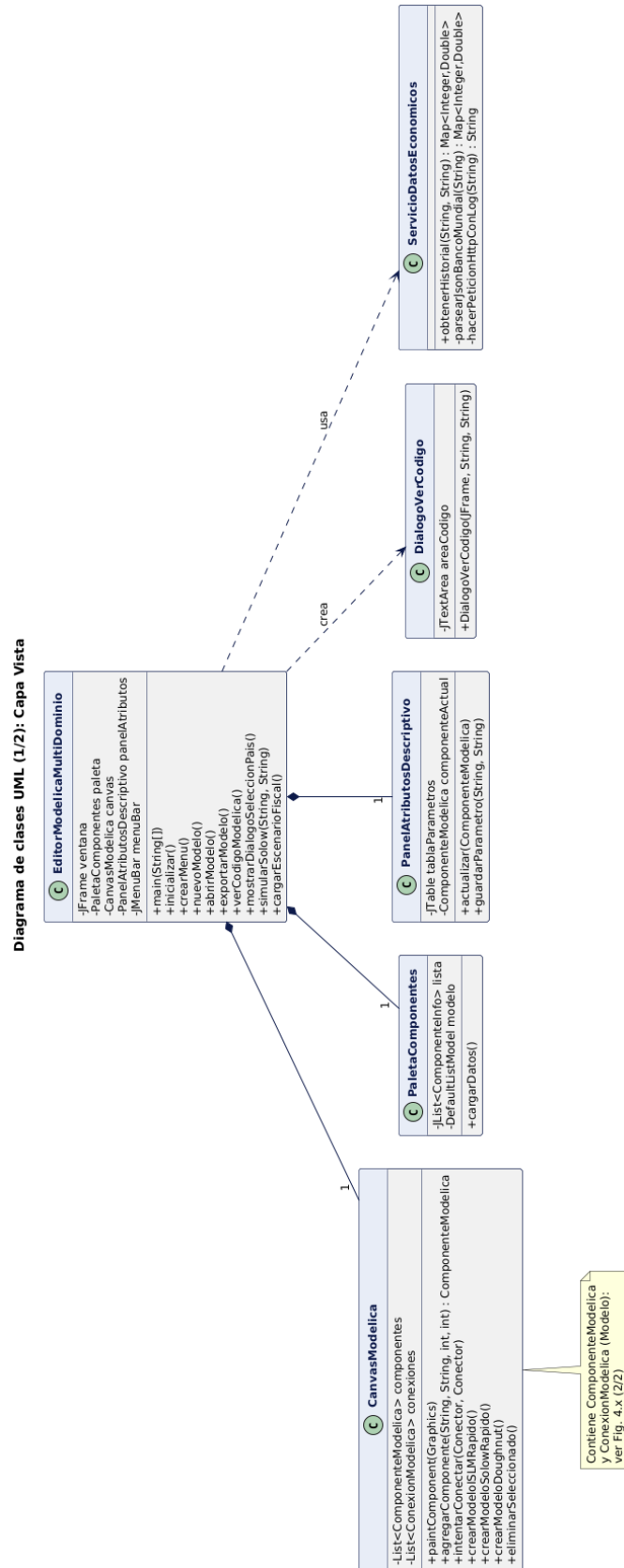


Figura 4.2: Diagrama de clases UML (1/2): capa Vista. Composición (rombo relleno) y dependencia (flecha discontinua) entre las cinco clases de interfaz gráfica.

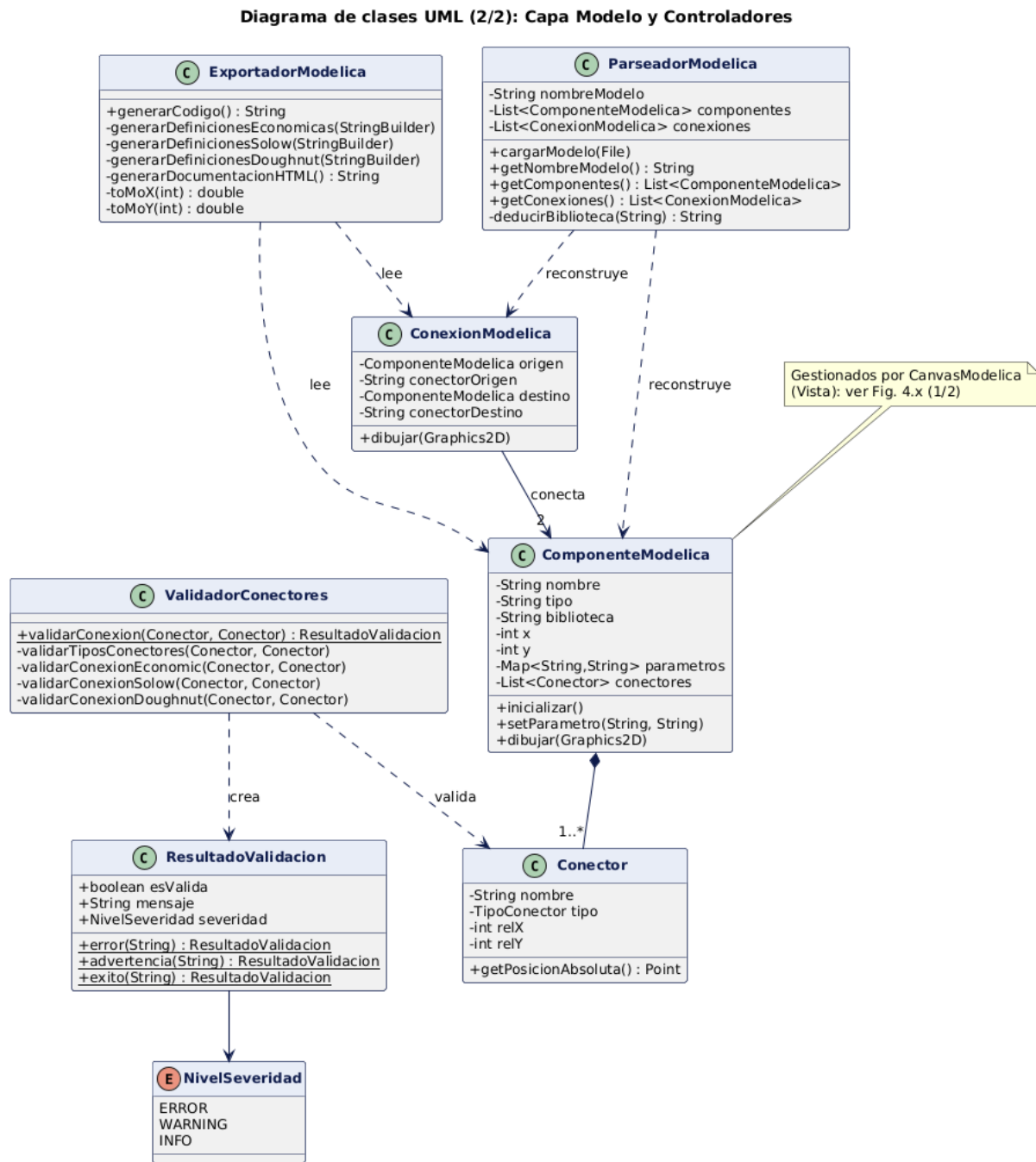


Figura 4.3: Diagrama de clases UML (2/2): capas Modelo y Controlador. Composición, asociación y dependencia entre las clases de datos (*ComponenteModelica*, *Conector*, *ConexionModelica*) y de lógica de negocio, con sus multiplicidades.

Ambos diagramas comparten un punto de unión que conviene señalar: *CanvasModelica* (Vista, Figura 4.2) compone las listas de *ComponenteModelica* y *ConexionModelica* (Modelo, Figura 4.3). Esta composición es la *única* vía por la que la capa Vista accede a los datos del sistema: ninguna otra clase Vista referencia directamente a *ComponenteModelica* o *ConexionModelica*, preservando así la separación de responsabilidades que exige el patrón MVC descrito en la Sec. 4.2.

4.3 Clase principal. EditorModelicaMultiDominio

`EditorModelicaMultiDominio` es la clase raíz del editor. Extiende `JFrame` y actúa simultáneamente como Controlador y Vista principal dentro del patrón MVC: coordina el canvas de modelado, la paleta de componentes, el panel de atributos y los servicios de exportación e importación, sin que ninguno de esos subsistemas necesite conocer a los demás directamente. Su responsabilidad es doble: construir y organizar toda la interfaz gráfica en el arranque, y responder a las acciones del usuario delegando la lógica en las clases especializadas. Esta sección describe sus atributos principales (Sec. 4.3.1), el proceso de inicialización de la interfaz (Sec. 4.3.2), los componentes Swing empleados (Sec. 4.3.3), la gestión del ciclo de vida de los modelos (Sec. 4.3.4), la integración con `OpenModelica` para el auto-lanzamiento de `OMEdit` (Sec. 4.3.5), los escenarios económicos y descarga de datos (Sec. 4.3.6), las fuentes de datos soportadas (Sec. 4.3.7) y los componentes de vista (Sec. 4.3.8).

4.3.1. Atributos de instancia

Los atributos de instancia de la clase referencian los subsistemas que coordina y mantienen el estado global del editor. La Tabla 4.3 resume los principales:

Tabla 4.3: Atributos principales de `EditorModelicaMultiDominio`.

Atributo	Tipo	Descripción
<code>canvas</code>	<code>CanvasModelica</code>	Panel central de dibujo; almacena y renderiza componentes y conexiones.
<code>paleta</code>	<code>PaletaComponentes</code>	Panel lateral izquierdo con los bloques disponibles y las plantillas.
<code>panelAtributos</code>	<code>PanelAtributos-Descriptivo</code>	Panel derecho para editar los parámetros del componente seleccionado.
<code>areaLog</code>	<code>JTextArea</code>	Registro de eventos de la sesión (cargas, exportaciones y errores).
<code>areaEcuaciones-Adicionales</code>	<code>JTextArea</code>	Área libre para ecuaciones Modelica personalizadas por el usuario.
<code>archivoActualRuta</code>	<code>String</code>	Ruta del archivo .mo abierto, usada para releer el código al visualizarlo.
<code>nombreModelo-Actual</code>	<code>String</code>	Nombre lógico del modelo en edición; identificador en la exportación.
<code>btnConectar</code>	<code>JButton</code>	Botón que activa el modo conexión; cambia de color al activarse.

Identificados los subsistemas coordinados y el estado global, el siguiente apartado describe cómo el constructor los inicializa al arrancar la aplicación.

4.3.2. Inicialización de la interfaz

El constructor delega la construcción de la UI en tres métodos privados especializados, ejecutados en secuencia para evitar un constructor monolítico:

inicializarComponentes(): instancia todos los paneles, los organiza en JSplitPane en tres columnas y registra los PropertyChangeListener que propagan eventos entre subcomponentes.

crearMenus(): construye la barra de menús con cinco menús principales (Archivo, Edición, Plantillas, Análisis y Ayuda), incluyendo atajos de teclado y separadores semánticos.

crearBarraHerramientas(): genera la barra de botones superior con acciones rápidas para nuevo, abrir, exportar, ver código, modo conexión y zoom. El método auxiliar crearBoton() centraliza el estilo y el registro de acciones.

Con los tres grupos de subcomponentes instanciados e interconectados, la siguiente sección detalla qué elemento Swing ocupa cada zona de la interfaz.

4.3.3. Componentes Swing de la ventana principal

La interfaz se construye íntegramente con la biblioteca estándar Swing. La Tabla 4.4 resume los componentes Swing utilizados y su función:

Tabla 4.4: Componentes Swing de la ventana principal.

Componente Swing	Función en el editor
JFrame	Ventana principal. Proporciona barra de título, botones de sistema y gestión de eventos del SO.
JPanel	Contenedores invisibles que estructuran la interfaz en zonas funcionales.
JScrollPane	Envuelve el canvas añadiendo barras de desplazamiento cuando el modelo supera el área visible.
JSplitPane	Divide la ventana en tres columnas redimensionables: paleta (200 px), canvas (central), atributos (300 px).
JButton	Cada acción de la barra de herramientas. Admite ActionListeners, iconos y estilos personalizados.
JMenuBar / JMenu / JMenuItem	Jerarquía que construye el sistema de menús con atajos de teclado.
JTextArea	Usada para el log de eventos y para el área de ecuaciones adicionales. Ambas con scroll propio.

La disposición general es la siguiente: el `JFrame` contiene un `JSplitPane` horizontal principal que divide la ventana en paleta (izquierda), zona central y panel de atributos (derecha). La zona central es a su vez un `JSplitPane` vertical que separa el canvas envuelto en `JScrollPane` del área de log inferior. Esta estructura en paneles anidados permite que el usuario redimensione cualquier zona sin romper el layout. Construida la interfaz, el controlador expone los métodos que gestionan el ciclo de vida de los modelos económicos.

4.3.4. Gestión del ciclo de vida de los modelos

El editor controla el ciclo de vida completo de un modelo a través de cuatro métodos principales:

nuevoModelo(): limpia el canvas y restablece el nombre del modelo, solicitando confirmación si hay cambios sin guardar.

abrirModelo(): abre un selector de archivos `.mo`, invoca `ParseadorModelica` para reconstruir componentes y conexiones en el canvas, y actualiza la etiqueta de archivo activo.

exportarModelo(): recopila el estado completo del canvas (componentes, conexiones, ecuaciones adicionales y datos históricos) y lo entrega a `ExportadorModelica` para generar el archivo `.mo`.

verCodigoModelica(): lee el archivo actual en disco si existe, o genera uno temporal, y presenta el código en `DialogoVerCodigo`.

4.3.5. Integración con OpenModelica: auto-lanzamiento de OMEdit

El editor puede lanzar OMEdit automáticamente tras exportar el modelo, eliminando el paso manual de abrir el archivo. La integración se implementa mediante `ProcessBuilder` de Java con detección automática de la ruta de instalación de OpenModelica. El Listado 4.1 muestra la lógica de detección: el editor prueba las rutas típicas de cada sistema operativo (Windows, Linux, macOS) y lanza OMEdit con el archivo exportado como argumento en cuanto encuentra una instalación válida:

Código 4.1: Auto-lanzamiento de OMEdit: detección multiplataforma de la ruta de instalación mediante `ProcessBuilder`.

```

1 String[] rutas = {
2     "C:/OpenModelica/bin/OMEdit.exe", // Windows ruta típica
3     "D:/OpenModelica/bin/OMEdit.exe", // unidad alternativa
4     "/usr/bin/OMEdit",                // Linux
5     "/Applications/OpenModelica.app/Contents/MacOS/OMEdit" // macOS
6 };
7 for (String ruta : rutas) {
8     if (new File(ruta).exists()) {
9         new ProcessBuilder(ruta, archivoMo.getAbsolutePath())
10            .start();
11         break;
12     }
13 }

```

La ruta preferida por el usuario se persiste entre sesiones mediante `java.util.prefs.Preferences`, de modo que solo se realiza la detección automática la primera vez. El usuario puede sobrescribir la ruta desde el menú **Herramientas** → **Configurar OpenModelica**.

4.3.6. Escenarios económicos y descarga de datos reales

Un segundo grupo de métodos implementa los casos de uso propios del dominio macro-económico. Las operaciones de red se ejecutan en hilos secundarios y las actualizaciones del canvas se despachan al EDT de Swing mediante `SwingUtilities.invokeLater`, evitando bloqueos y condiciones de carrera:

`mostrarDialogoSeleccionPais()`: presenta un selector de país a partir del catálogo de `ServicioDatosEconomicos` y redirige al flujo IS-LM o Solow según la elección.

`exportarPaisAModelica(país, iso)`: descarga en hilo secundario los datos históricos de gasto público y oferta monetaria del Banco Mundial, escala los valores, los inyecta en un modelo IS-LM y recarga el resultado en el canvas.

`simularSolow(país, iso)`: mismo flujo para el modelo Solow, inyectando tasa de ahorro y crecimiento poblacional.

`cargarEscenarioBasico()` / `Fiscal()` / `Monetario()`: cargan la plantilla IS-LM base y modifican selectivamente G o M_s para representar los choques macroeconómicos clásicos, con un cuadro informativo que anticipa el efecto esperado. El escenario fiscal aumenta G de

200 a 300 (el mismo choque validado analíticamente en la Sec. 5.7.2); el escenario monetario aumenta M_s de 500 a 700 (+40 %), un choque más moderado que el de 500 a 1200 empleado en la validación del multiplicador monetario (Sec. 5.7.6), por lo que ambos no son directamente comparables. Todos estos métodos consumen la integración con APIs de datos externos, cuyas fuentes disponibles se detallan a continuación.

4.3.7. Fuentes de datos soportadas

La única fuente de datos integrada es la API pública del Banco Mundial (<https://api.worldbank.org/v2/>), que proporciona cobertura de más de 200 países desde 1960 hasta el presente en formato JSON (Bray, 2017). Los indicadores macroeconómicos concretos utilizados (gasto público, oferta monetaria, tasa de ahorro y crecimiento poblacional) se detallan en la Tabla 4.8 de la Sec. 4.10, donde se describe la arquitectura interna del servicio de descarga. Con las fuentes de datos configuradas, los dos componentes visuales que cierran la interfaz del editor son la paleta de bloques y el inspector de propiedades.

4.3.8. Componentes de vista: `PaletaComponentes` y `PanelAtributosDescriptivo`

`PaletaComponentes` extiende `JPanel` y presenta en una lista lateral vertical los bloques constructivos organizados en tres categorías: plantillas completas (`Modelo_IS-LM_Base`, `Modelo_Solow_Base`, `Modelo_IS-LM con datos reales`), componentes IS-LM individuales (`FuncionConsumo`, `FuncionInversion`, `GastoPublico`, `DemandaDinero`, `OfertaMonetaria`, `MercadoBienes`, `MercadoMonetario`) y componentes Solow individuales (`Solow_Capital`, `Solow_Poblacion`, `Solow_Produccion`, `Solow_Inversion`, `Solow_Datos`). Al hacer doble clic, la paleta dispara el evento `componenteSeleccionado` mediante `PropertyChangeEvent`; el editor lo recibe y, si es una plantilla, carga el modelo completo; si es un componente individual, lo añade al canvas.

`PanelAtributosDescriptivo` es el inspector de propiedades que se actualiza dinámicamente al cambiar la selección. Se estructura en tres secciones: descripción (`JTextArea` de solo lectura con el rol económico del componente), parámetros (una `JTable` de tres columnas: nombre, valor editable, significado, en lugar de campos de texto individuales, lo que centraliza la edición en un único componente Swing) y fórmulas (`JTextArea` con la ecuación diferencial o algebraica del componente). Al igual que en el resto del editor (Sec. 4.7.3), la tabla no valida el formato del valor introducido: guarda directamente el texto de la celda mediante `ComponenteModelica.setParametro()` tras perder el foco, con una etiqueta de estado («Guardado: parámetro») que se auto-reinicia a los dos se-

gundos como única confirmación visual. Un botón «Aplicar Cambios» permite forzar el guardado de una celda que aún está en edición. Para componentes con datos del Banco Mundial aparece además el botón «Ver Datos TimeTable». El contenido de las tres secciones no es texto fijo: cuatro métodos internos (`obtenerDescripcionComponente()`, `obtenerDescripcionDatos()`, `obtenerSignificadoParametro()`, `obtenerFormulas()`) mapean cada tipo de componente y cada parámetro a su explicación económica mediante tablas `switch`, con reconocimiento adicional por subcadena en el nombre para componentes con datos reales inyectados. Estas explicaciones incluyen el código de indicador del Banco Mundial correspondiente (p.ej. `NE.GDI.TOTL.ZS` para la tasa de ahorro, `SP.POP.GROW` para el crecimiento poblacional), enlazando directamente con los indicadores de la Sec. 4.10.

Con la clase principal y los componentes de Vista definidos, la siguiente sección detalla el núcleo visual del editor: el lienzo de dibujo `CanvasModelica`, donde los modelos económicos cobran forma interactiva.

4.4 El lienzo del editor. `CanvasModelica`

`CanvasModelica` hereda de `JPanel` y constituye la superficie de dibujo principal. La clase gestiona dos estructuras de datos fundamentales que mantienen el estado completo del modelo: una lista de `ComponenteModelica` y una lista de `ConexiónModelica`. Sobre estas listas opera el sistema de interacción mediante `MouseAdapter`. La sección se organiza en tres apartados: el sistema de interacción con el ratón (Sec. 4.4.1), el ciclo de renderizado (Sec. 4.4.2) y la eliminación de elementos (Sec. 4.4.3).

La Figura 4.4 ilustra, mediante un diagrama de secuencia UML, el flujo de trabajo típico del usuario en el editor.

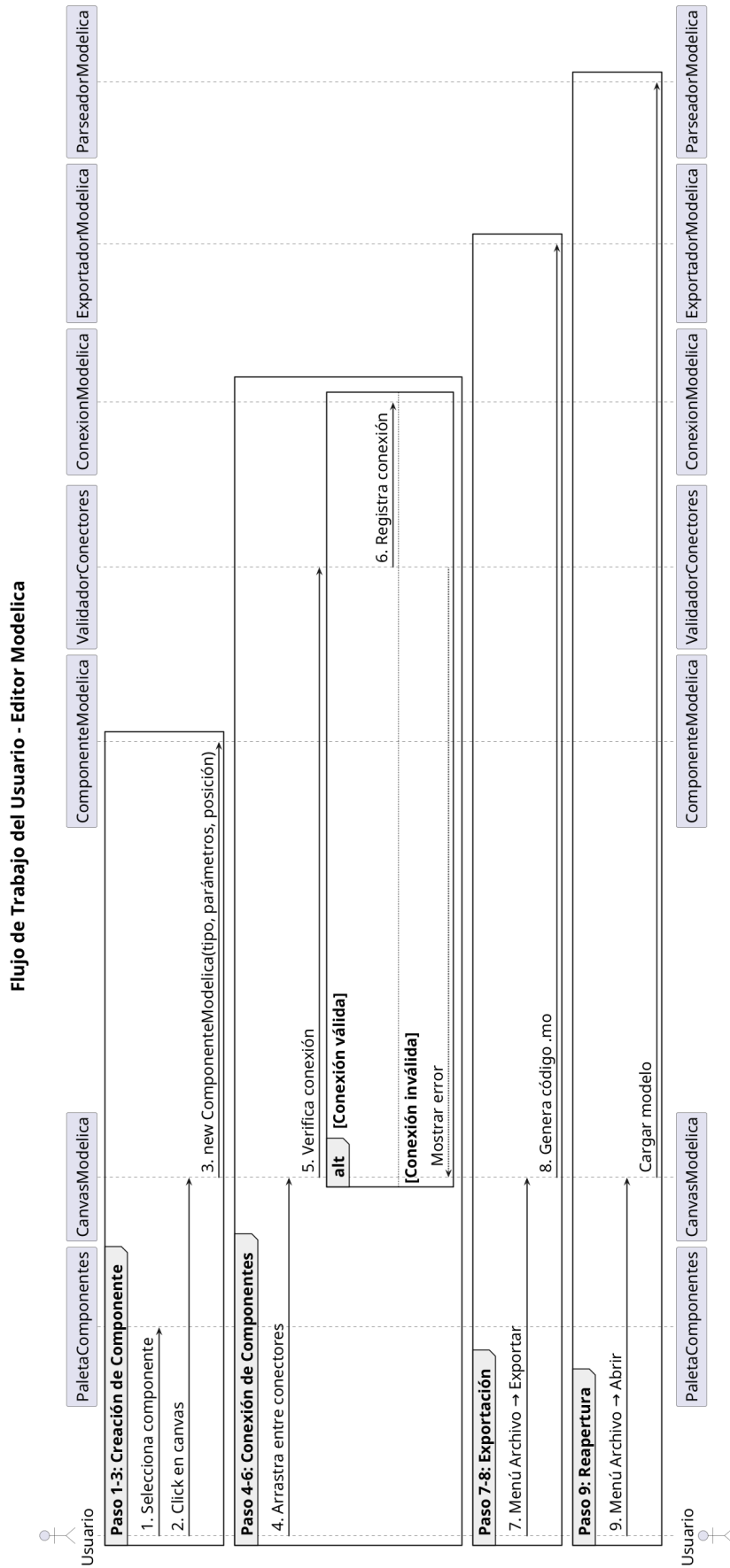


Figura 4.4: Diagrama de secuencia UML: flujo de trabajo del usuario en el editor.

4.4.1. Sistema de interacción con el ratón

El canvas implementa cuatro comportamientos de ratón diferenciados:

- **Selección.** Clic sobre un bloque: el canvas lo resalta con borde azul y emite un `PropertyChangeEvent` que actualiza `PanelAtributosDescriptivo`.
- **Arrastre.** Se calcula un offset ratón-esquina; durante `mouseDragged` las coordenadas (x, y) se actualizan continuamente y las conexiones se redibujan en tiempo real.
- **Modo conexión.** Dos clics sucesivos (conector origen $\rightarrow$ conector destino); durante el arrastre se dibuja una línea provisional punteada. Al soltar se invoca `ValidadorConectores`; si es válida se crea `ConexiónModelica`, si no se muestra un diálogo de error.
- **Arrastrar y soltar desde la paleta.** `DragGestureListener` crea un paquete `Transferable`; el `DropTargetListener` del canvas instancia el componente en las coordenadas exactas de la soltura, garantizando nombres únicos mediante un sufijo numérico.

Gestionada la interacción del ratón, el siguiente apartado describe el ciclo de renderizado que traduce el estado del modelo a píxeles en cada fotograma.

4.4.2. Ciclo de renderizado

El método `paintComponent(Graphics g)` sigue un orden de dibujado determinista:

1. Fondo blanco completo para limpiar el fotograma anterior.
2. *Grid* punteado gris claro opcional como referencia visual de alineación.
3. Todas las conexiones: `conexion.dibujar(g)` por cada `ConexiónModelica`.
4. Todos los componentes: `componente.dibujar(g)` por cada `ComponenteModelica`.
5. Si hay una conexión en curso, línea provisional punteada desde `conectorOrigen` hasta `puntoRatonActual`.

El orden conexiones-primero, componentes-después es deliberado: garantiza que los componentes aparezcan siempre sobre las líneas de conexión, mejorando la claridad visual del diagrama. El tercer comportamiento del canvas gestiona la eliminación de elementos manteniendo la integridad del grafo.

4.4.3. Eliminación de elementos

El método `eliminarSeleccionado()` elimina el componente o conexión actualmente seleccionado. Cuando se elimina un componente, también se eliminan automáticamente todas las conexiones que lo involucran, manteniendo la consistencia del modelo. Esto implementa el principio de integridad referencial a nivel de grafo visual.

El comportamiento visual del canvas depende directamente de los objetos que renderiza. La sección siguiente examina `ComponenteModelica`, la clase que encapsula la representación de datos y la lógica gráfica de cada bloque económico.

4.5 Representación de componentes.

`ComponenteModelica`

`ComponenteModelica` encapsula toda la representación de un componente individual del modelo en el lienzo. Agrupa la lógica de datos (tipo, parámetros, posición) y la lógica de presentación gráfica (renderizado con `Graphics2D`), constituyendo el núcleo del Modelo en el patrón MVC. La clase se describe en cuatro apartados: la estructura interna y atributos (Sec. 4.5.1), la clase interna `Conector` (Sec. 4.5.2), el método `inicializar()` como factoría (Sec. 4.5.3) y las plantillas de modelos predefinidos (Sec. 4.5.4).

4.5.1. Estructura interna

La Tabla 4.5 recoge los atributos principales de la clase `ComponenteModelica`: Estos atributos son gestionados y renderizados a través de la clase interna `Conector`, que encapsula los puertos de comunicación del componente.

Tabla 4.5: Atributos principales de `ComponenteModelica`.

Atributo	Tipo	Descripción
nombre	String	Identificador único, p. ej. <code>FuncionConsumo_1</code> .
tipo	String	Tipo de componente, p. ej. <code>FuncionConsumo</code> .
biblioteca	String	Dominio al que pertenece: <code>Economic</code> (IS-LM) o <code>Solow</code> .
x, y	int	Coordenadas de posición en el lienzo (píxeles Java).
ancho, alto	int	Dimensiones del rectángulo (por defecto 100×60 px).
parametros	Map<String, String>	Valores configurables, p. ej. {c=0.8, C0=200}.
conectores	List<Conector>	Puertos de entrada y salida del componente.
datosHistoricos	Map<Integer, Double>	Serie temporal del Banco Mundial (clave: año → valor).
colorFondo	Color	Amarillo (<code>Economic</code>), azul (<code>Solow</code>), morado (<code>Doughnut</code>).
simbolo	String	Texto del bloque, p. ej. <code>C(Y)</code> , <code>IS</code> .

4.5.2. Clase interna conector

Cada componente contiene múltiples conectores que representan sus puertos de entrada y salida. La clase interna `Conector` almacena el nombre del puerto (p. ej. `Y_in`, `C_out`), el tipo de señal (`RealInput`, `RealOutput`), las coordenadas relativas al centro del componente (`relX`, `relY`) y la referencia al `ComponenteModelica` padre.

El método clave es `getPosicionAbsoluta()`, que calcula la posición en píxeles sumando las coordenadas relativas a la posición actual del componente padre. Esto garantiza que cuando el componente se mueve, sus conectores se desplazan automáticamente y las líneas de conexión siguen siendo correctas sin recalculación explícita.

El método `dibujar(Graphics2D g)` de cada `Conector` lo representa como un círculo de 8 px de diámetro: verde para entradas (`RealInput`) y rojo para salidas (`RealOutput`), proporcionando retroalimentación visual inmediata sobre los tipos de puertos. La creación y configuración de estos conectores es responsabilidad del método `inicializar()`, que actúa como factoría centralizada.

4.5.3. Método inicializar() como factoría

El método `inicializar()` actúa como factoría (patrón Factory) configurando el componente según tipo y biblioteca: un único punto de entrada centraliza la creación de todos los tipos de bloque, lo que garantiza coherencia y simplifica la adición de nuevos componentes (véase Sec. 4.12). El Listado 4.2 muestra los casos más representativos del switch, uno por cada tipo de componente económico registrado:

Código 4.2: Biblioteca de componentes IS-LM y Solow (patrón Factory en `inicializar()`).

```

1 // Biblioteca ECONOMIC (IS-LM)
2 case "FuncionConsumo":
3     simbolo = "C(Y)";
4     parametros.put("C0", "200"); parametros.put("c", "0.8");
5     addConector("Y_in", "RealInput", -50, 0);
6     addConector("C_out", "RealOutput", 50, 0);
7     break;
8 case "MercadoBienes":
9     simbolo = "IS ()";
10    ancho = 120; alto = 80;
11    addConector("C_in", "RealInput", -60, -30);
12    addConector("I_in", "RealInput", -60, -10);
13    addConector("G_in", "RealInput", -60, 10);
14    addConector("Y_out", "RealOutput", 60, 0);
15    break;
16 // Biblioteca SOLOW (Crecimiento)
17 case "Solow_Capital":
18     simbolo = "K";
19     parametros.put("delta", "0.05");
20     addConector("I_in", "RealInput", -50, 0);
21     addConector("K_out", "RealOutput", 50, 0);
22     break;

```

El diseño permite añadir fácilmente nuevos tipos de componentes extendiendo el switch en `inicializar()`, como se detalla en la Sec. 4.12 sobre extensibilidad.

La Tabla 4.6 recoge el inventario completo de los 16 tipos de componentes actualmente registrados en el sistema:

Tabla 4.6: Registro completo de componentes del editor (16 tipos).

Tipo	Biblio.	Símbolo	Conect.	Ecuación principal
<i>Biblioteca IS-LM (Economic)</i>				
FuncionConsumo	Economic	C(Y)	2	$C = C_0 + c \cdot Y$
FuncionInversion	Economic	I(r)	2	$I = I_0 - b \cdot r$
GastoPublico	Economic	G	1	$G = G_0$ (exógeno)
MercadoBienes	Economic	IS()	4	$\dot{Y} = \alpha_{IS}(C + I + G - Y)$
DemandaDinero	Economic	Md	3	$M_d = kY - hr$
OfertaMonetaria	Economic	Ms	1	$M_s = M_{s0}$ (exógeno)
MercadoMonetario	Economic	LM	3	$\dot{r} = \beta_{LM}(M_d - M_s)$
<i>Biblioteca Solow (Solow)</i>				
Solow_Capital	Solow	K	2	$\dot{K} = I - \delta K$
Solow_Produccion	Solow	Y	3	$Y = K^\alpha L^{1-\alpha}$
Solow_Poblacion	Solow	L	2	$\dot{L} = n \cdot L$
Solow_Inversion	Solow	I	3	$I = s \cdot Y$
Solow_Datos	Solow	src	1	Series históricas (TimeTable WB)
Solow_Circularidad	Solow	K_ρ	0	$\dot{k} = sy - [\delta(1 - \rho) + n]k$
<i>Biblioteca Doughnut (prueba de concepto)</i>				
Doughnut_RentaMin	Doughnut	R $\geq$	2	$Y_{in} \geq Y_{min} \cdot Y_{ref}$ (0,6×1000)
Doughnut_EducMin	Doughnut	E $\geq$	2	$\phi \cdot G_{in} \geq E_{min}$ (0,15×G _{in} ≥ 30)
Doughnut_RecursoMax	Doughnut	R $\leq$	2	ok = 1 si $\mu Y \leq R_{max}$ (uso de recursos proporcional a Y)
<i>Nota:</i> α_{IS} y β_{LM} son velocidades de ajuste del mercado de bienes y monetario; α en la fila Solow es la elasticidad Cobb-Douglas. δ: tasa de depreciación. Y_{min} (0,6) y Y_{ref} (1000): fracción mínima de renta y renta de referencia. ϕ (0,15): fracción del gasto público destinada a educación. E_{min} (30): gasto educativo mínimo exigido. Los tres componentes Doughnut emiten un indicador booleano ok_out en vez de una variable continua.				

4.5.4. Plantillas de modelos predefinidos

Cuatro métodos del canvas actúan como plantillas predefinidas que instancian un modelo completo (IS-LM, Solow, Solow Circular o Doughnut) con todos sus componentes ya conectados, sin que el usuario tenga que arrastrar y conectar cada bloque manualmente. El Listado 4.3 muestra `crearModeloISLMRapido()`: crea los siete componentes IS-LM, los posiciona en el lienzo y establece las nueve conexiones causales del modelo con una sola llamada. Se activa cuando el usuario hace doble clic en la plantilla «Modelo IS-LM Base» de la paleta:

Código 4.3: `crearModeloISLMRapido()`: plantilla que instancia los 7 componentes IS-LM y sus 9 conexiones causales en un clic.

```

1 public void crearModeloISLMRapido(boolean conDatosReales) {
2     limpiarCanvas();
3     int cx = visibleWidth / 2 - 100;
4     int cy = visibleHeight / 2;
5     ComponenteModelica cons = agregarComponente("Consumo",
6         "FuncionConsumo", "Economic", cx - 250, cy - 150);
7     ComponenteModelica inv = agregarComponente("Inversion",
8         "FuncionInversion", "Economic", cx - 250, cy);
9     ComponenteModelica gp = agregarComponente("Gasto",
10        "GastoPublico", "Economic", cx - 250, cy + 150);
11    ComponenteModelica is = agregarComponente("IS",
12        "MercadoBienes", "Economic", cx, cy);
13    ComponenteModelica md = agregarComponente("DemandaDin",
14        "DemandaDinero", "Economic", cx + 250, cy - 100);
15    ComponenteModelica ms = agregarComponente("OfertaMon",
16        "OfertaMonetaria", "Economic", cx + 250, cy + 100);
17    ComponenteModelica lm = agregarComponente("LM",
18        "MercadoMonetario", "Economic", cx + 500, cy);
19    crearConexion(cons, "C_out", is, "C_in");
20    crearConexion(inv, "I_out", is, "I_in");
21    crearConexion(gp, "G_out", is, "G_in");
22    crearConexion(is, "Y_out", cons, "Y_in");
23    crearConexion(is, "Y_out", md, "Y_in");
24    crearConexion(md, "Md_out", lm, "Md_in");
25    crearConexion(ms, "Ms_out", lm, "Ms_in");
26    crearConexion(lm, "r_out", inv, "r_in");
27    crearConexion(lm, "r_out", md, "r_in");
28    repaint();
29 }

```

El método homólogo `crearModeloSolowRapido()` sigue el mismo patrón para los cuatro componentes Solow, cerrando el bucle Y (producción) $\rightarrow I$ (inversión) $\rightarrow K$ (capital) $\rightarrow Y$ que es la característica definitoria del modelo neoclásico de crecimiento. Una tercera plantilla, `crearModeloCircularRapido()`, reutiliza ese mismo esquema pero instancia el bloque de capital como `Solow_Circularidad` (con la tasa de circularidad ρ), de modo que la extensión de economía circular del Capítulo 3 queda disponible en un clic como modelo nativo del editor, exportable y simulable en OpenModelica.

La cuarta plantilla, `crearModeloDoughnut()`, construye el núcleo IS-LM completo (los

mismos siete componentes de `crearModeloISLMRapido()` y lo amplía con los tres bloques Doughnut de la Tabla 4.6 (renta mínima, acceso a educación y uso de recursos) ya parametrizados ($Y_{\min} = 0,6$, $\phi = 0,15$, $\mu = 0,3$) y conectados a las variables de salida Y y G del núcleo IS-LM. La biblioteca Doughnut no es, por tanto, solo una prueba de concepto de componentes sueltos: dispone de una plantilla nativa completa equivalente en madurez a las de IS-LM y Solow.

Sobre el nombrado de instancias: los ejemplos anteriores usan nombres semánticos fijos ("Consumo", "IS", "LM"...), porque las plantillas los asignan explícitamente. Cuando el usuario arrastra un componente suelto desde la paleta en lugar de usar una plantilla, el canvas genera el nombre automáticamente como `tipo + "_" + contador`, donde el contador es el número total de componentes ya presentes en el canvas (compartido entre todos los tipos, no uno independiente por tipo), por lo que dos componentes de tipos distintos añadidos consecutivamente no reinician la numeración.

Una vez que los componentes están en el canvas, el usuario necesita establecer relaciones causales entre ellos. La sección siguiente describe `ConexiónModelica`, la clase que modela esas relaciones y su representación visual.

4.6 Representación de conexiones. `ConexiónModelica`

La clase `ConexiónModelica` modela una relación direccional entre dos puertos de componentes distintos. Almacena cuatro referencias: el `ComponenteModelica` origen, el nombre del conector de salida, el `ComponenteModelica` destino y el nombre del conector de entrada. Esta información es suficiente para generar la instrucción Modelica `connect(comp1.conector1, comp2.conector2)` y para renderizar la línea de conexión en el canvas.

El método `dibujar(Graphics2D g)` obtiene las posiciones absolutas de ambos conectores mediante `getPosicionAbsoluta()` y traza una línea desde el origen hasta el destino. Para mejorar la claridad del diagrama en modelos complejos, las líneas pueden incluir un punto intermedio para evitar solapamientos. Se usa color azul oscuro (0, 0, 150) como estándar de señales causales, coherente con las convenciones de Modelica.

Una conexión queda invalidada si se elimina alguno de sus componentes extremos; el canvas detecta esto en `eliminarSeleccionado()` y borra automáticamente todas las conexiones huérfanas, manteniendo la integridad del grafo del modelo.

La Figura 4.5 sintetiza visualmente cómo se relacionan las tres piezas descritas en esta sección y la anterior: cada `ComponenteModelica` expone sus `Conector` como puntos verdes (entrada) y rojos (salida);

una `ConexiónModelica` enlaza un puerto de salida con uno de entrada; y esa relación en memoria es precisamente lo que el `ExportadorModelica` (Sec. 4.7) traduce a la instrucción `connect()` del archivo `.mo` generado.

De `ComponenteModelica` y `Conector` a `ConexiónModelica` y código `.mo`

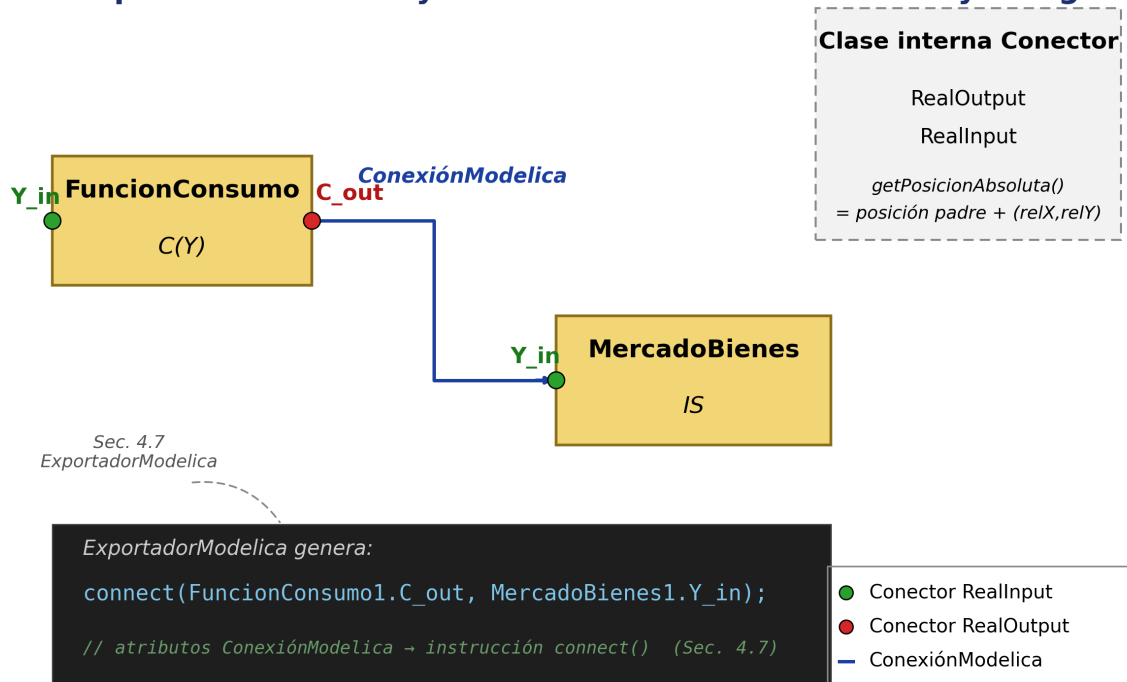


Figura 4.5: De `ComponenteModelica` y `Conector` a `ConexiónModelica` y código `.mo`: el puente visual entre el modelo de datos en memoria (Secs. 4.5–4.6) y su serialización a `Modelica` (Sec. 4.7).

Con los objetos del modelo (`ComponenteModelica`, `ConexiónModelica`) completamente definidos, el siguiente paso lógico es describir cómo se serializa ese modelo a código textual. La Sec. 4.7 presenta el `ExportadorModelica`, que transforma el estado en memoria en un archivo `.mo` estándar.

4.7 Generador-Exportador a código Modelica.

ExportadorModelica

`ExportadorModelica` es el núcleo de la funcionalidad de exportación: transforma el grafo de componentes y conexiones almacenado en memoria en un archivo `.mo` sintácticamente

correcto y compatible con el estándar Modelica. La sección se estructura en cinco apartados: estructura del archivo .mo generado (Sec. 4.7.1), transformación de coordenadas Java → Modelica (Sec. 4.7.2), validación de identificadores (Sec. 4.7.3), generación de documentación HTML embebida (Sec. 4.7.4) e integración de datos históricos como bloques TimeTable (Sec. 4.7.5). La Figura 4.6 muestra el pipeline completo, desde la recolección de datos del canvas hasta la generación del archivo .mo.

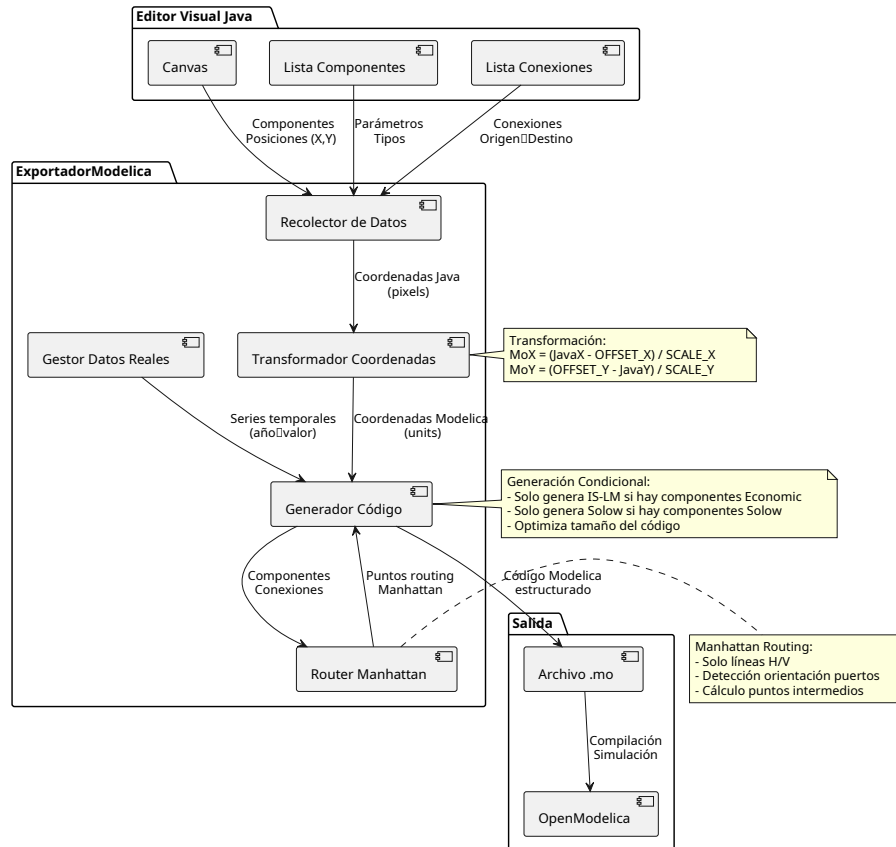


Figura 4.6: Pipeline de exportación a código Modelica: recolección, transformación de coordenadas, generación condicional y enrutado Manhattan.

4.7.1. Estructura del archivo .mo generado

El método orquestador es `generarCodigo()`: construye el archivo completo en un único `StringBuilder` siguiendo una secuencia fija de fases: definiciones de tipos por biblioteca, definiciones de datos reales, modelo `Main` con instancias, ecuaciones de conexión, documentación HTML y cierre del paquete, tal como muestra el Listado 4.4.

Un aspecto que merece explicación específica es cómo el exportador evita declarar el mismo tipo de componente más de una vez, algo que Modelica rechazaría como error de compilación por definición duplicada, sin necesidad de rastrear qué tipos concretos están en uso. La solución adoptada es más simple que un registro

de tipos ya emitidos: cada biblioteca económica (Economic, Solow, Doughnut) tiene su catálogo completo de definiciones de modelo escrito una única vez dentro de un método dedicado (`generarDefinicionesEconomicas()`, `generarDefinicionesSolow()`, `generarDefinicionesDoughnut()`), y cada uno de esos métodos se invoca como máximo una vez por archivo generado, condicionado únicamente a que exista al menos un componente de esa biblioteca en el canvas (`componentes.stream().anyMatch(...)`). El catálogo completo de la biblioteca se escribe de una sola vez aunque solo se use un subconjunto de sus tipos, evitando así cualquier declaración duplicada sin necesidad de un registro de tipos ya emitidos. Las instancias, en cambio, se generan por separado en un segundo recorrido sobre `componentes` dentro del modelo `Main`, referenciando el tipo ya declarado por su nombre.

El código generado sigue la estructura de paquete Modelica estándar. El Listado 4.4 ilustra esa estructura:

Código 4.4: Estructura del archivo `.mo` generado por el exportador: definiciones de tipos, modelo `Main` e instrucciones `connect`.

```

1 package NombreModelo
2 // 1. Definiciones de tipos de componentes
3 model FuncionConsumo
4   parameter Real C0 = 100;
5   parameter Real c = 0.8;
6   RealInput Y_in annotation(Placement(...));
7   RealOutput C_out annotation(Placement(...));
8   equation
9     C_out = C0 + c * Y_in;
10    annotation(Icon(graphics={...}),
11      Documentation(info="<html>...</html>"));
12 end FuncionConsumo;
13 // 2. Modelo principal con instancias y conexiones
14 model Main
15   FuncionConsumo Consumo_1(c=0.8, C0=200)
16   annotation(Placement(transformation(origin={-21.2,-20.4},
17     extent={{-40,-40},{40,40}})));
18   ...
19   equation
20     connect(Consumo_1.C_out, IS_1.C_in)
21     annotation(Line(points={{-21.2,19.6},{-21.2,-20.4}},
22       color={0,0,255}));
23 end Main;
24 end NombreModelo;
```

primero se declaran los modelos de cada tipo de componente con sus parámetros y ecuaciones; después se instancian en el modelo `Main` y se conectan con `connect()`, reproduciendo exactamente el grafo del canvas: Generada la estructura del paquete, el exportador convierte las coordenadas del canvas Java al sistema de referencia de OpenModelica.

4.7.2. Transformación de coordenadas

El sistema de coordenadas Java (píxeles, origen arriba-izquierda, eje Y hacia abajo) debe transformarse al sistema Modelica (unidades abstractas, origen centrado, eje Y hacia arriba), o las posiciones de los componentes en el diagrama Modelica no coincidirían con su disposición visual en el canvas. El Listado 4.5 muestra los métodos `toMoX()` y `toMoY()` que implementan esta transformación lineal de escala y origen:

Código 4.5: Transformación de coordenadas Java a Modelica: cambio de escala y origen.

```
1 private double toMoX(int javaX) {  
2     return (javaX - OFFSET_X) / SCALE_X;  
3 }  
4 private double toMoY(int javaY) {  
5     return (OFFSET_Y - javaY) / SCALE_Y; // eje Y invertido  
6 }  
7 // SCALE_X = SCALE_Y = 2.5 (calibrados empíricamente)
```

Los factores de escala $SCALE_X = SCALE_Y = 2.5$ y los offsets `OFFSET_X`, `OFFSET_Y` se calibraron empíricamente para que las posiciones en OpenModelica coincidan visualmente con las del editor Java.

4.7.3. Ausencia de validación sintáctica en el exportador

`ExportadorModelica` no valida la sintaxis de los identificadores ni el formato de los valores numéricos antes de escribirlos: escribe directamente lo que encuentra en el mapa de parámetros de cada `ComponenteModelica`, sin ningún control de formato. Esta responsabilidad no recae en ninguna otra clase del sistema tampoco: `ComponenteModelica.setParametro()` (Sec. 4.5) también asigna el valor directamente sin comprobación alguna, salvo por un único chequeo trivial: el diálogo de edición de parámetros de `CanvasModelica` (Sec. 4.4.1) rechaza únicamente los campos vacíos antes de llamar a `setParametro()`, sin comprobar que el valor introducido sea numérico ni que respete la sintaxis de un identificador Modelica válido. En la práctica, un valor no numérico introducido manualmente (por ejemplo, texto en un campo destinado a un parámetro

como C_0) se propaga sin detección hasta el archivo `.mo` generado, donde solo se manifestaría como un error de compilación en OpenModelica. Esta ausencia de validación defensiva en la capa de exportación es una limitación reconocida del sistema (véase Sec. 7.2).

4.7.4. Generación de documentación HTML embebida

El método `generarDocumentacionHTML()` escribe código HTML que se embebe dentro del archivo Modelica como documentación del modelo. Esta documentación se renderiza en la pestaña Documentation de OpenModelica y no se limita a texto descriptivo: el exportador *calcula en el momento de exportar* los valores de equilibrio del modelo y los embebe ya resueltos, apoyándose en seis métodos auxiliares organizados en pares teórico/con-datos-reales:

- `extraerParametrosISLM()` / `extraerParametrosSolow()`: leen los parámetros ($C_0, c, I_0, b, G, M_s, k, h$ / s, n, δ, α) directamente de los componentes del canvas.
- `generarEquilibrioISLM()`: resuelve Y^*, r^*, C, I con las fórmulas cerradas de la Sec. 5.6, verifica $Y = C + I + G$ y añade los multiplicadores fiscal y monetario.
- `generarEstadoEstacionarioSolow()`: aplica $k^* = [s/(n+\delta)]^{1/(1-\alpha)}$, calcula la regla de oro del capital e informa si el modelo presenta subahorro o sobreahorro.
- `calcularPromediosTimeTable()`: si el modelo usa datos reales del Banco Mundial, promedia la serie histórica inyectada.
- `generarResultadosISLMConDatos()` / `generarResultadosSolowConDatos()`: repiten los cálculos anteriores sustituyendo los parámetros teóricos por esos promedios históricos, cuando hay datos reales disponibles.

El resultado es que el archivo `.mo` exportado documenta, sin necesidad de simular, cuál será aproximadamente el equilibrio del modelo cargado.

4.7.5. Integración de datos reales: componentes TimeTable

Cuando un componente tiene datos históricos inyectados (campo `datosHistoricos` no nulo), el método `generarDefinicionesDatosReales()` genera una clase especial que encapsula un bloque `Modelica.Blocks.Sources.TimeTable`. El Listado 4.6 muestra un ejemplo real: los datos históricos del Banco Mundial se convierten en una tabla de pares (año, valor) que OpenModelica interpola linealmente durante la simulación. Los puntos extremos en los años 0 y 3000 aseguran que la simulación no falle si el tiempo de simulación excede el rango histórico disponible:

Código 4.6: Código generado con datos históricos del Banco Mundial: bloque `TimeTable` con interpolación lineal.

```

1 model GastoPublico_DATA_GastoPublico_1
2 Modelica.Blocks.Sources.TimeTable datos(table=[
3 0, 150.5; // valor inicial constante
4 2000, 150.5;
5 2001, 155.2;
6 ...
7 2023, 220.3;
8 3000, 220.3 // valor final constante
9 ]);
10 RealOutput G_out;
11 equation
12 G_out = datos.y; // interpolación lineal
13 end GastoPublico_DATA_GastoPublico_1;

```

La tabla se extiende automáticamente con puntos extremos en los años 0 y 3000 para asegurar que la simulación no falle si el tiempo excede el rango de datos. `TimeTable` realiza interpolación lineal entre los puntos durante la simulación, proporcionando valores continuos.

El exportador transforma el modelo hacia código textual. La operación inversa, reconstruir el modelo gráfico a partir de un archivo `.mo` existente, es responsabilidad del Parseador-Modelica, descrito en la siguiente sección.

4.8 Parseador de archivos Modelica. ParseadorModelica

`ParseadorModelica` implementa el proceso inverso al `ExportadorModelica`: lee archivos `.mo` previamente generados y reconstruye los componentes económicos y sus conexiones en el lienzo del editor. Su función es esencial para habilitar la edición iterativa de modelos ya guardados, cerrando el ciclo edición → exportación → importación. La sección se organiza en tres apartados: los patrones de extracción empleados (Sec. 4.8.1), la transformación inversa de coordenadas (Sec. 4.8.2) y la integración con el sistema del editor (Sec. 4.8.3). El análisis se realiza mediante **expresiones regulares** que reconocen los patrones de la sintaxis Modelica generada por el propio exportador.

Esta decisión de diseño se sitúa entre varias alternativas posibles para implementar un parseador (Aho et al., 2006; Grune and Jacobs, 2008).

Un analizador de descenso recursivo, o una gramática formal LL/LR generada con herramientas como ANTLR o JavaCC, ofrecerían mayor robustez ante variaciones sintácticas y facilitarían la detección temprana de errores estructurales, a costa de una implementación más compleja y de mantener una gramática completa del lenguaje Modelica. Dado que `ParseadorModelica` solo necesita reconocer archivos `.mo` generados por el propio `ExportadorModelica` (con una sintaxis fija y controlada, no el lenguaje Modelica completo), un conjunto acotado de expresiones regulares resulta suficiente y considerablemente más ligero de implementar y mantener que un analizador sintáctico completo. Esta elección es coherente con la limitación documentada más abajo: al no cubrir la gramática general de Modelica, el parseador no reconoce archivos `.mo` escritos manualmente con estructuras distintas a las que el exportador produce.

La Figura 4.7 resume la arquitectura de parseo basada en las clases `Pattern` y `Matcher` de Java.

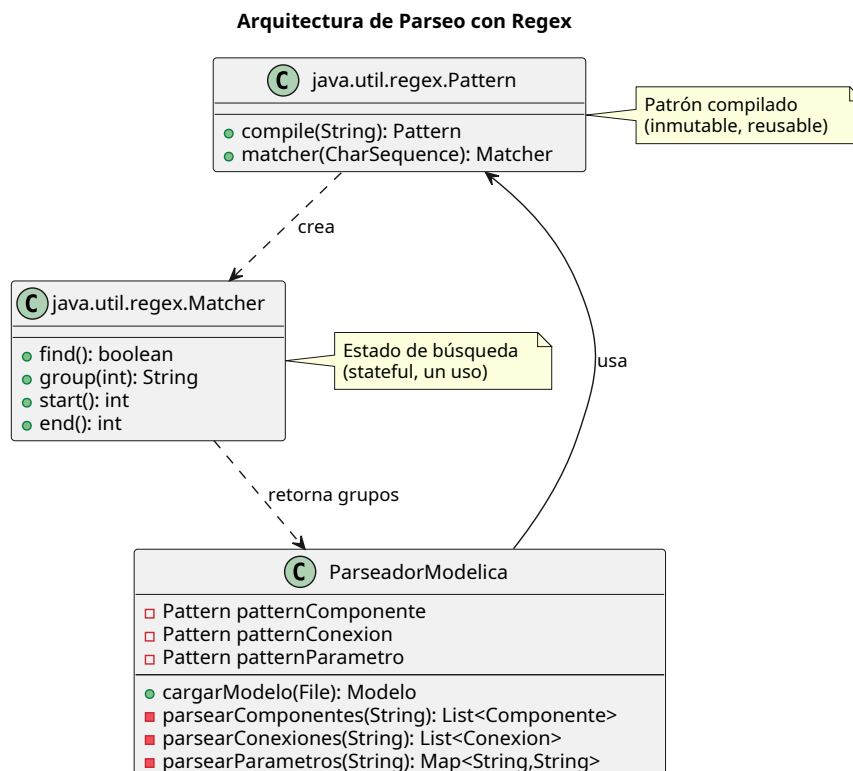


Figura 4.7: Arquitectura de parseo con expresiones regulares: clases `Pattern`, `Matcher` y `ParseadorModelica`.

4.8.1. Patrones de extracción

El parseador localiza cada elemento del archivo `.mo` mediante seis expresiones regulares distintas, encadenadas en dos pasadas sobre el texto

Una limitación del parseador actual es que solo reconoce los patrones generados por el propio exportador del editor. Archivos .mo creados manualmente con sintaxis diferente (uso de variables locales, ecuaciones estructuradas de forma no estándar) pueden no parsearse completamente. Esta limitación está documentada y es coherente con el objetivo principal del sistema: el ciclo de vida editor $\rightarrow$ OpenModelica $\rightarrow$ editor.

Una segunda limitación, más específica, afecta a la biblioteca Doughnut: el método `deducirBiblioteca()`, encargado de asignar cada componente reimportado a su biblioteca de origen, solo distingue explícitamente **Solow** de **Economic** (con un caso residual para **Electrical**, remanente de las pruebas iniciales del editor con componentes eléctricos genéricos antes de su especialización económica). Al no existir un caso explícito para **Doughnut**, un componente de esa biblioteca reimportado desde un archivo .mo se clasificaría por defecto como **Economic**. Esta limitación es coherente con el estado de la biblioteca Doughnut descrito en la Sec. 4.9.5: al no estar validada con el mismo rigor que IS-LM y Solow, tampoco su reimportación ha sido objeto de corrección específica todavía.

Con el ciclo exportación-importación completo, la arquitectura dispone de dos funcionalidades transversales que mejoran la calidad del modelado: el validador de conexiones y la integración de datos reales. Estas se describen a continuación.

4.9 Sistema de validación de conexiones.

ValidadorConectores

ValidadorConectores implementa la lógica de validación en dos niveles: sintáctico (compatibilidad de tipos Modelica) y semántico (corrección económica según el modelo). Esta doble validación convierte al editor en una herramienta educativa activa: no solo previene errores técnicos, sino que guía al usuario hacia conexiones económicamente coherentes con la teoría macroeconómica.

La Figura 4.8 recoge el diagrama de clases de **ValidadorConectores**.

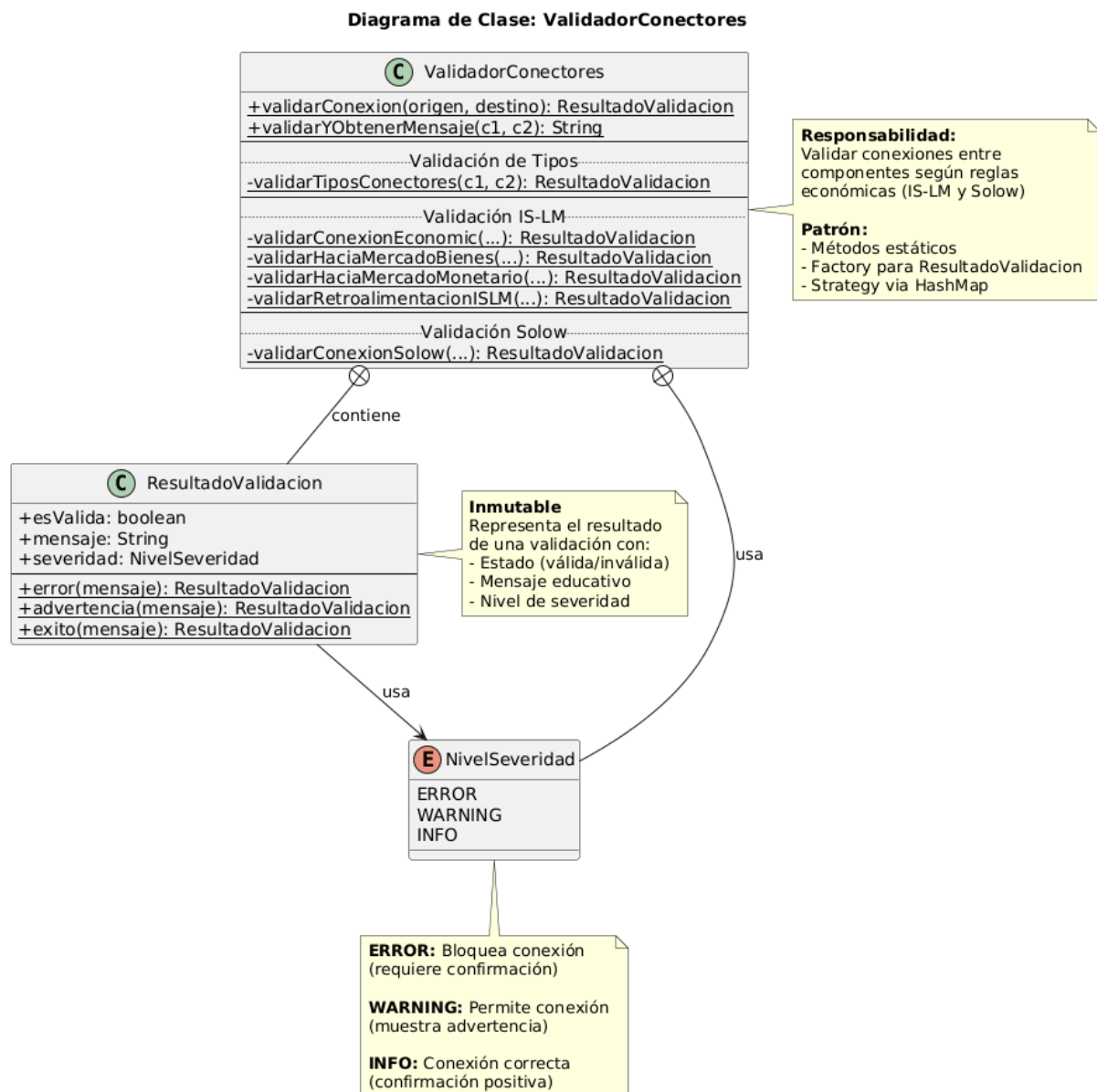


Figura 4.8: Diagrama de clases de ValidadorConectores.

El validador se estructura en seis apartados: la clase de retorno **ResultadoValidacion** (Sec. 4.9.1), la validación sintáctica de tipos (Sec. 4.9.2), las reglas semánticas IS-LM (Sec. 4.9.3), las reglas semánticas Solow (Sec. 4.9.4), las reglas semánticas Doughnut (Sec. 4.9.5) y la integración con el canvas (Sec. 4.9.6). Las reglas IS-LM y Solow validan modelos que superan además la validación analítica formal del Capítulo 5; las reglas Doughnut, en cambio, validan una biblioteca que es todavía una prueba de concepto (Sec. 4.5) sin ese mismo nivel de contraste analítico, y se presentan aquí como avanzadilla del trabajo futuro del Capítulo 7.

4.9.1. Estructura de ResultadoValidacion

Todas las verificaciones devuelven un objeto `ResultadoValidacion` con tres campos: `esValida` (boolean que indica si se puede crear la conexión), `mensaje` (texto explicativo para el usuario) y `severidad` (enum con tres valores: `INFO`, `WARNING`, `ERROR`). Este diseño evita el uso de excepciones para el control de flujo normal y proporciona mensajes descriptivos con la razón económica del rechazo y la alternativa correcta. Sobre esta estructura, el validador aplica primero reglas de compatibilidad de tipos y después conocimiento experto de los modelos económicos.

4.9.2. Validación sintáctica de tipos

El método `validarTiposConectores()` aplica las reglas básicas de Modelica:

- **Input-Input:** `ERROR`, dos entradas no pueden conectarse; ninguna genera el valor.
- **Output-Output:** `ERROR`, dos salidas entran en conflicto al imponer valores distintos.
- **Output-Input / Input-Output:** `INFO`, estructura causal correcta; el valor fluye de salida a entrada.

Superada la validación de tipos, el sistema comprueba la coherencia económica de la conexión según el dominio del modelo.

4.9.3. Validación semántica IS-LM

El método `validarConexionEconomic()` orquesta cinco comprobaciones en secuencia, delegando cada una en un método auxiliar: `validarHaciaMercadoBienes()` exige que C , I y G entren en `MercadoBienes` por sus puertos correctos; `validarHaciaMercadoMonetario()` hace lo propio para M_d y M_s hacia `MercadoMonetario`; dos comprobaciones directas verifican que `MercadoBienes.Y_out` solo se conecte a puertos `Y_in` y que `MercadoMonetario.r_out` solo se conecte a puertos `r_in`; y `validarRetroalimentacionISLM()` comprueba la dirección inversa, que las entradas `Y_in` y `r_in` de los componentes reciban su valor precisamente de esos mismos puertos y no de otro origen. Las tres primeras comprobaciones agrupan sus reglas en un `HashMap`: esta estructura facilita añadir nuevas reglas sin tocar las existentes (coherente con el principio OCP) y evita cadenas largas de `if-else` anidados. El Listado 4.7 muestra un extracto representativo de `validarHaciaMercadoBienes()`: cada entrada del `HashMap` asocia un tipo de componente origen con la lista de conexiones económicamente

válidas, y el mensaje devuelto explica al usuario la razón económica del error:

Código 4.7: Extracto de reglas semánticas IS-LM: HashMap de conexiones válidas con mensajes educativos.

```

1 // Reglas C, I, G -> MercadoBienes
2 Map<String, String> reglas = new HashMap<>();
3 reglas.put("FuncionConsumo.C_out", "C_in");
4 reglas.put("FuncionInversion.I_out", "I_in");
5 reglas.put("GastoPublico.G_out", "G_in");
6 String clave = tipoOrigen + "." + nombreOrigen;
7 if (reglas.containsKey(clave)) {
8     String entradaEsperada = reglas.get(clave);
9     if (!tipoDestino.equals("MercadoBienes") ||
10         !nombreDestino.equals(entradaEsperada)) {
11         return ResultadoValidacion.advertencia(
12             "ADVERTENCIA: " + variable +
13             " debe conectarse a MercadoBienes." + entradaEsperada +
14             ". Razón: En IS-LM,  $Y = C + I + G$ .");
15     }
16 }

```

El modelo Solow dispone de reglas análogas que verifican la integridad del ciclo de acumulación de capital.

4.9.4. Validación semántica Solow

El método `validarConexionSolow()` verifica las cuatro relaciones directas del ciclo de crecimiento ($Y \rightarrow I \rightarrow K \rightarrow Y$, es decir, producción $\rightarrow$ inversión $\rightarrow$ capital $\rightarrow$ producción; y $L \rightarrow Y$, fuerza laboral $\rightarrow$ producción). La más crítica es el cierre del bucle `Solow_Produccion.Y_out  $\rightarrow$  Solow_Inversion.Y_in`: sin esta retroalimentación, el modelo pierde su propiedad de crecimiento endógeno del capital, degradándose a una simulación sin convergencia al estado estacionario. Dos reglas adicionales regulan el componente `Solow_Datos` (Sec. 4.5): su salida solo puede conectarse a entradas de tasa (`s_tasa`, `n_tasa`), y cualquier entrada de tasa debe recibir su valor exclusivamente de `Solow_Datos`, centralizando así el origen de los parámetros calibrables del modelo.

4.9.5. Validación semántica Doughnut

El método `validarConexionDoughnut()` (90 líneas) se ejecuta con prioridad sobre las reglas IS-LM y Solow: si el origen o el destino de la conexión pertenece a la biblioteca Doughnut, sus reglas se aplican independientemente del resto. Cada uno de los tres bloques acepta una única entrada tipada procedente de un componente IS-LM concreto (`Doughnut_RentaMin` y `Doughnut_RecursoMax` desde `MercadoBienes.Y_out`, `Doughnut_EducMin` desde `GastoPublico.G_out`) y expone una única salida, `ok_out`, con un aviso explícito si el usuario intenta realimentarla de vuelta a un componente IS-LM o Solow: al ser un indicador de evaluación (0 o 1), no debe alterar la dinámica del modelo económico.

A diferencia de las reglas IS-LM y Solow, que validan modelos ya contrastados analíticamente en el Capítulo 5, las reglas Doughnut validan una biblioteca que carece de esa misma validación formal. Su función actual es garantizar la coherencia estructural de las conexiones dentro del editor, no confirmar la corrección numérica de un modelo todavía sin contrastar; deben entenderse, por tanto, como una avanzadilla del trabajo futuro descrito en el Capítulo 7, no como una validación al mismo nivel de madurez que IS-LM o Solow.

4.9.6. Integración con el canvas

Las tres estrategias de validación (IS-LM, Solow y Doughnut) se activan en el canvas mediante el mecanismo descrito a continuación. El editor no delega esta lógica en una interfaz de *listener* independiente: `CanvasModelica.intentarConectar()` invoca directamente `ValidadorConectores.validarConexion()` cada vez que el usuario intenta crear una conexión. El Listado 4.8 resume ese flujo: a diferencia de un validador que bloquea de forma irreversible, el editor deja siempre la decisión final en manos del usuario, incluso ante un ERROR:

Código 4.8: Integración del validador en CanvasModelica.intentarConectar(): el usuario puede forzar incluso una conexión con ERROR.

```

1 ResultadoValidacion validacion = ValidadorConectores.validarConexion(c1, c2
  ↪ );
2 log(ValidadorConectores.validarYObtenerMensaje(c1, c2));
3
4 if (!validacion.esValida) {
5     int opcion = JOptionPane.showConfirmDialog(this,
6         validacion.mensaje + "\n\n¿Deseas crear la conexión de todas formas
  ↪ ?",
7         "Conexión No Válida - ERROR", JOptionPane.YES_NO_OPTION);
8     if (opcion != JOptionPane.YES_OPTION) { return; } // cancelada
9     // en otro caso: conexión forzada, advertencia registrada en el log
10 }
11 if (validacion.esValida && validacion.severidad == NivelSeveridad.WARNING)
  ↪ {
12     JOptionPane.showMessageDialog(this, validacion.mensaje + "...",
13         "Advertencia - Conexión Subóptima", JOptionPane.WARNING_MESSAGE);
14 }
15 // la conexión se crea a continuación en ambos casos

```

Un ERROR muestra un diálogo de confirmación que pregunta explícitamente si el usuario desea crear la conexión de todas formas; si acepta, la conexión se crea igualmente y solo queda registrada una advertencia en el log de que podría producir errores al simular en OpenModelica. Las conexiones con nivel WARNING o INFO no piden confirmación: se muestra un aviso informativo y la conexión se crea sin más trámite. En ningún caso el validador impide de forma irreversible una conexión: prioriza la autonomía del usuario sobre la corrección estricta del modelo, coherente con el propósito educativo del editor descrito al inicio de esta sección.

Con el validador garantizando la coherencia económica del modelo, la siguiente funcionalidad extiende la capacidad del editor hacia el mundo real: la integración con APIs de datos históricos.

4.10 Integración con fuentes de datos externas.

ServicioDatosEconomicos

La clase `ServicioDatosEconomicos` proporciona acceso a datos macroeconómicos reales desde APIs públicas (World Bank, 2026), transformando los modelos de herramientas puramente teóricas en instrumentos de análisis aplicado. Abstrae la complejidad de las peticiones HTTP, el parsing de respuestas JSON y la gestión de errores de red, ofreciendo una interfaz simple para obtener series históricas que alimentan los modelos IS-LM y Solow. La sección se organiza en dos apartados: la arquitectura interna del servicio de descarga (Sec. 4.10.1) y la gestión de errores y optimización de rendimiento (Sec. 4.10.2). La Tabla 4.8 recoge los indicadores del Banco Mundial utilizados (World Bank, 2024).

Tabla 4.8: Indicadores del Banco Mundial utilizados por `ServicioDatosEconomicos`.

Indicador WB	Variable del modelo	Descripción
NE.CON.GOVT.CD	IS-LM: Gasto público G	Government consumption, current USD
FM.LBL.BMNY.GD.ZS	IS-LM: Oferta monetaria M_s	Broad Money M3, % of GDP
NE.GDI.TOTL.ZS	Solow: Tasa de ahorro s	Gross savings, % of GDP
SP.POP.GROW	Solow: Crec. poblacional n	Population growth, annual %
NY.GDP.MKTP.CD	Referencia: PIB	GDP, current USD
FP.CPI.TOTL.ZG	Referencia: Inflación	Consumer price index, annual %
SL.UEM.TOTL.ZS	Referencia: Desempleo	Unemployment, % of labour force

Nota sobre integración con el INE. En una fase inicial del proyecto se planificó la integración con la API del Instituto Nacional de Estadística (<https://servicios.ine.es/wstempus/>) para disponer de series históricas específicas de España. Sin embargo, durante la implementación se comprobó que los datos de España relevantes para los modelos IS-LM y Solow están disponibles con cobertura completa en el Banco Mundial, que los descarga mediante los mismos indicadores que para cualquier otro país. Por esta razón, la integración con el INE no se implementó finalmente: España se trata como cualquier otro país de los más de 200 disponibles en el Banco Mundial, con el código ISO ES.

La Figura 4.9 muestra la arquitectura de procesamiento JSON del servicio y el método *anchor-based* de extracción de pares año-valor.

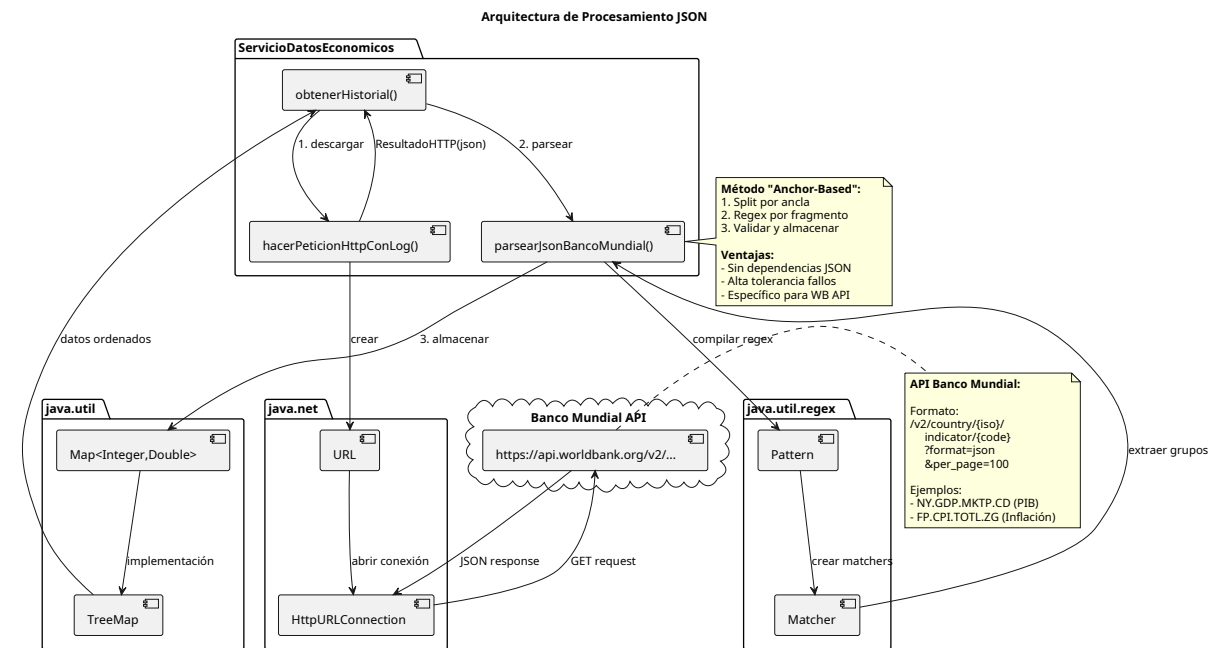


Figura 4.9: Arquitectura de procesamiento JSON del servicio de datos económicos (método *anchor-based*).

4.10.1. Arquitectura interna del servicio

El servicio se compone de cinco módulos internos:

- **Repositorio de países.** Diccionario de 35 naciones con sus códigos ISO, organizado geográficamente por continentes.
- **Motor de descarga HTTP.** Peticiones GET con reintentos (hasta 3 intentos) con esperas exponenciales, *timeout* de 15s para conexión y 30s para lectura.
- **Parser JSON *anchor-based*.** Extrae pares año-valor mediante *regex* sobre anclas del JSON. Más robusto que parsers estrictos: tolera campos adicionales y respuestas parcialmente malformadas.
- **Sistema de *logging* por callbacks.** Informa del progreso de descarga sin bloquear el EDT de Swing.
- **Inyector de datos.** El método `inyectarDatosEnComponente()` asocia la serie temporal descargada a un `ComponenteModelica` específico, marcándolo para que el exportador genere un bloque `TimeTable`.

La secuencia completa de una descarga (con sus reintentos y manejo de errores) se representa en la Figura 4.10.

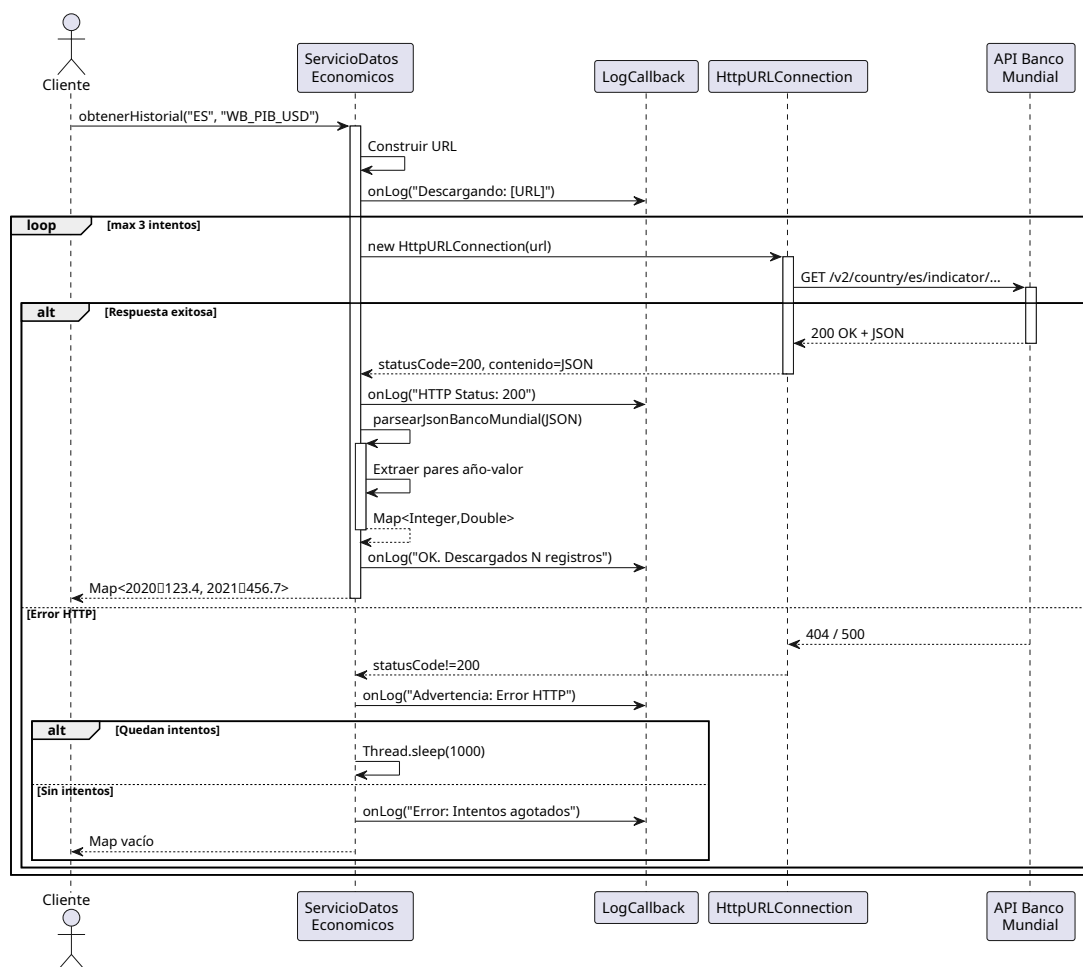


Figura 4.10: Secuencia de descarga de datos del Banco Mundial: reintentos, manejo de errores y *logging*.

El comportamiento del servicio frente a fallos de red y sus optimizaciones de rendimiento se detallan en el siguiente apartado.

4.10.2. Gestión de errores y rendimiento

Todas las operaciones críticas están protegidas con try-catch apropiados. Las descargas se ejecutan en hilos secundarios para no bloquear el EDT de Swing, y las actualizaciones del canvas se despachan mediante `SwingUtilities.invokeLater()`. Las series temporales solo se descargan cuando el usuario lo solicita explícitamente (descarga bajo demanda), y los conectores cachean su posición absoluta para evitar recálculos innecesarios. Con estas optimizaciones, el editor maneja fluidamente modelos de hasta 50 componentes con 100 conexiones en hardware moderno.

Una vez generado y exportado el código, el usuario puede querer revisarlo sin salir del editor. Esta necesidad motivó el visor de código integrado que se describe a continuación.

4.11 Visor de código Modelica integrado.

DialogoVerCodigo

`DialogoVerCodigo` es una ventana modal que muestra el código Modelica generado con resaltado básico de sintaxis. Está implementada íntegramente con la biblioteca estándar de Java (JDK), sin dependencias externas, lo que garantiza portabilidad y simplifica la gestión de dependencias. La sección se describe en dos apartados: el flujo de datos desde el canvas hasta la vista (Sec. 4.11.1) y el mecanismo de resaltado de sintaxis Modelica (Sec. 4.11.2).

4.11.1. Flujo de datos del visor

El visor obtiene el código a partir del modelo en pantalla siguiendo un flujo lineal, desde el lienzo hasta el texto resaltado. `EditorModelicaMultiDominio.verCodigoModelica()` (Sec. 4.3.4) crea un `ExportadorModelica`, invoca su método `generarCodigo()` (el mismo de la Sec. 4.7) y pasa la cadena resultante al constructor de `DialogoVerCodigo`, que únicamente se encarga de resaltarla y mostrarla; el propio diálogo no genera código ni conoce `ExportadorModelica`. El Listado 4.9 resume ese recorrido en pseudocódigo:

Código 4.9: Flujo de datos del visor de código (pseudocódigo): del canvas al texto resaltado.

```
1 Canvas -> ExportadorModelica.generarCodigo()
2 v
3 StringBuilder código = new StringBuilder()
4 código.append("package NombreModelo\n")
5 código.append(" model Componente1...\n")
6 código.append(" model Main...\n")
7 código.append(" connect(...);\n")
8 código.append("end NombreModelo;\n")
9 v
10 return código.toString() // NO escribe archivo en disco
11 v
12 DialogoVerCodigo.mostrar(codigoGenerado)
13 v
14 JTextArea muestra el código con resaltado de sintaxis
```

El código se genera en memoria sin escribir ningún archivo temporal en disco, lo que mejora el rendimiento y evita artefactos en el sistema de archivos del usuario.

4.11.2. Resaltado de sintaxis

El método `aplicarResaltadoSintaxis()` identifica palabras clave Modelica y les aplica colores mediante `StyleDocument` de Swing: azul para palabras estructurales (`model`, `end`, `equation`, `parameter`, `Real`, `input`, `output`), verde para instrucciones de conexión (`connect`, `import`, `annotation`) y rojo para comentarios (líneas que empiezan por `//`). Este resaltado mejora significativamente la legibilidad del código generado y facilita el aprendizaje de la correspondencia entre representación gráfica y código textual.

La ventana incluye tres botones de acción: Copiar (copia al portapapeles del sistema mediante `java.awt.datatransfer.Clipboard`), Exportar (abre el diálogo de guardado para escribir el archivo `.mo`) y Cerrar.

Con el visor, el conjunto de funcionalidades del editor queda completo. La siguiente sección muestra cómo la arquitectura modular facilita la extensión del sistema con nuevos componentes sin modificar el código existente.

4.12 Adición de nuevos componentes. Mecanismo de extensibilidad

Esta sección ilustra en la práctica el principio Abierto/Cerrado (OCP) introducido en la Sec. 4.2: gracias al diseño MVC y a la correcta separación de responsabilidades, añadir un nuevo tipo de componente al editor requiere modificaciones localizadas en exactamente cuatro archivos Java, sin afectar al resto del sistema: está *abierto a extensión* sin necesidad de *modificar código existente*. A continuación se ilustra el proceso con el ejemplo del componente `Impuestos (T)` para el modelo IS-LM.

Paso 1: Definición del componente en `ComponenteModelica.java`

Se añade un nuevo caso en el *switch* del método `inicializar()` con el símbolo, color, parámetros y conectores del nuevo tipo:

Código 4.10: Paso 1: ComponenteModelica.java — definición del componente Impuestos.

```

1 case "Impuestos":
2     simbolo = "T";
3     colorFondo = new Color(100, 149, 237); // Azul
4     addConector("T_data_in", "RealInput", -50, 0);
5     addConector("T_out", "RealOutput", 50, 0);
6     parametros.put("tasa_impositiva", "0.25");
7     parametros.put("T_autonomo", "10000");
8     break;

```

Paso 2: Registro en la paleta en PaletaComponentes.java

Se añade una entrada para que el componente aparezca en el panel lateral del editor:

Código 4.11: Paso 2: PaletaComponentes.java — entrada en la paleta.

```

1 modelo.addElement(new ComponenteInfo(
2     "Economic", // biblioteca
3     "Impuestos", // tipo
4     "Impuestos (T)", // etiqueta visible para el usuario
5     false)); // false = componente, no título de sección

```

Paso 3: Regla semántica en ValidadorConectores.java

Se añade la regla de validación correspondiente, incluyendo el mensaje educativo con la razón económica:

Código 4.12: Paso 3: ValidadorConectores.java — regla semántica.

```

1 if (tipoOrigen.equals("Impuestos") &&
2     nombreOrigen.equals("T_out")) {
3     if (!tipoDestino.equals("FuncionConsumo") ||
4         !nombreDestino.equals("T_in")) {
5         return ResultadoValidacion.advertencia(
6             "ADVERTENCIA: T_out debe ir a FuncionConsumo.T_in. " +
7             "Razon: el ingreso disponible es  $Y_d = Y - T$ .");
8     }
9 }

```

Paso 4: Definición Modelica en ExportadorModelica.java (si el componente tiene ecuación propia)

Se añade una llamada a crearBloque() dentro de generarDefinicionesEconomicas(),

que genera automáticamente el bloque `model ... equation ... end` del componente:

Código 4.13: Paso 4: `ExportadorModelica.java` — ecuación Modelica del componente.

```

1 crearBloque(sb, "Impuestos", "T",
2     "T_out = tasa_impositiva * T_data_in + T_autonomo;",
3     "parameter Real tasa_impositiva=0.25; " +
4     "parameter Real T_autonomo=10000;",
5     "RealInput T_data_in",
6     "RealOutput T_out");

```

La Figura 4.11 resume visualmente el proceso completo: los cuatro pasos descritos, los ficheros y métodos concretos que tocan, y el principio de diseño (OCP) que queda demostrado al no requerir cambios en el resto del sistema.

Mecanismo de extensibilidad: añadir el componente Impuestos en 4 pasos

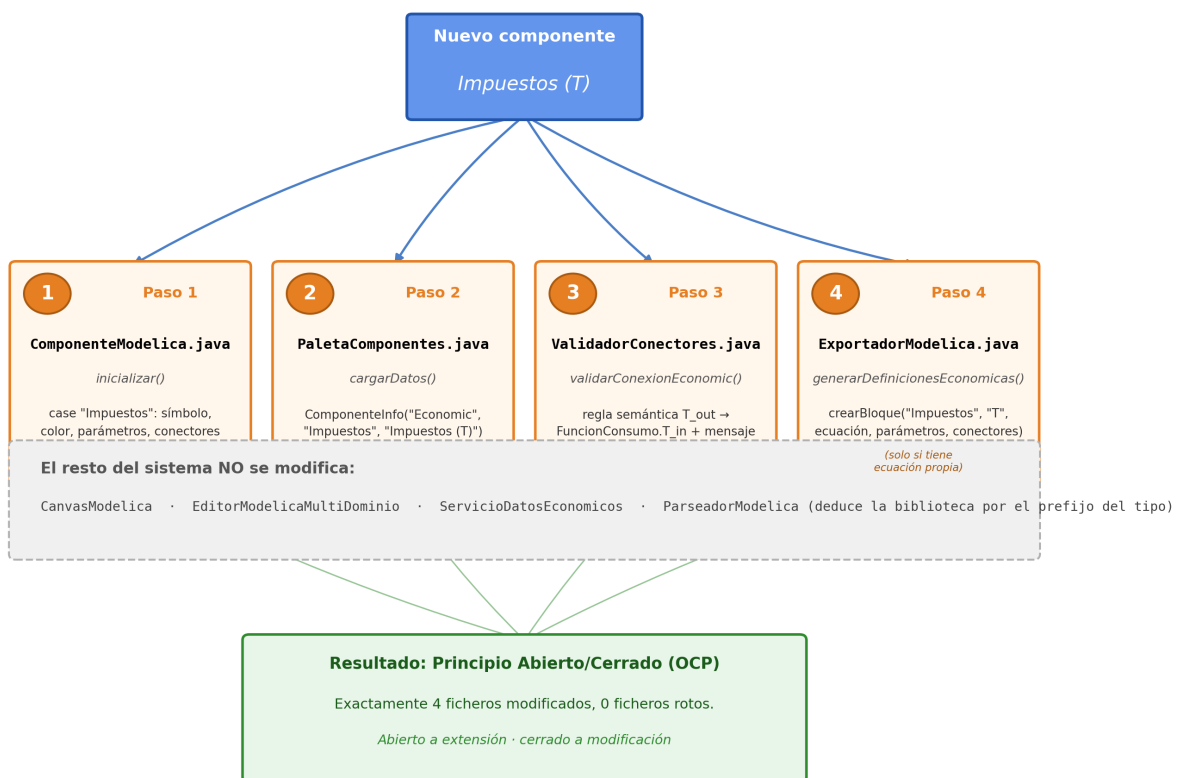


Figura 4.11: Mecanismo de extensibilidad: los cuatro ficheros Java modificados al añadir el componente *Impuestos*, y las partes del sistema que permanecen intactas, ilustrando el principio Abierto/Cerrado (OCP).

Resumen: 4 modificaciones para añadir cualquier nuevo componente

- `ComponenteModelica.java`: añadir case en `inicializar()` con símbolo, color, parámetros y conectores.
- `PaletaComponentes.java`: añadir entrada en `cargarDatos()` para que aparezca en el panel lateral.
- `ValidadorConectores.java`: añadir regla semántica en `validarConexionEconomic()` con el mensaje educativo.
- `ExportadorModelica.java`: añadir llamada a `crearBloque()` en `generarDefinicionesEconomicas()` (solo si el componente tiene ecuación Modelica propia).

El `ParseadorModelica` generalmente no requiere modificaciones, ya que deduce la biblioteca por el prefijo del tipo. El resto del sistema (`CanvasModelica`, `EditorModelicaMultiDominio`, `ServicioDatosEconomicos`) no se toca. Habiendo descrito todos los componentes y el mecanismo de extensibilidad, la sección de conclusiones sintetiza las decisiones arquitectónicas adoptadas y su impacto en los objetivos del proyecto.

4.13 Conclusiones

En este capítulo se ha presentado la arquitectura software completa del editor gráfico Modelica, describiendo sus once módulos principales y los patrones de diseño que los articulan. La arquitectura modular basada en MVC facilita tanto el mantenimiento como la extensibilidad del sistema, permitiendo añadir nuevas funcionalidades (nuevos componentes, nuevos dominios económicos, nuevas fuentes de datos) sin afectar a los módulos existentes.

Los módulos trabajan de forma coordinada siguiendo una cadena funcional clara: el lienzo (`CanvasModelica`) captura y visualiza las interacciones del usuario; los componentes (`ComponenteModelica`, `ConexiónModelica`) encapsulan la lógica de modelado; el validador (`ValidadorConectores`) garantiza la coherencia económica antes de cada modificación; el exportador (`ExportadorModelica`) serializa el modelo a código .mo estándar; y el parseador (`ParseadorModelica`) permite la reedición iterativa de modelos previamente guardados. El servicio de datos (`ServicioDatosEconomicos`) y el visor (`DialogoVerCodigo`) completan el conjunto como funcionalidades avanzadas que enriquecen la experiencia sin añadir complejidad al núcleo.

La separación de responsabilidades siguiendo principios SOLID asegura código mantenible y testeable. En particular, el Principio de Responsabilidad Única (SRP) se respeta: cada clase tiene una razón clara de cambio; el Principio de Abierto-Cerrado (OCP) se materializa en el mecanismo de extensibilidad de cuatro pasos descrito en la Sec. 4.12; y la Inversión de Dependencias (DIP) aparece en el uso de `PropertyChangeListener` como contrato entre subsistemas.

Como limitación a reconocer, el controlador implícito distribuido en forma de listeners hace que la traza de un evento de usuario a través del sistema requiera leer múltiples clases. Esta es una consecuencia inherente del modelo de eventos de Swing y se ha mitigado documentando el flujo de interacción en la Sec. 4.2. En un proyecto de mayor escala, esta arquitectura se beneficiaría de la introducción de un bus de eventos centralizado.

Los componentes y algoritmos presentados en este capítulo implementan la funcionalidad del editor cuya corrección es verificada en el Capítulo 5, donde las pruebas exhaustivas de validación confirman que el código generado es sintácticamente correcto y que los modelos económicos producen los resultados numéricos esperados por la teoría macroeconómica.

Capítulo 5

Validación del Sistema y Pruebas

5.1 Introducción

En este capítulo se valida el resultado del código Modelica generado por el editor, comparando los resultados de compilación y simulación en OpenModelica con los valores calculados analíticamente con lápiz, papel y calculadora. El modelo IS-LM arroja puntos de equilibrio cuando se igualan las ecuaciones IS y LM; el modelo Solow, en cambio, es una función continua (Cobb-Douglas) que converge a largo plazo, lo que ofrece más posibilidades de simulación y análisis gráfico.

La validación se estructura en cuatro niveles progresivos:

Nivel sintáctico. Compilación sin errores en OpenModelica y Dymola.

Nivel numérico. Precisión de los equilibrios calculados frente a los valores teóricos.

Nivel económico. Consistencia de las respuestas a las políticas fiscal y monetaria con la teoría macroeconómica.

Nivel empírico. Ajuste cualitativo a series históricas reales del Banco Mundial.

El capítulo se estructura en doce secciones, organizadas en tres bloques.

Verificación del modelo IS-LM: la Sec. 5.2 lo compila y simula en OpenModelica; las secciones 5.4 y 5.5 prueban la validación de conexiones y la importación de modelos; las secciones 5.6 y 5.7 realizan la validación analítica del equilibrio y de las políticas económicas.

Verificación del modelo de Solow: la Sec. 5.3 lo compila y simula; las secciones 5.8 y 5.9 verifican el estado estacionario y lo validan con datos reales de EE.UU.

Cierre: la Sec. 5.10 valida la extensión de economía circular, la Sec. 5.11 describe las pruebas adicionales en Dymola y la 5.12 resume las conclusiones.

5.2 Verificación del modelo IS-LM. Compilación y simulación en OpenModelica

Esta sección verifica que el editor genera código Modelica correcto para el modelo IS-LM y que OpenModelica puede compilarlo y simularlo hasta alcanzar el equilibrio. Se realizaron las siguientes comprobaciones empleando el editor y OpenModelica:

1. Iniciar el editor mediante `java EditorModelicaMultiDominio`.
2. Hacer doble clic en «**Modelo IS-LM Base**» de la paleta izquierda.
3. Verificar que el modelo aparece completo en el lienzo con todas las conexiones renderizadas.
4. Exportar el modelo mediante **Archivo** → **Generar .mo**.
5. Abrir el archivo generado en OMEdit de OpenModelica.
6. Verificar que compila sin errores.
7. Ejecutar la simulación verificando la convergencia al equilibrio.

Los resultados obtenidos confirman que:

- El código Modelica generado es sintácticamente correcto y OpenModelica puede compilarlo y simularlo.
- Las conexiones entre componentes se traducen correctamente a instrucciones `connect()` en el archivo `.mo`.
- Los parámetros configurados en el editor se exportan con los valores correctos.
- Las anotaciones visuales **Placement** preservan el *layout* gráfico original al reimportar.
- Los archivos `.mo` generados pueden reabrirse en el editor manteniendo toda la información.

La Figura 5.1 muestra el modelo IS-LM creado con el editor. Una vez exportado, el modelo se carga y simula en OMEdit; la Figura 5.2 muestra el resultado, con convergencia al equilibrio.

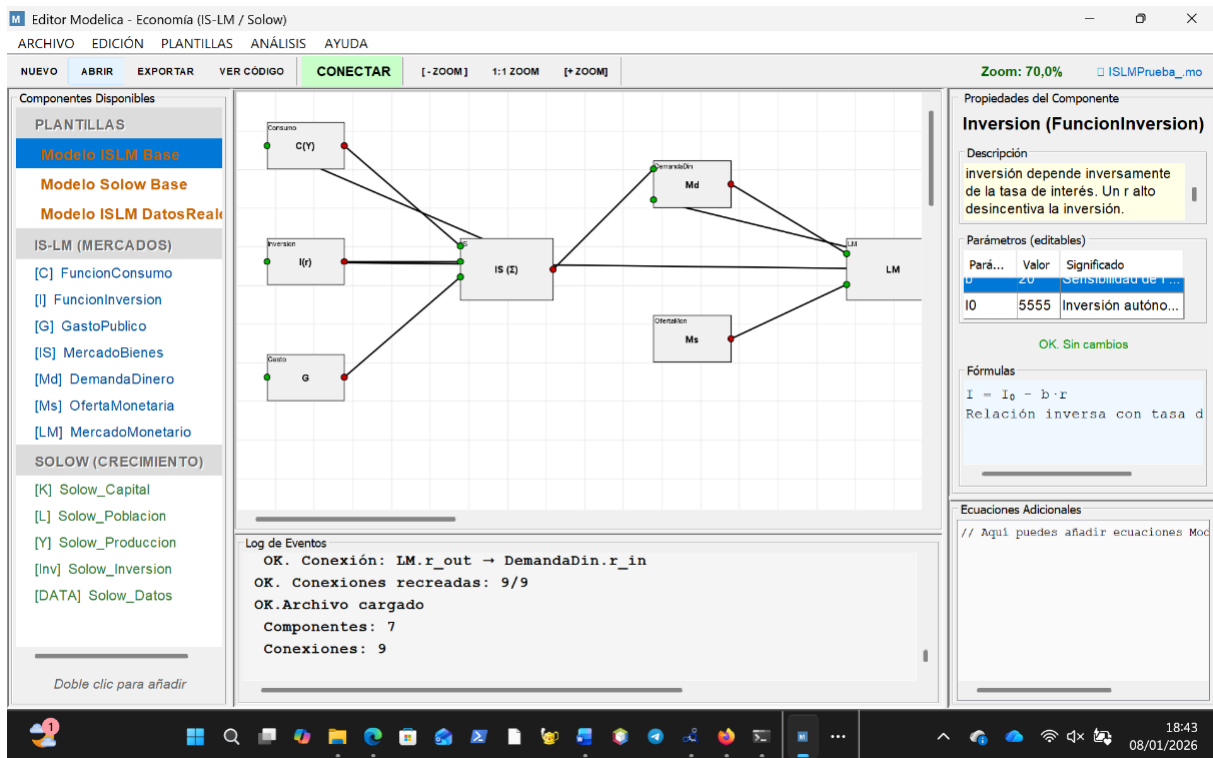


Figura 5.1: Captura del editor gráfico con el modelo IS-LM cargado, mostrando componentes, conexiones y panel de propiedades.

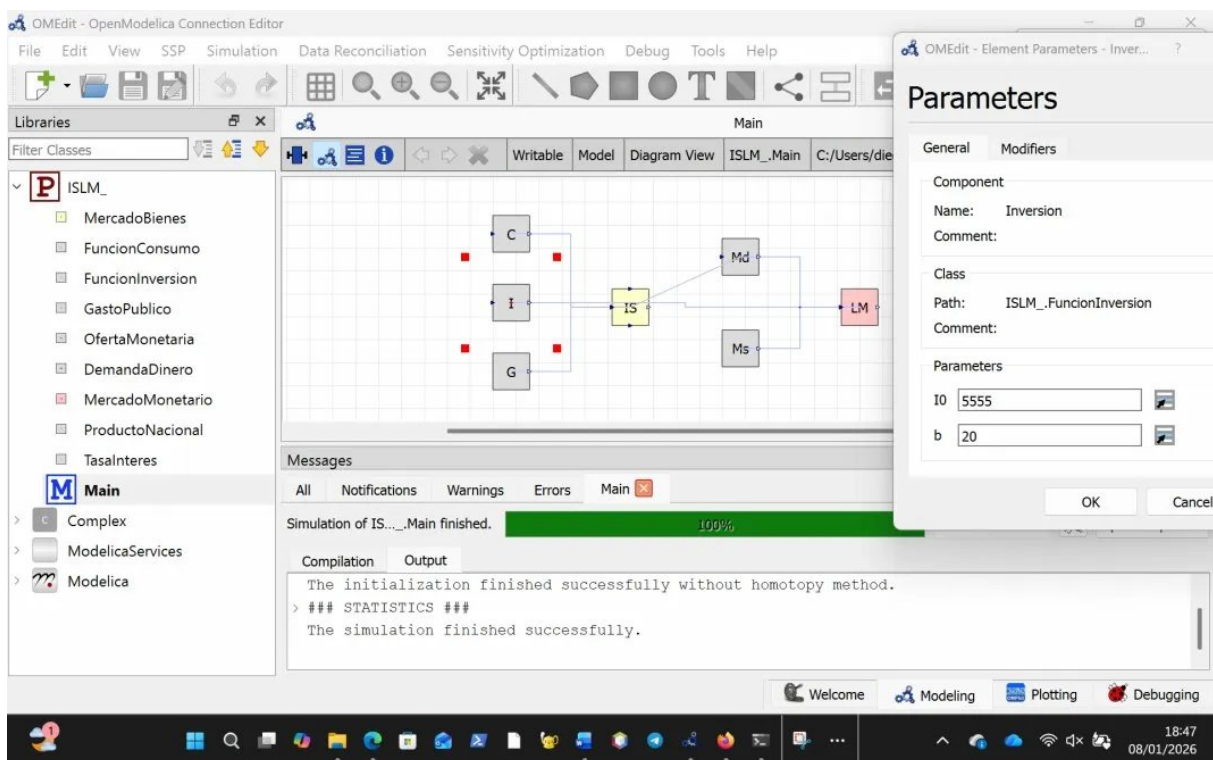


Figura 5.2: Modelo IS-LM en OMEdit con parámetros de FuncionInversion ($i_0=5555$, $b=20$) y simulación completada.

Con el modelo IS-LM compilado y simulado correctamente en OpenModelica, el siguiente

paso repite el procedimiento de verificación para el modelo de Solow.

5.3 Verificación del modelo de Solow. Compilación y simulación en OpenModelica

Las variables del modelo de Solow (s , n , δ , α , K , L) están definidas en la Tabla 3.3 del Cap. 3. Se siguió el mismo procedimiento que para el IS-LM: creación del modelo en el editor, exportación a archivo .mo y carga en OMedit para compilación y simulación. Los resultados confirman que:

- El modelo Solow se crea correctamente con los cuatro componentes (Capital, Producción, Población, Inversión) y el bucle de retroalimentación cerrado $Y \rightarrow I \rightarrow K \rightarrow Y$ (producción $\rightarrow$ inversión $\rightarrow$ capital $\rightarrow$ producción).
- El exportador genera código Modelica sintácticamente correcto que OpenModelica compila sin errores.
- La simulación reproduce la convergencia al estado estacionario esperado para los parámetros configurados.

Verificada la compilación sin errores de ambos modelos, la siguiente prueba comprueba que el validador semántico rechaza correctamente las conexiones económicamente incorrectas.

5.4 Prueba de validación de conexiones incompatibles

El objetivo de esta prueba es verificar que el editor rechaza conexiones que no son matemática o económicamente correctas. Los componentes y variables referenciados en las reglas IS-LM (C , I , G , Y , r , M_d , M_s) están definidos en la Tabla 3.1 del Cap. 3; los del modelo Solow (K , Y , L , I , s) en la Tabla 3.3 del mismo capítulo. El sistema implementa siete reglas semánticas en `ValidadorConectores.java`, agrupadas en dos bloques según el modelo. La Sec. 5.4.1 describe las cuatro reglas IS-LM, la Sec. 5.4.2 las tres reglas Solow y la Sec. 5.4.3 recoge los resultados de las pruebas.

5.4.1. Reglas del modelo IS-LM

El modelo IS-LM implementa cuatro reglas semánticas que verifican la coherencia de cada conexión con las relaciones causales de la teoría keynesiana:

Regla 1: Sumatorio de demanda agregada ($Y = C + I + G$).

Las salidas de consumo, inversión y gasto público deben conectarse siempre al Mercado de Bienes:

FuncionConsumo.C_out $\rightarrow$ MercadoBienes.C_in

FuncionInversion.I_out $\rightarrow$ MercadoBienes.I_in

GastoPublico.G_out $\rightarrow$ MercadoBienes.G_in

Regla 2: Equilibrio monetario ($M_s = M_d$).

La oferta y demanda de dinero deben ir al Mercado Monetario:

DemandaDinero.Md_out $\rightarrow$ MercadoMonetario.Md_in

OfertaMonetaria.Ms_out $\rightarrow$ MercadoMonetario.Ms_in

Regla 3: Retroalimentación del producto nacional Y.

El producto Y sale del Mercado de Bienes y alimenta los componentes que dependen de él:

MercadoBienes.Y_out $\rightarrow$ FuncionConsumo.Y_in (C depende de Y)

MercadoBienes.Y_out $\rightarrow$ DemandaDinero.Y_in (Md depende de Y)

Regla 4: Retroalimentación del tipo de interés r.

El tipo de interés sale del Mercado Monetario y afecta a inversión y demanda de dinero:

MercadoMonetario.r_out $\rightarrow$ FuncionInversion.r_in (I depende de r)

MercadoMonetario.r_out $\rightarrow$ DemandaDinero.r_in (Md depende de r)

Definidas las cuatro reglas IS-LM, el modelo Solow define sus propias reglas para verificar la integridad del ciclo de crecimiento.

5.4.2. Reglas del modelo Solow

El modelo Solow define tres reglas que verifican la integridad del ciclo de crecimiento endógeno $Y \rightarrow I \rightarrow K \rightarrow Y$ (producción, inversión y capital, respectivamente):

Regla 5: Bucle de crecimiento (ciclo $Y \rightarrow I \rightarrow K \rightarrow Y$).

Solow_Produccion.Y_out $\rightarrow$ Solow_Inversion.Y_in ($I = sY$)

Solow_Inversion.I_out $\rightarrow$ Solow_Capital.I_in ($\dot{K} = I - \delta K$)

Solow_Capital.K_out $\rightarrow$ Solow_Produccion.K_in ($Y = K^\alpha \cdot L^{1-\alpha}$)

Regla 6: Factor trabajo.

La fuerza laboral se conecta desde la componente de población:

Solow_Poblacion.L_out $\rightarrow$ Solow_Produccion.L_in

Regla 7: Parámetros centralizados de datos.

Solow_Datos.out $\rightarrow$ Solow_Inversion.s_tasa (tasa de ahorro)

Solow_Datos.out $\rightarrow$ Solow_Poblacion.n_tasa (crecimiento poblacional)

Aplicadas las siete reglas, la siguiente sección recoge los resultados de las pruebas sobre conexiones correctas e incorrectas.

5.4.3. Resultados de las pruebas de conexiones

La Tabla 5.1 recoge los resultados de las pruebas de validación de conexiones.

Tabla 5.1: Resultados de las pruebas de validación de conexiones.

Conexión probada	Resultado (esperado / obtenido)
MercadoBienes.Y_out $\rightarrow$ FuncionConsumo.Y_in	CORRECTA (Regla 3) $\rightarrow$ CORRECTA, aceptada
Solow_Capital.K_out $\rightarrow$ Solow_Produccion.K_in	CORRECTA (Regla 5) $\rightarrow$ CORRECTA, aceptada
MercadoMonetario.r_out $\rightarrow$ FuncionInversion.r_in	CORRECTA (Regla 4) $\rightarrow$ CORRECTA, aceptada
FuncionConsumo.C_out $\rightarrow$ DemandaDinero.Y_in	INCORRECTA (Regla 1) $\rightarrow$ RECHAZADA, advertencia generada
Output-Output (dos salidas)	ERROR sintáctico $\rightarrow$ RECHAZADA, error mostrado

La Figura 5.3 ilustra la validación de una conexión coherente.



Figura 5.3: Validación de conexión coherente IS-LM.

En todos los casos el validador responde con el comportamiento esperado. Las conexiones incorrectas son rechazadas con mensajes que explican la razón económica y sugieren la alternativa correcta, proporcionando retroalimentación educativa al usuario. Verificada la validación semántica de conexiones, la siguiente prueba comprueba que el ciclo completo de exportación e importación preserva el estado del modelo.

5.5 Prueba de importación de modelo existente

El objetivo es verificar que los modelos exportados pueden reimportarse preservando el layout (posición de los componentes) y los parámetros modificados. El procedimiento seguido fue:

1. Crear un modelo IS-LM básico en el editor.
2. Reposicionar los componentes en una configuración diferente a la predeterminada.
3. Modificar algunos valores de parámetros (por ejemplo, C_0 de 200 a 350, G de 200 a 280).
4. Exportar el modelo a un archivo `test_reimport.mo`.
5. Reiniciar el editor (limpiar el lienzo).
6. Importar el archivo `test_reimport.mo` mediante **Archivo** → **Abrir .mo**.
7. Verificar que los componentes aparecen en las posiciones modificadas y con los parámetros actualizados.

El resultado confirma que el ciclo exportación → edición → importación funciona correctamente: los componentes se importan con los tipos y parámetros preservados, en posiciones

equivalentes a las guardadas. La Figura 5.4 muestra el diálogo de exportación del archivo `test_reimport.mo` que se utiliza para verificar el ciclo completo.

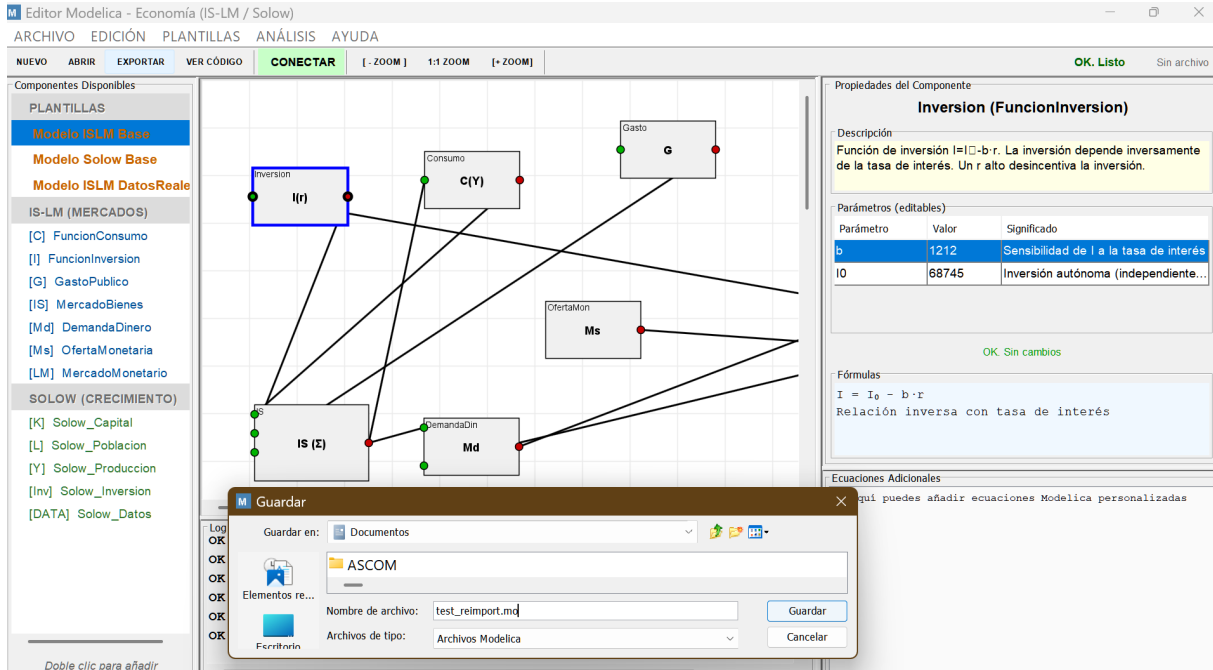


Figura 5.4: Editor gráfico: prueba de exportación.

Esta prueba valida que el ciclo de edición iterativa (exportar, modificar parámetros directamente en el archivo `.mo`, reimportar) es viable para usuarios avanzados que deseen ajustar valores fuera del editor. Superadas las pruebas de compilación, validación de conexiones e importación, la siguiente sección comprueba que los valores numéricos producidos por OpenModelica coinciden con los calculados analíticamente.

5.6 Validación analítica del equilibrio IS-LM

El modelo IS-LM implementado en Modelica requiere validación analítica para garantizar la corrección de su formulación matemática. Esta validación compara los valores de equilibrio calculados teóricamente con los resultados obtenidos en la simulación. Las variables empleadas (C_0 , c , I_0 , b , G , M_s , k , h) están definidas en la Tabla 3.1 del Cap. 3; sus ecuaciones de comportamiento corresponden a las Ecs. (3.1)–(3.4). Sus valores numéricos concretos para esta validación (valores por defecto del editor) son:

Con estos parámetros fijados, las secciones siguientes derivan analíticamente las curvas IS y LM, resuelven el sistema de equilibrio y verifican la coherencia del resultado.

Tabla 5.2: Parámetros del modelo IS-LM base para la validación analítica.

Parámetro	Símbolo	Valor
Consumo autónomo	C_0	200
Propensión marginal a consumir	c	0.8
Inversión autónoma	I_0	300
Sensibilidad inversión al tipo de interés	b	20
Gasto público	G	200
Oferta monetaria	M_s	500
Sensibilidad demanda de dinero a Y	k	0.2
Sensibilidad demanda de dinero a r	h	10

5.6.1. Derivación de la curva IS

La condición de equilibrio en el mercado de bienes Ec. (3.6) establece que el producto debe igualar la demanda agregada ($DA = C + I + G$). Sustituyendo la función de consumo Ec. (3.1), la de inversión Ec. (3.2) y el gasto público exógeno Ec. (3.3), y aplicando los parámetros del escenario base ($C_0 = 200$, $c = 0,8$, $I_0 = 300$, $b = 20$ y $G = 200$; los parámetros monetarios k , h y M_s intervienen en la curva LM, que se deriva en la Sec. 5.6.2):

$$Y = (C_0 + c \cdot Y) + (I_0 - b \cdot r) + G \quad (5.1)$$

$$Y(1 - c) = C_0 + I_0 + G - b \cdot r$$

$$Y(0,2) = 200 + 300 + 200 - 20 \cdot r = 700 - 20 \cdot r \quad (5.2)$$

$$\Rightarrow Y = 3500 - 100 \cdot r \quad \leftarrow \text{curva IS} \quad (5.3)$$

donde el término independiente $3500 = (C_0 + I_0 + G)/(1 - c) = 700/0,2$ y la pendiente $-100 = -b/(1 - c) = -20/0,2$. A mayor tipo de interés, menor inversión y menor producto de equilibrio. Esta ecuación representa todas las combinaciones de Y y r que equilibran el mercado de bienes. El siguiente paso es derivar la curva LM a partir del equilibrio en el mercado monetario.

5.6.2. Derivación de la curva LM

La condición de equilibrio en el mercado monetario Ec. (3.12) iguala la oferta monetaria con la función de demanda de dinero Ec. (3.4):

$$M_s = M_d = k \cdot Y - h \cdot r \quad (5.4)$$

donde $M_s = 500$ es la oferta monetaria y $M_d = 0,2 \cdot Y - 10 \cdot r$ la función de demanda de dinero. Despejando Y :

$$\begin{aligned} 500 &= 0,2 \cdot Y - 10 \cdot r \\ Y &= \frac{500 + 10 \cdot r}{0,2} \end{aligned} \quad (5.5)$$

$$\Rightarrow Y = 2500 + 50 \cdot r \quad \leftarrow \text{curva LM} \quad (5.6)$$

donde el término independiente $2500 = M_s/k = 500/0,2$ y la pendiente $+50 = h/k = 10/0,2$. La pendiente positiva de la curva LM refleja que un mayor tipo de interés libera liquidez que se destina al mercado de bienes, elevando la renta. Con las dos curvas formuladas, se resuelve el sistema simultáneo (Y^*, r^*) .

5.6.3. Resolución del sistema: equilibrio simultáneo

El equilibrio general se alcanza en la intersección $IS = LM$:

$$3500 - 100 \cdot r = 2500 + 50 \cdot r \quad (5.7)$$

$$1000 = 150 \cdot r \rightarrow r^* = 6,67\% \quad (5.8)$$

$$Y^* = 2500 + 50 \cdot (6,67) = 2500 + 333,33 = 2833,33 \quad (5.9)$$

Con el equilibrio resuelto analíticamente, el siguiente paso es verificar que los valores obtenidos son consistentes con las condiciones de equilibrio en ambos mercados.

5.6.4. Verificación del equilibrio

Para confirmar que estos valores representan un equilibrio real, se calculan las variables derivadas:

$$C^* = C_0 + c \cdot Y^* = 200 + 0,8 \cdot (2833,33) = 200 + 2266,67 = 2466,67 \quad (5.10)$$

$$I^* = I_0 - b \cdot r^* = 300 - 20 \cdot (6,67) = 300 - 133,33 = 166,67 \quad (5.11)$$

$$DA^* = C^* + I^* + G = 2466,67 + 166,67 + 200 = 2833,33 = Y^* \quad (5.12)$$

$$M_d^* = k \cdot Y^* - h \cdot r^* = 0,2 \cdot (2833,33) - 10 \cdot (6,67) = 566,67 - 66,67 = 500 = M_s \quad (5.13)$$

confirmando que la demanda de dinero en el equilibrio iguala exactamente la oferta $M_s = 500$, lo que valida la condición de equilibrio del mercado monetario.

Valores de equilibrio IS-LM base

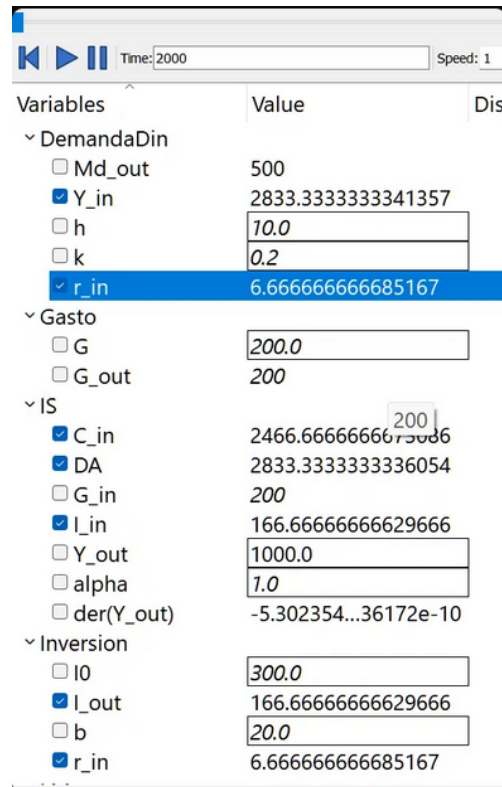
$Y^* = 2833,33$ unidades (Producto Nacional de equilibrio, miles de millones de €/año)

$r^* = 6,67 \%$ (Tipo de interés de equilibrio)

Verificación:

- **Mercado de bienes:** $Y^* = DA^*$ $(2833,33 = 2833,33)$
- **Mercado monetario:** $M_s = M_d^*$ $(500 = 500)$

Los resultados de la simulación en OpenModelica coinciden con estos valores, como muestra la Figura 5.5.



Variables	Value	Dis
DemandaDin		
☐ Md_out	500	
☒ Y_in	2833.3333333341357	
☐ h	10.0	
☐ k	0.2	
☒ r_in	6.666666666685167	
Gasto		
☐ G	200.0	
☐ G_out	200	
IS		
☒ C_in	2466.6666666666666	
☒ DA	2833.3333333336054	
☐ G_in	200	
☒ I_in	166.66666666629666	
☐ Y_out	1000.0	
☐ alpha	1.0	
☐ der(Y_out)	-5.302354...36172e-10	
Inversion		
☐ I0	300.0	
☒ I_out	166.66666666629666	
☐ b	20.0	
☒ r_in	6.666666666685167	

Figura 5.5: Variables de equilibrio IS-LM en OpenModelica.

En concreto, el solver de OpenModelica devuelve $Y^* = 2833,3333333341$ y $r^* = 6,666666666685$ (Figura 5.5), frente a los valores analíticos exactos $8500/3$ y $20/3$ respectivamente: la diferencia relativa es del orden de 10^{-12} , muy por debajo del umbral del 0,1 % citado en la Tabla 5.16.

Con los valores de equilibrio confirmados analítica y numéricamente, la siguiente sección simula el efecto de perturbaciones en la política fiscal y monetaria para verificar que el modelo responde con los multiplicadores teóricos esperados.

5.7 Validación de política fiscal y monetaria

Esta sección simula el efecto de variaciones en la política económica y comprueba si los datos generados por OpenModelica concuerdan con los calculados manualmente. Se analizan dos experimentos: variación del Gasto Público G (política fiscal) y variación de la Oferta Monetaria M_s (política monetaria). Para cada experimento se determinan el equilibrio inicial, la curva afectada por la perturbación, el nuevo equilibrio y los multiplicadores de política resultantes.

5.7.1. Variación de la política fiscal: expansión fiscal (G: 200 → 300)

Se simula un aumento del gasto público de $G = 200$ a $G = 300$, manteniendo constantes los demás parámetros. El análisis prevé dos efectos encadenados:

Efecto directo: el incremento en G aumenta la demanda agregada, elevando el producto nacional Y .

Efecto **crowding-out**: el aumento de Y eleva la demanda de dinero M_d (Ec. (3.4)), lo que presiona al alza el tipo de interés r . A su vez, este incremento de r reduce la inversión privada I (Ec. (3.2)), amortiguando parcialmente el efecto expansivo inicial.

Equilibrio inicial ($G = 200$).

$$\begin{aligned} Y_0^* &= 2833,33 \\ r_0^* &= 6,67 \% \end{aligned} \tag{5.14}$$

Nueva curva IS con $G = 300$.

$$Y = (C_0 + I_0 + G)/(1 - c) - (b/(1 - c)) \cdot r \tag{5.15}$$

$$Y = \frac{200 + 300 + 300}{0,2} - \frac{20}{0,2} \cdot r = 4000 - 100 \cdot r \tag{5.16}$$

Nuevo equilibrio ($G = 300$). Igualando la nueva curva IS con la curva LM (sin cambios):

$$4000 - 100 \cdot r = 2500 + 50 \cdot r \rightarrow 1500 = 150 \cdot r \tag{5.17}$$

$$r_1^* = 10,00 \% \tag{5.18}$$

$$Y_1^* = 2500 + 50 \cdot (10) = 3000 \tag{5.19}$$

5.7.2. Cuantificación del efecto multiplicador fiscal

Calculados el equilibrio inicial y el nuevo equilibrio tras la expansión fiscal, esta sección cuantifica cuánto varía realmente el producto por cada unidad de gasto público adicional. La teoría IS-LM predice, combinando las Ecs. (3.6) y (3.12), un multiplicador fiscal cerrado que incorpora la atenuación del mercado monetario:

$$\frac{\Delta Y}{\Delta G} = \frac{1}{(1-c) + bk/h} = \frac{1}{0,2 + 20 \cdot 0,2/10} = \frac{1}{0,6} = 1,67 \quad (5.20)$$

$$\Delta Y = Y_1 - Y_0 = 3000 - 2833,33 = 166,67 \quad (5.21)$$

$$\Delta G = 300 - 200 = 100 \quad (5.22)$$

A partir de los dos equilibrios calculados directamente, el multiplicador fiscal observado confirma este valor teórico:

$$\text{Multiplicador observado} = \Delta Y / \Delta G = 166,67 / 100 = 1,67 \quad (5.23)$$

donde $\Delta Y = 166,67$ es el incremento del producto y $\Delta G = 100$ el aumento del gasto público. El valor 1,67 es inferior al multiplicador keynesiano simple $1/(1-c) = 5$ porque el mercado de dinero atenúa el efecto vía tipo de interés. El multiplicador obtenido tiene como contrapartida una reducción de la inversión privada (efecto *crowding-out*) que se cuantifica a continuación.

5.7.3. Cuantificación del efecto crowding-out

El tipo de interés aumentó de 6,67 % a 10,00 % (+3,33 puntos porcentuales), lo que redujo la inversión privada:

$$I_0 = 300 - 20 \cdot (6,67) = 166,67 \quad (5.24)$$

$$I_1 = 300 - 20 \cdot (10,00) = 100 \quad (5.25)$$

$$\Delta I = 100 - 166,67 = -66,67 \quad (5.26)$$

donde ΔI es la reducción de la inversión privada causada por el efecto *crowding-out*: el aumento del tipo de interés de 6,67 % a 10,00 % encarece el crédito y reduce la inversión en 66,67 unidades.

Tabla 5.3: Resumen expansión fiscal ($G: 200 \rightarrow 300$).

	Antes ($G = 200$)	Después ($G = 300$)
Y^*	2833,33	3000,00
r^*	6,67 %	10,00 %
Multiplicador fiscal	$\Delta Y / \Delta G = 1,67$ (exacto)	
Efecto crowding-out	$\Delta I = -66,67$ unidades	

Nota: sin efecto monetario el multiplicador keynesiano simple sería $1/(1 - c) = 5$; la interacción con el mercado de dinero (crowding-out) lo reduce a 1,67.

La Fig. 5.6 muestra los resultados de la política fiscal en OpenModelica.

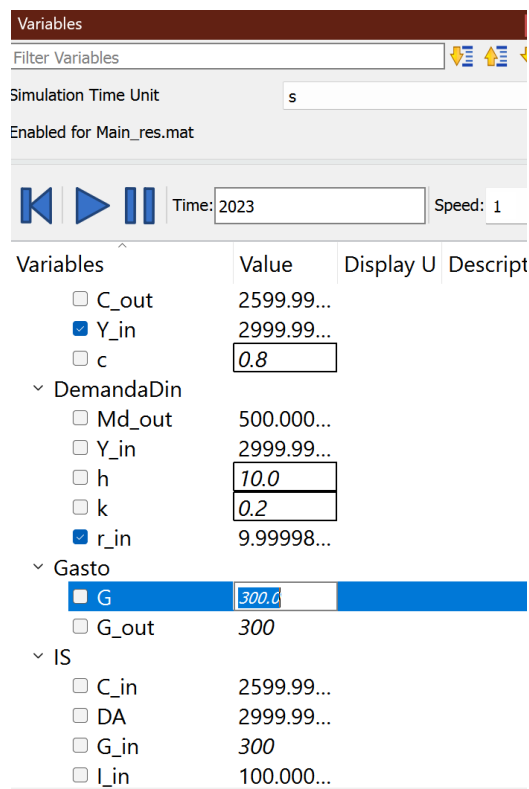


Figura 5.6: OpenModelica: política fiscal $G = 300$.

Analizados los efectos de la expansión fiscal, la segunda parte de esta sección estudia el impacto de un incremento en la oferta monetaria.

5.7.4. Variación de la política monetaria: expansión monetaria ($M_s: 500 \rightarrow 1200$)

Se simula un aumento de la oferta monetaria de $M_s = 500$ a $M_s = 1200$ unidades (Miles millones €/año), manteniendo constantes los demás parámetros. El análisis prevé dos efectos encadenados:

Efecto directo: el aumento de M_s desplaza la curva LM hacia la derecha, lo que reduce el tipo de interés de equilibrio.

Efecto sobre la inversión y la renta: el menor tipo de interés r abarata el coste del crédito e incentiva la inversión privada I ; el aumento de I eleva la demanda agregada y, por tanto, el producto nacional Y , lo que a su vez incrementa el consumo C vía la función $C(Y)$. La cadena causal completa es: $M_s \rightarrow r \rightarrow I \rightarrow Y \rightarrow C$, ilustrando el mecanismo keynesiano de transmisión monetaria.

Curva IS (invariante). La curva IS no varía respecto al equilibrio base, manteniéndose $Y = 3500 - 100 \cdot r$.

Nueva curva LM con $M_s = 1200$.

$$\begin{aligned} 1200 &= 0,2 \cdot Y - 10 \cdot r \\ 10 \cdot r &= 0,2 \cdot Y - 1200 \end{aligned} \quad (5.27)$$

$$\Rightarrow r = 0,02 \cdot Y - 120 \quad \leftarrow \text{nueva curva LM} \quad (5.28)$$

donde el término independiente $-120 = -M_{s,\text{nuevo}}/h = -1200/10$ y el coeficiente $0,02 = k/h = 0,2/10$. La ordenada en el origen más negativa (-120 frente a -50 original) refleja el incremento de la oferta monetaria.

Resolución del sistema. Sustituyendo la nueva ecuación LM en la curva IS:

$$\begin{aligned} Y &= 3500 - 100 \cdot (0,02 \cdot Y - 120) \\ Y &= 3500 - 2 \cdot Y + 12000 \\ 3 \cdot Y &= 15500 \end{aligned} \quad (5.29)$$

$$\Rightarrow Y_2^* = 5166,67 \quad (5.30)$$

$$r_2^* = 0,02 \cdot (5166,67) - 120 = 103,33 - 120 = -16,67\% \quad (5.31)$$

Nota sobre el resultado. El tipo de interés de equilibrio obtenido ($r_2^* = -16,67\%$) es negativo porque la expansión monetaria simulada (un aumento del 140 % de M_s) resulta muy intensa para un modelo IS-LM lineal, que no incorpora la cota inferior cero del tipo de interés. El valor es coherente con la formulación matemática, pero debe interpretarse como una limitación del modelo lineal y no como una predicción realista; con expansiones monetarias moderadas el equilibrio se mantiene en valores positivos.

5.7.5. Efectos sobre las variables macroeconómicas

La siguiente tabla resume el impacto de la expansión monetaria ($M_s: 500 \rightarrow 1200$) sobre las principales variables del modelo IS-LM:

Tabla 5.4: Efectos de la expansión monetaria ($M_s: 500 \rightarrow 1200$) sobre el equilibrio IS-LM.

Variable	Antes ($M_s=500$)	Después ($M_s=1200$)	Variación
Renta (Y)	2833.33	5166.67	+2333.34 (+82.3 %)
Tipo de interés (r)	6.67 %	-16.67 %	-23.33 puntos
Consumo (C)	2466.67	4333.33	+1866.67 (+75.6 %)
Inversión (I)	166.67	633.33	+466.66 (+280.0 %)

La Figura 5.7 muestra los valores de equilibrio obtenidos en OpenModelica tras la expansión monetaria. El panel izquierdo reproduce los valores teóricos calculados analíticamente, que coinciden con los resultados de la simulación.

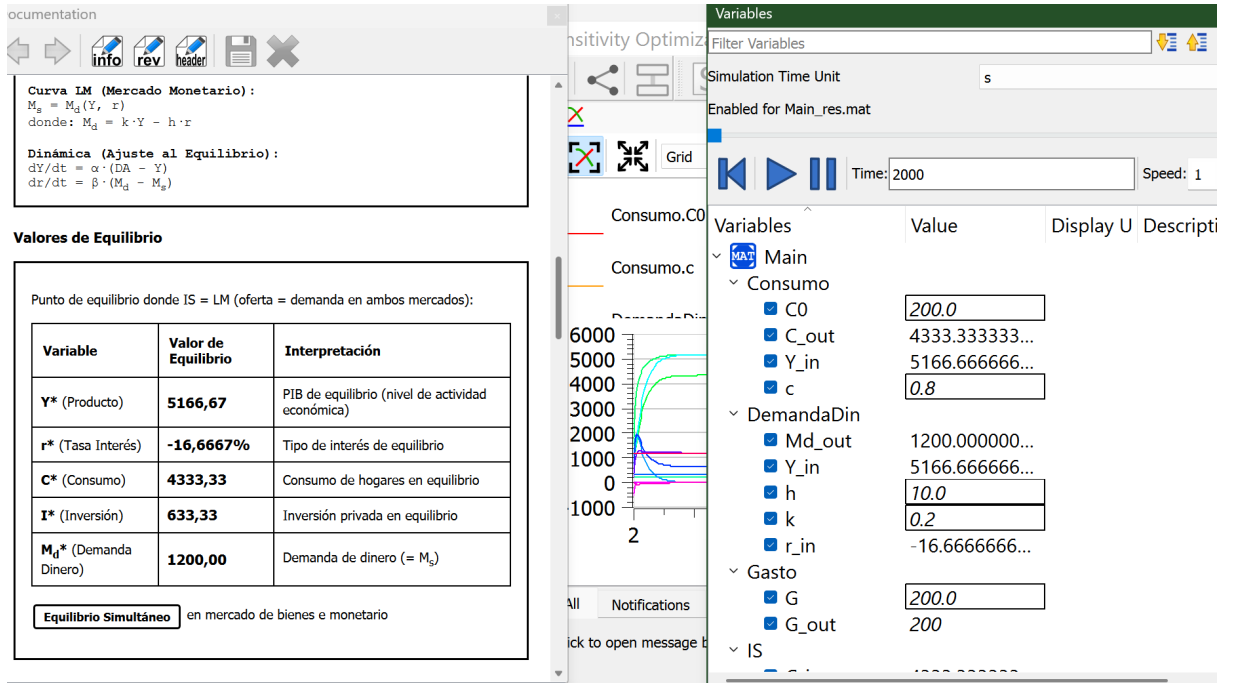


Figura 5.7: OpenModelica: valores con la política monetaria $M_s = 1200$.

5.7.6. Multiplicador monetario

De forma análoga al multiplicador fiscal de la Sec. 5.7.2, la teoría predice un multiplicador monetario cerrado:

$$\mu_m = \frac{\Delta Y}{\Delta M_s} = \frac{b}{h(1-c) + bk} = \frac{20}{10 \cdot 0,2 + 20 \cdot 0,2} = \frac{20}{6} = 3,33 \quad (5.32)$$

El multiplicador monetario observado, definido como la variación de renta por unidad de oferta monetaria inyectada, confirma este valor teórico:

$$\mu_m = \Delta Y / \Delta M_s = 2333,34 / 700 = 3,33 \quad (5.33)$$

donde μ_m es el **multiplicador monetario** (se usa μ_m para distinguirlo del parámetro k de sensibilidad de la demanda de dinero), $\Delta Y = 5166,67 - 2833,33 = 2333,34$ el incremento del producto (partiendo del equilibrio base de la Figura 5.5) y $\Delta M_s = 1200 - 500 = 700$ el aumento de la oferta monetaria. Un valor $\mu_m = 3,33$ indica que cada unidad adicional de oferta monetaria genera 3,33 unidades de producto adicional.

La Figura 5.8 muestra los parámetros del modelo IS-LM y el panel de variables tras la expansión monetaria en OpenModelica.

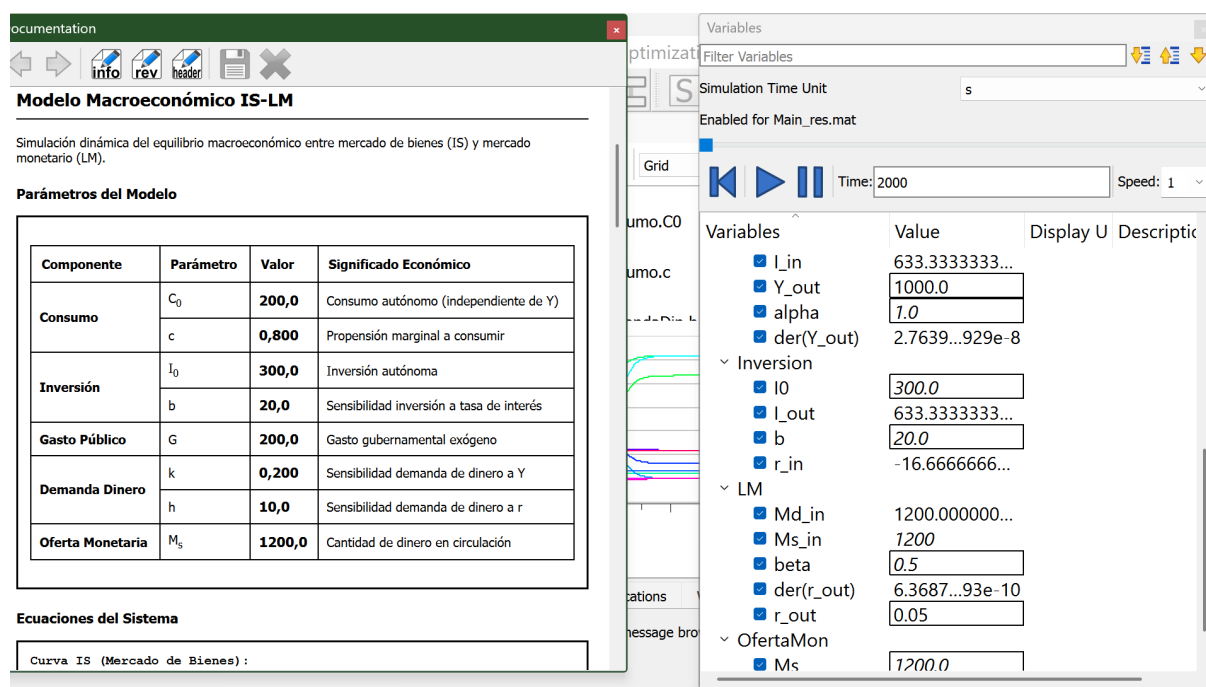


Figura 5.8: OpenModelica: parámetros del modelo IS-LM y panel de variables tras expansión monetaria.

5.7.7. Caso España: Gasto Público 2010–2023 (datos reales del Banco Mundial)

Se realizaron pruebas de integración con datos históricos del Banco Mundial para validar la capacidad del sistema de incorporar series temporales reales. Se descargó la serie WB_GASTO_PUB para el código ISO ES y se inyectaron los datos en el componente GastoPublico (Miles millones €/año), exportando el modelo a ISLM_España.mo y simulando en OpenModelica con StartTime=2010, StopTime=2023.

Tabla 5.5: Serie histórica de gasto público de España descargada del Banco Mundial.

Año	Gasto Público (Miles M €)	Observación
2010	293,4222	Expansión post-crisis
2011	306,0882	Máximo pre-austeridad
2012	264,6498	Inicio austeridad (pico crisis)
2013	269,8484	Austeridad
2014	269,8623	Austeridad
2015	233,4604	Mínimo austeridad
2016	235,6187	Estabilización
2017	245,0968	Recuperación
2018	266,0631	Recuperación
2019	262,9959	Pre-COVID
2020	281,3754	Shock fiscal COVID-19
2021	306,9365	Expansión post-COVID
2022	288,7807	Normalización
2023	315,3271	Nuevo máximo

La simulación en OpenModelica reproduce correctamente la serie temporal de Gasto Público G . La variable exhibe la evolución esperada: expansión 2010–2011, contracción 2012–2015 (período de austeridad), estabilización 2015–2019, expansión 2020 (choque fiscal COVID-19) y normalización 2021–2023. El modelo captura la dinámica cualitativa (direcciones de cambio interanuales) de forma esencial.

La interpretación económica de estos datos en el contexto IS-LM es que cada punto temporal tiene su propio equilibrio: cada año, dependiendo del Gasto Público real, el modelo obtiene una Renta de equilibrio $Y^*(t)$ y un tipo de interés de equilibrio $r^*(t)$ distintos, capturando así la dinámica real de las políticas económicas.

Una limitación a señalar: los servidores del Banco Mundial no siempre devuelven datos completos. Para algunos países con series temporales cortas (principalmente economías emergentes), puede faltar información correspondiente de algunos años o indicadores.

La Figura 5.9 muestra la simulación con datos reales de España.

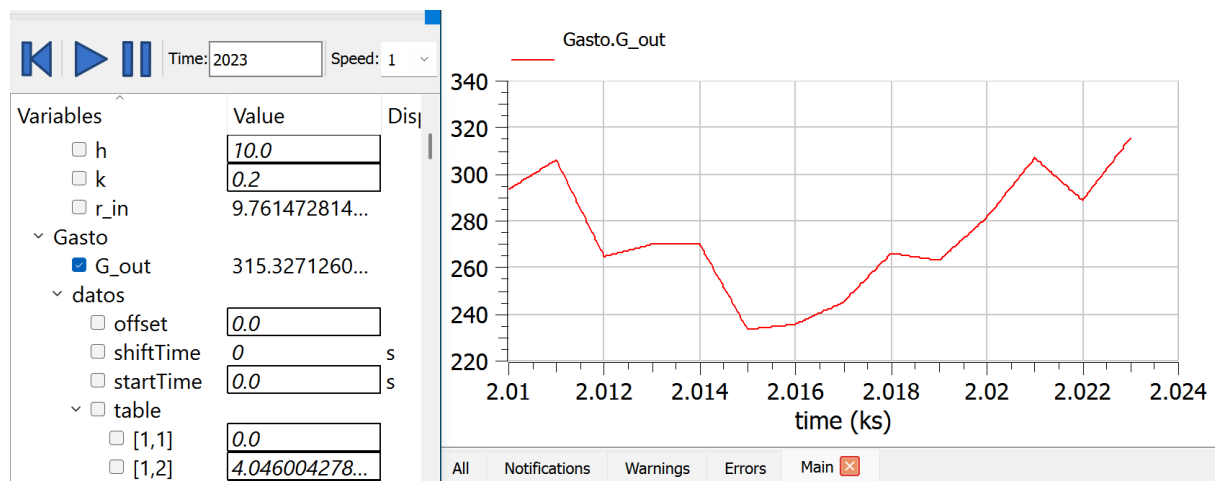


Figura 5.9: OpenModelica: gasto público de España 2010–2023.

5.7.8. Simulación IS-LM con datos reales de EE.UU. (1970–2022)

El editor descarga automáticamente series históricas del Banco Mundial mediante la API REST api.worldbank.org/v2. Para la simulación IS-LM de EE.UU. se utilizaron dos indicadores:

- **Gasto público G :** indicador NE.CON.GOV.T.CD (*General government final consumption expenditure, current USD*), serie 1970–2022, en miles de millones de dólares corrientes.
- **Oferta monetaria M_s :** indicador FM.LBL.BMNY.GD.ZS (*Broad Money M3, % of GDP*), serie 1960–2024. El programa escala el valor al rango del modelo.

El modelo IS-LM se simuló sobre el rango común 1970–2022 con los parámetros estructurales calibrados para la economía estadounidense.

Las Figuras 5.10 y 5.11 muestran la evolución del producto nacional Y y del tipo de interés r a lo largo del periodo simulado. El producto nacional crece de aproximadamente 1.000 en 1970 hasta 12.000 miles de millones de dólares en 2022, siguiendo la trayectoria histórica de la economía estadounidense. El tipo de interés r evoluciona desde valores negativos iniciales hasta estabilizarse en torno a 85–90 unidades al final del periodo. Estas unidades son las del modelo IS-LM lineal calibrado (no puntos porcentuales reales), ya que los indicadores del Banco Mundial se escalan internamente al rango del modelo; el comportamiento cualitativo (tendencia creciente del tipo de interés con el crecimiento de la economía) es coherente con la teoría.

Los resultados confirman que el exportador genera código Modelica que OpenModelica

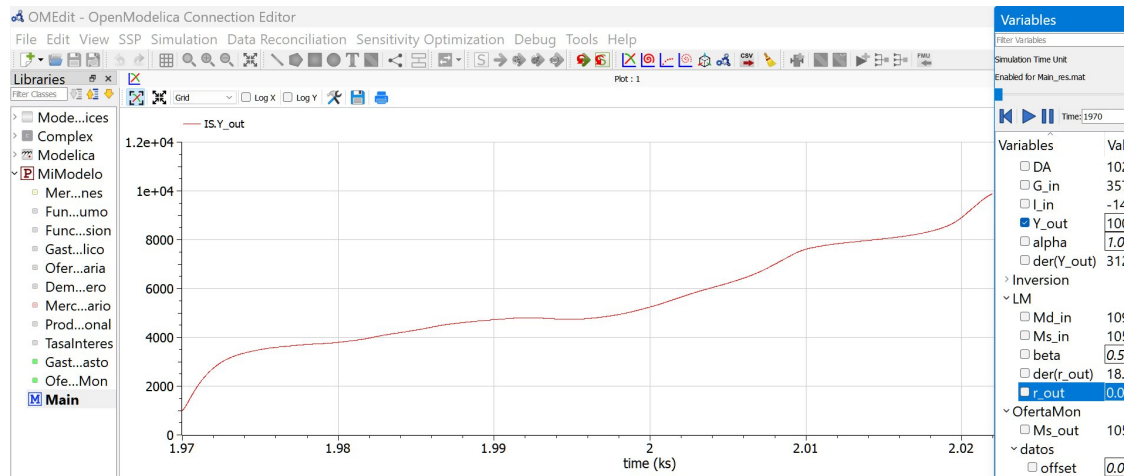


Figura 5.10: Simulación IS-LM con datos reales EE.UU. (1970–2022): producto nacional Y (miles de millones USD, Banco Mundial).

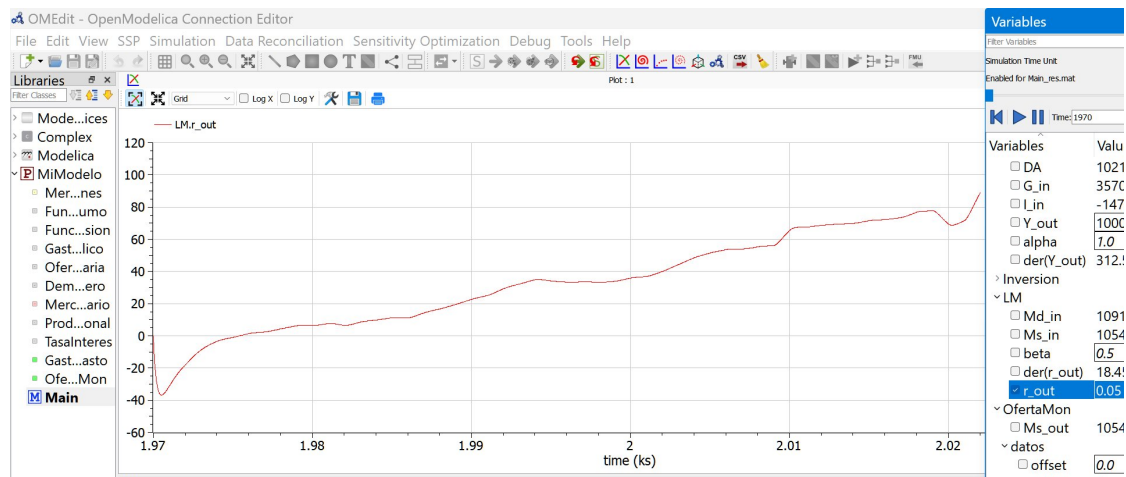


Figura 5.11: Simulación IS-LM con datos reales EE.UU. (1970–2022): tipo de interés r (Banco Mundial).

compila y simula correctamente con datos históricos reales, produciendo trayectorias económicamente coherentes con la evolución documentada de la economía estadounidense en el periodo analizado.

Con esto queda completo el bloque de validación del modelo IS-LM: compilación, validación semántica, equilibrio analítico, respuesta a políticas fiscal y monetaria, y ajuste a datos históricos. El siguiente bloque repite el proceso para el modelo de Solow, cuya validación es especialmente exigente por tratarse de un sistema dinámico de largo plazo.

5.8 Verificación matemática del modelo de Solow. Validación analítica del estado estacionario

La verificación matemática del modelo de Solow compara el estado estacionario calculado analíticamente con el alcanzado por la simulación dinámica. Esta validación es crítica porque el Solow es un modelo dinámico donde la convergencia depende de la correcta implementación de la ecuación diferencial de acumulación de capital. Esta validación se realiza con valores de parámetros estándar, no con datos específicos de un país concreto; más adelante (Sec. 5.9) se realiza la validación con datos reales del Banco Mundial. La sección se estructura en cuatro apartados: parámetros de validación (Sec. 5.8.1), cálculo analítico del estado estacionario (Sec. 5.8.2), simulación a 200 años con convergencia casi completa (Sec. 5.8.3) y simulación a 100 años como contraste de convergencia parcial (Sec. 5.8.4).

5.8.1. Parámetros de validación

La validación emplea los parámetros estándar de la literatura neoclásica (definidos en la Tabla 3.3 del Cap. 3); la dinámica del modelo está gobernada por la ecuación de acumulación de capital per cápita Ec. (3.18). Sus valores concretos para esta validación son:

Con los parámetros fijados, se calcula analíticamente el estado estacionario que la simulación debe alcanzar.

Tabla 5.6: Parámetros estándar del modelo de Solow para la validación.

Parámetro	Símbolo	Valor
Tasa de ahorro	s	0.25 (25 %)
Tasa de depreciación	δ	0.05 (5 %)
Tasa de crecimiento poblacional	n	0.02 (2 %)
Elasticidad del capital (Cobb-Douglas)	α	0.33

5.8.2. Cálculo analítico del estado estacionario

Aplicando la fórmula del estado estacionario del capital por trabajador (Ec. (3.23)):

$$k^* = \left[\frac{s}{n + \delta} \right]^{\frac{1}{1-\alpha}} \quad (5.34)$$

$$\Rightarrow k^* = 3,5714^{1,4925} = 6,6855 \quad (5.35)$$

La producción per cápita de estado estacionario es:

$$y^* = (k^*)^\alpha = 6,6855^{0,33} = 1,872 \quad (5.36)$$

donde $y^* = 1,872$ es el **producto por trabajador en el estado estacionario** con los parámetros base ($s = 0,25$, $\delta = 0,05$, $n = 0,02$, $\alpha = 0,33$). La siguiente subsección simula la convergencia dinámica y compara el resultado numérico con este valor teórico.

5.8.3. Simulación de convergencia a 200 años (parámetros fijos)

Esta simulación utiliza **parámetros calibrados fijos** ($s = 0,25$, $n = 0,02$, $\delta = 0,05$, $\alpha = 0,33$), no datos históricos reales. El objetivo es verificar que el modelo converge al estado estacionario teórico calculado analíticamente en la sección anterior.

A 200 años la convergencia es prácticamente completa (Tabla 5.7):

Tabla 5.7: Comparativa estado estacionario Solow: simulación 200 años.

Variable	Valor analítico	OpenModelica (200 años)	Error relativo
k^* (capital por trabajador)	6.6855	6.6847	<0.02 %
y^* (producción por trabajador)	1.872	1.8718	<0.02 %

La Figura 5.12 muestra la simulación Solow a 200 años.

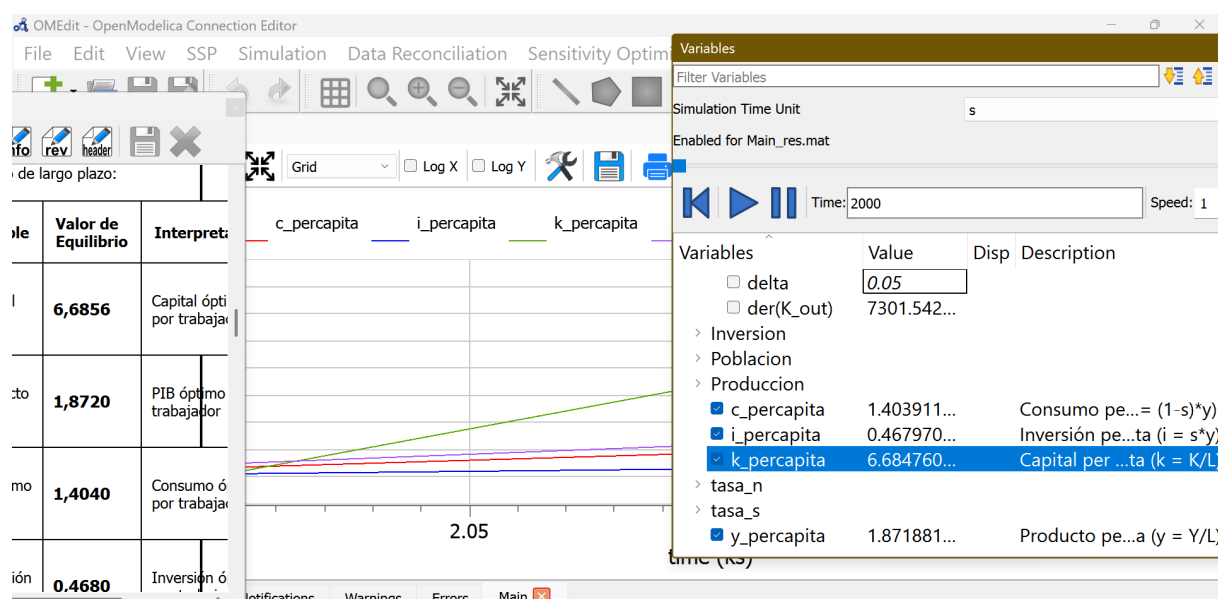


Figura 5.12: OpenModelica: simulación Solow 200 años.

Comprobada la convergencia a 200 años, la siguiente simulación repite el experimento a la mitad de plazo para verificar que el resultado no depende del horizonte temporal elegido.

5.8.4. Simulación de convergencia a 100 años (parámetros fijos, 2000–2100)

Esta simulación usa los mismos **parámetros calibrados fijos** que la anterior, proyectando la evolución del modelo desde 2000 hasta 2100. A 100 años la convergencia no es completa, lo que se refleja en valores ligeramente inferiores al teórico (Tabla 5.8):

Tabla 5.8: Comparativa estado estacionario Solow: simulación 100 años.

Variable	Valor analítico	OpenModelica (100 años)	Observación
k^* (capital por trabajador)	6.6855	6.5996	Convergencia parcial a 100 años
y^* (producción por trabajador)	1.872	1.8640	Convergencia parcial a 100 años

El estado estacionario se visualiza en la Figura 5.13.

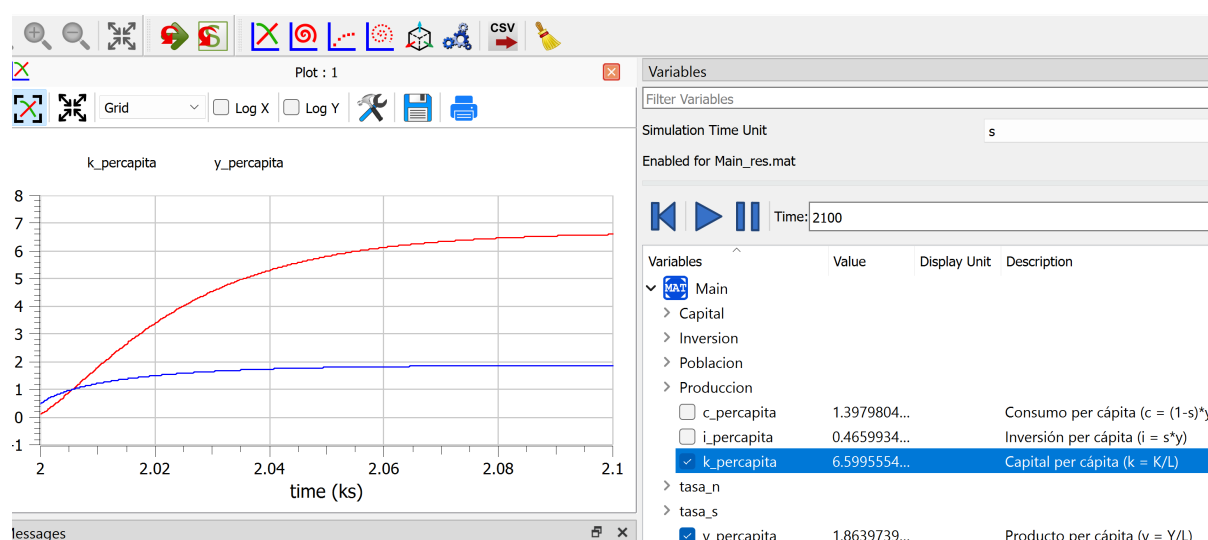


Figura 5.13: OpenModelica: estado estacionario Solow.

La pequeña discrepancia entre la simulación a 100 años y el valor analítico es atribuible a que OpenModelica aplica las tasas s y n de forma continua, mientras que la fórmula analítica asume convergencia completa. La concordancia excelente a 200 años confirma que las funciones y las fórmulas de exportación están correctamente construidas. Verificado el modelo con parámetros fijos, la siguiente sección lo calibra con series históricas reales del Banco Mundial.

5.9 Solow EE.UU. con datos históricos reales del Banco Mundial (1970–2024)

La validación con datos reales calibra el modelo de Solow con información económica de Estados Unidos obtenida del Banco Mundial, verificando que la evolución simulada reproduce patrones observados en series históricas. Esta prueba transforma el modelo de una herramienta puramente teórica a un instrumento de análisis aplicado. La sección se organiza en seis apartados: simulación con indicadores reales (Sec. 5.9.1), el concepto de estado estacionario móvil (Sec. 5.9.2), los parámetros de simulación (Sec. 5.9.3), el estado estacionario teórico de referencia (Sec. 5.9.4), la comparativa con datos reales a 54 años (Sec. 5.9.5) y la proyección a 130 años (Sec. 5.9.6).

5.9.1. Simulación Solow con datos reales de EE.UU. (1970–2024)

El modelo de Solow se parametrizó con series históricas reales de EE.UU. descargadas del Banco Mundial mediante los indicadores:

- **Tasa de ahorro s :** indicador NE.GDI.TOTL.ZS (*Gross savings, % of GDP*), serie 1970–2024.
- **Tasa de crecimiento poblacional n :** indicador SP.POP.GROW (*Population growth, annual %*), serie 1961–2024.

Los valores en $t = 1970$ son los del panel de Variables: $k_{\text{per cápita}} = 5,9509$, $y_{\text{per cápita}} = 1,8014$, $i_{\text{per cápita}} = 0,3934$, $c_{\text{per cápita}} = 1,4080$.

Las tres gráficas siguientes muestran la evolución de las variables per cápita durante el periodo simulado, confirmando el comportamiento teórico del modelo de Solow con parámetros reales.

La Figura 5.14 representa la acumulación de capital por trabajador en la simulación de Solow para EE.UU.

La Figura 5.15 muestra la evolución del producto por trabajador en la misma simulación.

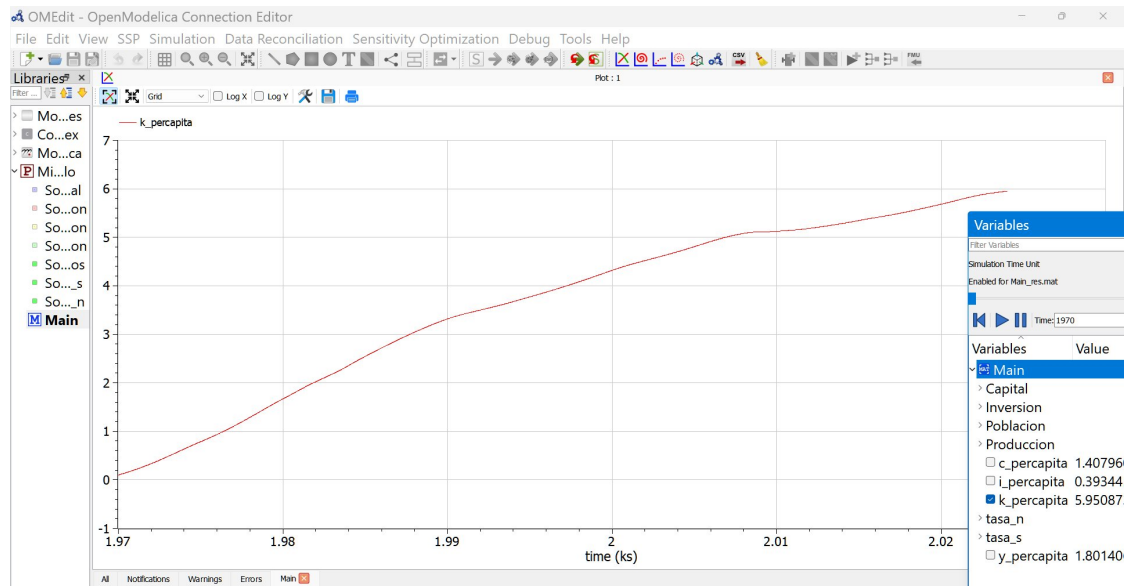


Figura 5.14: Solow EE.UU. (1970–2024): capital por trabajador k , con acumulación progresiva hacia el estado estacionario (Banco Mundial).

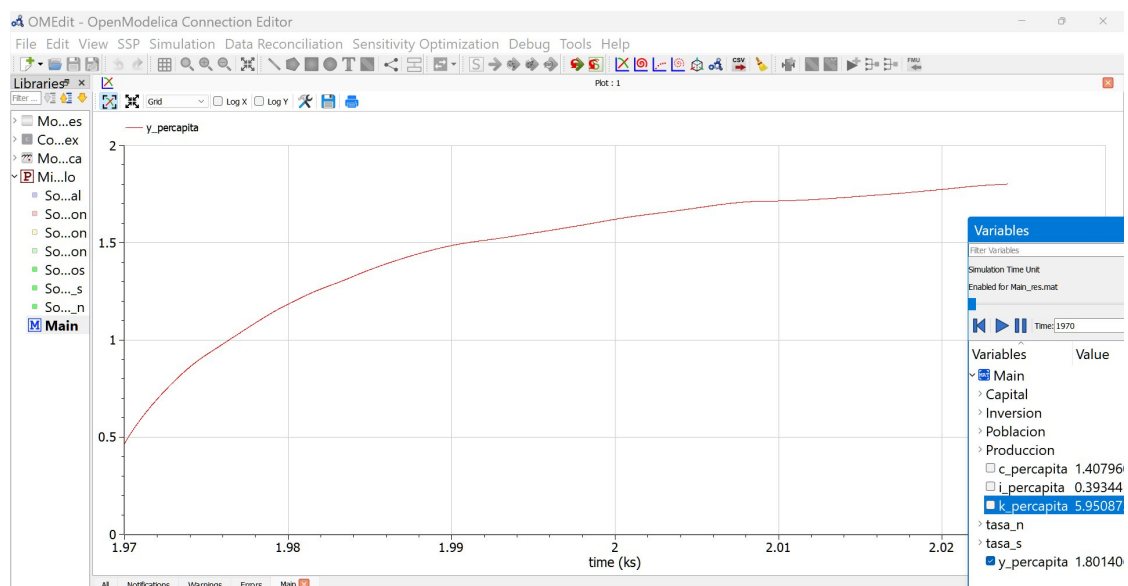


Figura 5.15: Solow EE.UU. (1970–2024): producción por trabajador y , con convergencia al estado estacionario con curva cóncava característica (Banco Mundial).

La Figura 5.16 muestra la evolución de la inversión por trabajador i en la misma simulación.

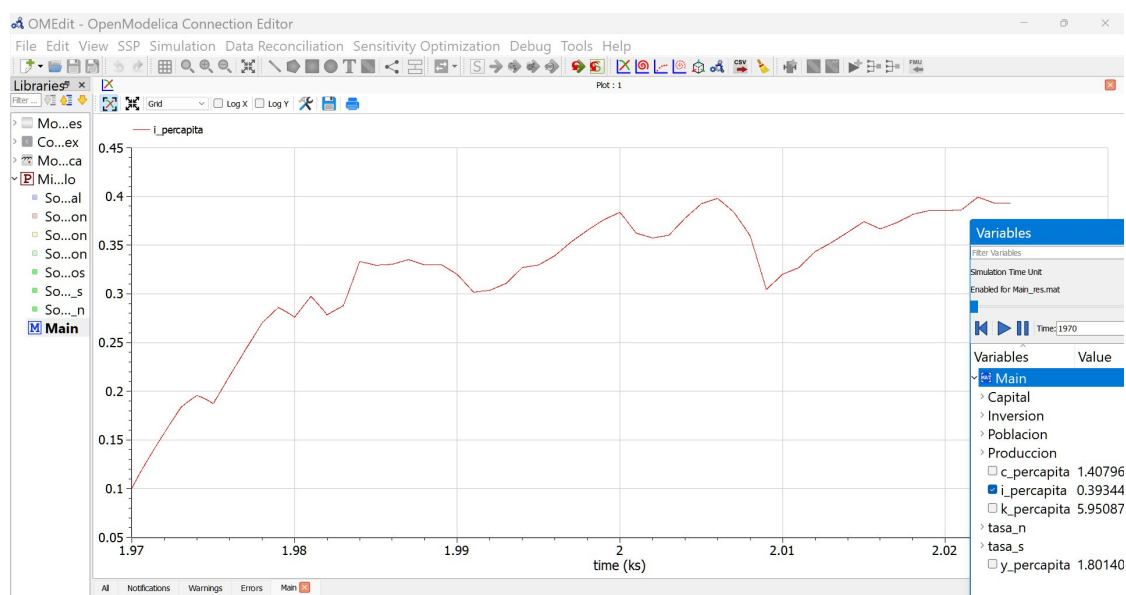


Figura 5.16: Solow EE.UU. (1970–2024): inversión por trabajador i ; las oscilaciones reflejan los ciclos económicos reales captados por la tasa de ahorro histórica (Banco Mundial).

Las oscilaciones visibles en la inversión per cápita (Figura 5.16) no son ruido numérico sino el reflejo directo de los **ciclos económicos reales** de EE.UU. captados por la serie histórica de la tasa de ahorro: las caídas corresponden a las recesiones de 1973–1975, 1981–1982, 2000–2001, 2008–2009 y 2020 (COVID-19). Esto demuestra que el modelo responde correctamente a las variaciones en los parámetros de entrada. Para comprender la dinámica de estas simulaciones con datos históricos variables, el siguiente apartado introduce el concepto de estado estacionario móvil.

5.9.2. Concepto clave: estado estacionario móvil

Cuando las tasas $s(t)$ y $n(t)$ varían con el tiempo (datos reales anuales), el estado estacionario deja de ser fijo y se convierte en un objetivo móvil (Tabla 5.9):

Tabla 5.9: Evolución del estado estacionario objetivo en EE.UU. con datos reales.

Año	$s(t)$	$n(t)$	$k^*(t)$ calculado	Interpretación
2000	22.0 %	1.2 %	6.62	Objetivo de largo plazo en 2000
2010	19.8 %	0.3 %	7.15	Nuevo objetivo (mayor k^* por menor n)
2020	25.1 %	-0.1 %	11.45	Nuevo objetivo (mayor ahorro, pop. estancada)

5.9.3. Parámetros de simulación: EE.UU. 1970-2024 (54 años)

La Tabla 5.10 recoge los parámetros empleados en la simulación de Solow con datos reales de EE.UU. (1970–2024).

Tabla 5.10: Parámetros de la simulación Solow EE.UU. 1970–2024.

Parámetro	Valor
Año de inicio	1970 (primer año con datos completos en TimeTable)
Población inicial (L_0)	205.000 (205 millones de habitantes en 1970)
Capital inicial (K_0)	600.000 (estimado)
Tasa de ahorro media histórica (s)	0.2184 (21.84 %)
Tasa de crec. poblacional media (n)	0.009764 (0.9764 %)
Tasa de depreciación (δ)	0.05 (5 %)
Elasticidad del capital (α)	0.33

Con los parámetros medios identificados, se calcula el estado estacionario teórico que servirá de referencia para comparar con la simulación.

5.9.4. Estado estacionario teórico con parámetros históricos medios

Usando las medias históricas de la tabla anterior como parámetros de referencia, el estado estacionario teórico de largo plazo es:

$$k^* = \left(\frac{0,2184}{0,009764 + 0,05} \right)^{\frac{1}{0,67}} = \left(\frac{0,2184}{0,059764} \right)^{1,4925} = 3,6544^{1,4925} = 6,9186 \quad (5.37)$$

$$y^* = (k^*)^{0,33} = 6,9186^{0,33} = 1,8932 \quad (5.38)$$

$$i^* = s \cdot y^* = 0,2184 \times 1,8932 = 0,4135 \quad (5.39)$$

$$c^* = y^* - i^* = 1,8932 - 0,4135 = 1,4798 \quad (5.40)$$

donde c^* es el **consumo por trabajador en el estado estacionario**, equivalente a la fracción $(1 - s) = 0,7816$ del producto per cápita $y^* = 1,8932$, e $i^* = 0,4135$ la inversión por trabajador. Calculado este estado estacionario teórico, la siguiente sección lo contrasta con los resultados obtenidos al simular el modelo con la serie histórica completa en OpenModelica.

5.9.5. Comparativa de resultados con datos reales: simulación 54 años (1970–2024)

La Figura 5.17 muestra los resultados para EE.UU. en t=2024.

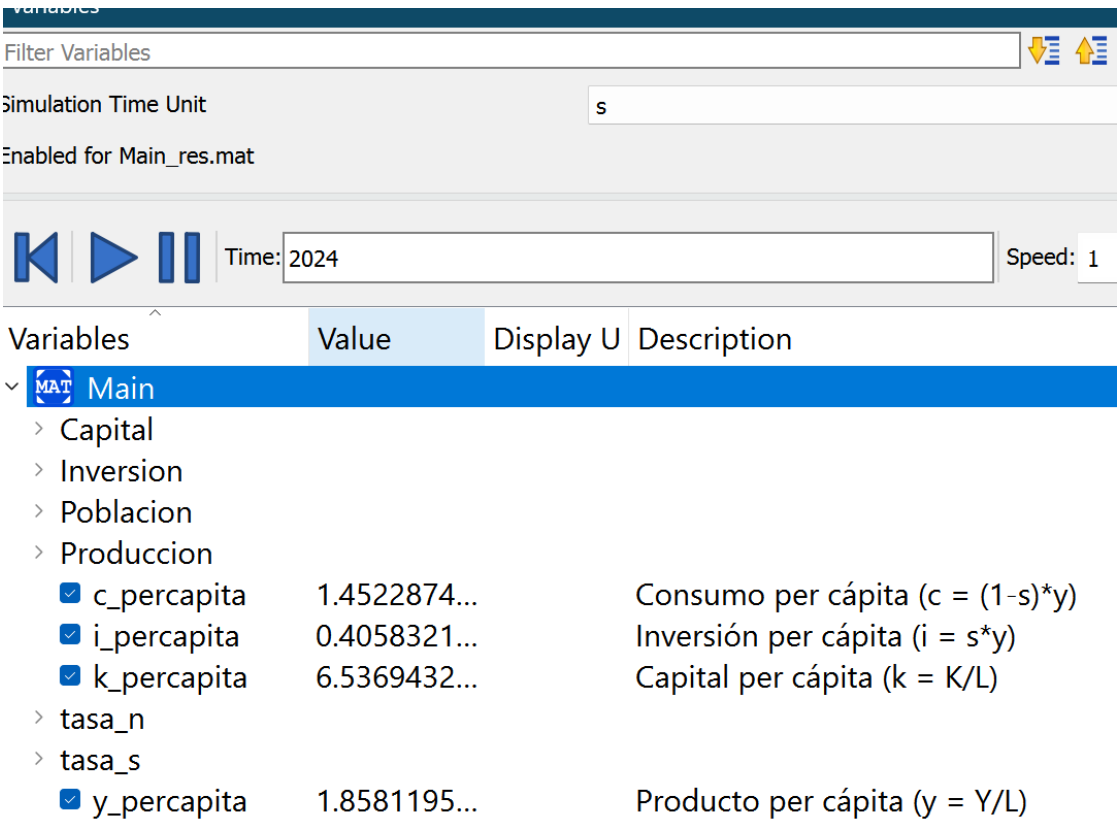


Figura 5.17: OpenModelica: Solow EE.UU. t=2024.

La Tabla 5.11 compara los resultados de la simulación de Solow para EE.UU. a lo largo de los 54 años analizados.

Tabla 5.11: Comparativa resultados Solow EE.UU. 54 años (1970–2024).

Variable	OpenModelica (1970-2024. Da- tos anuales)	Valor analíti- co, calculado (medias fijas)	Diferencia
k (capital por trabajador)	6.5369	6.9186	-0.38 (-5.5 %)
y (producción por trabajador)	1.8581	1.8932	-0.035 (-1.9 %)
i* (inversión por trabajador)	0.4058	0.4135	-0.0077 (-1.9 %)
c* (consumo por trabajador)	1.4522	1.4798	-0.0276 (-1.9 %)

Para confirmar la convergencia a largo plazo, se extiende la simulación a 130 años empleando los mismos datos históricos reales.

5.9.6. Proyección con datos reales: simulación 130 años (1970–2100)

La Tabla 5.12 resume la proyección del modelo de Solow con datos reales extendida a 130 años (1970–2100).

Tabla 5.12: Tabla comparativa Solow EE.UU. 130 años.

Variable	Valor analíti- co	OpenModelica (1970–2100)	Diferencia
k* (capital por trabajador)	6.9186	6.9007	-0.018 (-0.26 %)
y* (producción por trabajador)	1.8932	1.8918	-0.001 (-0.09 %)
i* (inversión por trabajador)	0.4135	0.4131	-0.0004 (-0.10 %)
c* (consumo por trabajador)	1.4798	1.4787	-0.0011 (-0.07 %)

La convergencia es casi completa a 130 años (error < 0,26 %), lo que confirma que el modelo responde correctamente. La discrepancia a 54 años (5,5 %) es inherente al uso de datos anuales variables (`TimeTable`) frente a la fórmula analítica con medias fijas.

La proyección a $t = 2100$ se muestra en la Figura 5.18.

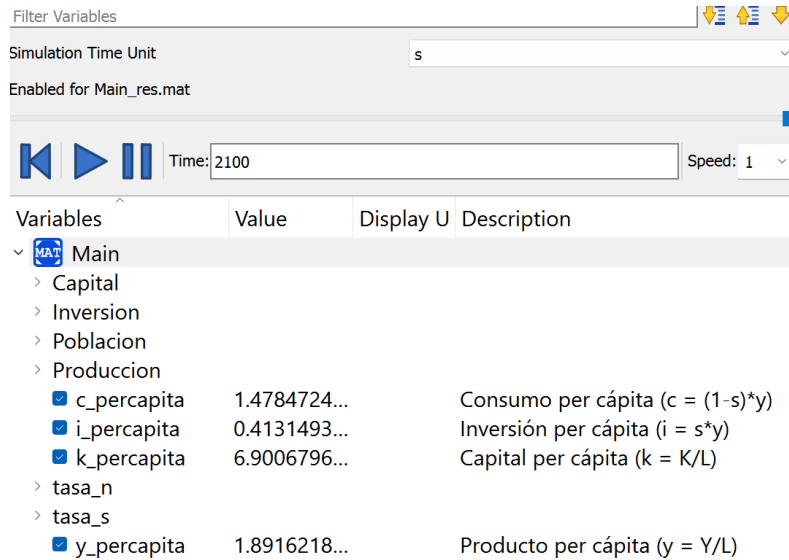


Figura 5.18: OpenModelica: Solow EE.UU. $t=2100$.

5.10 Validación del modelo Solow con Economía Circular

Esta sección valida la extensión de Economía Circular del modelo Solow, verificando que el componente `Solow_Circularidad` genera código Modelica correcto y que las simulaciones en OpenModelica reproducen los valores analíticos para distintos valores de la tasa de circularidad $\rho \in [0, 1]$. La sección se organiza en cinco apartados: definición de escenarios de circularidad (Sec. 5.10.1), cálculo analítico del estado estacionario para cada escenario (Sec. 5.10.2), comparativa completa de resultados (Sec. 5.10.3), validación mediante simulación en OpenModelica (Sec. 5.10.4) e interpretación económica (Sec. 5.10.5).

5.10.1. Parámetros de validación

Las variables base (s, n, δ, α) están definidas en la Tabla 3.3 del Cap. 3. La tasa de circularidad $\rho \in [0, 1]$ se introduce en la Ec. (3.28) como reducción de la depreciación efectiva: $\delta_{ef} = \delta(1 - \rho)$; el estado estacionario circular resultante se expresa mediante la Ec. (3.30). La validación usa los parámetros base ($s = 0,25, n = 0,02, \alpha = 0,33, \delta = 0,05$) y añade tres escenarios de circularidad (Tabla 5.13):

Definidos los tres escenarios, se aplica la fórmula del estado estacionario para cuantificar el efecto de cada nivel de circularidad.

Tabla 5.13: Escenarios de circularidad para la validación del modelo Solow extendido.

Escenario	ρ	δ_{ef}	Modelo	Descripción
Base	$\rho = 0,0$	$\delta_{\text{ef}} = 0,05$	Solow estándar	Sin circularidad (lineal)
Parcial	$\rho = 0,3$	$\delta_{\text{ef}} = 0,035$	Solow circular	30 % del capital se reutiliza
Alto	$\rho = 0,6$	$\delta_{\text{ef}} = 0,02$	Solow circular	60 % del capital se reutiliza

5.10.2. Cálculo analítico del estado estacionario

Aplicando la ecuación (3.30) para cada escenario:

$$\begin{aligned}
 \rho = 0,0 : \quad k^* &= \left[\frac{0,25}{0,05 + 0,02} \right]^{\frac{1}{0,67}} = 3,5714^{1,4925} = 6,6856 \\
 \rho = 0,3 : \quad k^* &= \left[\frac{0,25}{0,035 + 0,02} \right]^{\frac{1}{0,67}} = 4,5455^{1,4925} = 9,5821 \\
 \rho = 0,6 : \quad k^* &= \left[\frac{0,25}{0,02 + 0,02} \right]^{\frac{1}{0,67}} = 6,25^{1,4925} = 15,4128
 \end{aligned} \tag{5.41}$$

Calculado el estado estacionario analítico para cada escenario de circularidad, la siguiente sección compara estos valores con los obtenidos al simular cada modelo en OpenModelica.

5.10.3. Comparativa de resultados

La Tabla 5.14 amplía los tres escenarios anteriores con un barrido completo de ρ en incrementos de 0,1, permitiendo apreciar la progresión no lineal del estado estacionario para todo el rango de circularidad.

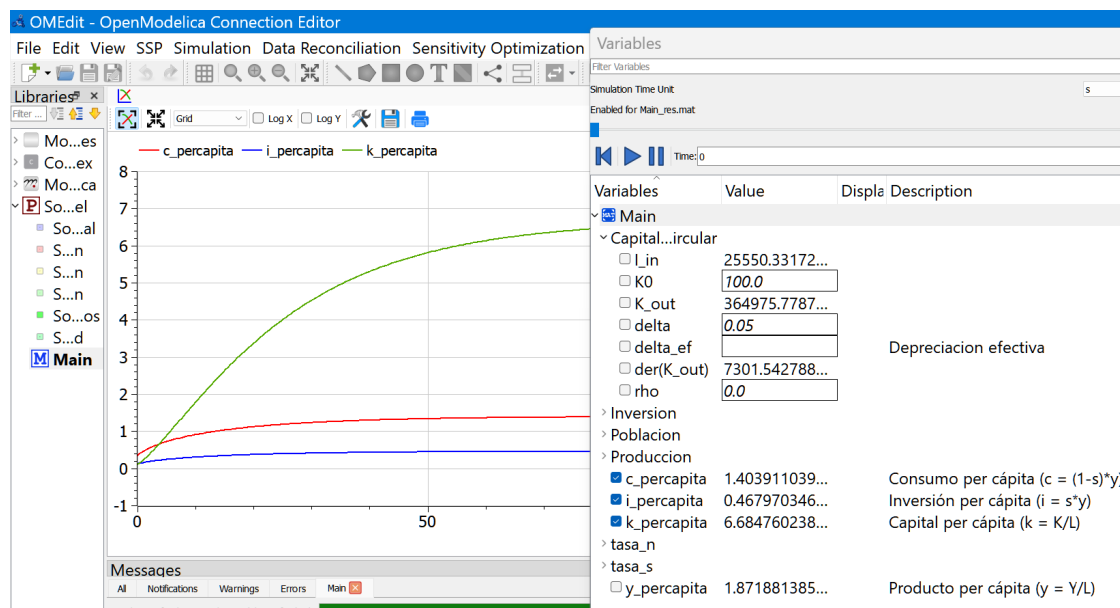
A continuación comparamos los resultados numéricos de la tabla 5.14 con los valores del modelo Solow_Circularidad generado con el editor gráfico y que ha sido exportado a OpenModelica.

Tabla 5.14: Estado estacionario del modelo Solow con Economía Circular: resultados analíticos.

ρ	k^*	y^*	i^*	c^*	Δk^* vs base
0,0 (base)	6,6856	1,8720	0,4680	1,4040	—
0,1	7,4675	1,9415	0,4854	1,4562	+11,7 %
0,2	8,4151	2,0196	0,5049	1,5147	+25,9 %
0,3	9,5821	2,1081	0,5270	1,5810	+43,3 %
0,4	11,0469	2,2094	0,5523	1,6570	+65,2 %
0,5	12,9281	2,3271	0,5818	1,7453	+93,4 %
0,6	15,4128	2,4660	0,6165	1,8495	+130,5 %

5.10.4. Validación del modelo Solow circular mediante simulación en OpenModelica

Para verificar empíricamente los resultados analíticos anteriores, se exportó el modelo `Solow_Circularidad` a OpenModelica y se simularon los tres escenarios representativos ($\rho = 0,0$, $\rho = 0,3$ y $\rho = 0,6$) con un horizonte de 200 años (`StartTime=0`, `StopTime=200`), manteniendo fijos los parámetros base ($s = 0,25$, $n = 0,02$, $\alpha = 0,33$, $\delta = 0,05$) y partiendo de $k(0) = 1$ ($K_0 = L_0 = 100$). A continuación, en la figura 5.19, figura 5.20 y figura 5.21 se observan los resultados correspondientes a la simulación de estos tres escenarios correspondientes.

**Figura 5.19:** Simulación OpenModelica: Solow con Economía Circular, $\rho = 0,0$ (escenario base). Convergencia de $k_{percapita}$, $c_{percapita}$ e $i_{percapita}$ hacia el estado estacionario.

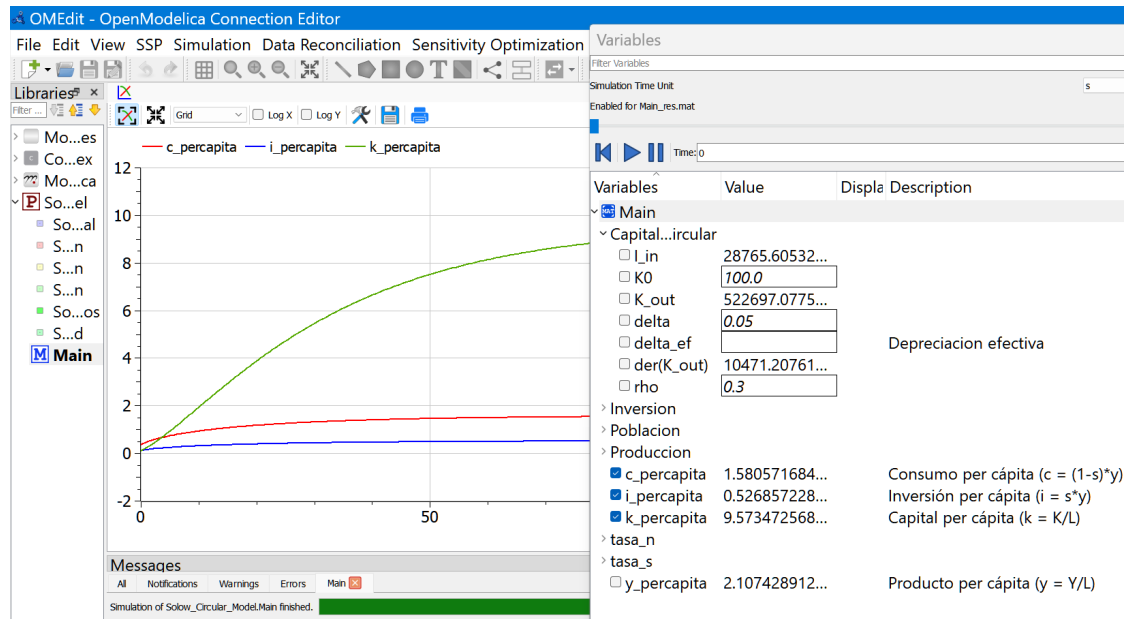


Figura 5.20: Simulación OpenModelica: Solow con Economía Circular, $\rho = 0,3$ (circularidad parcial). El estado estacionario se desplaza visiblemente hacia un nivel de capital per cápita más alto.

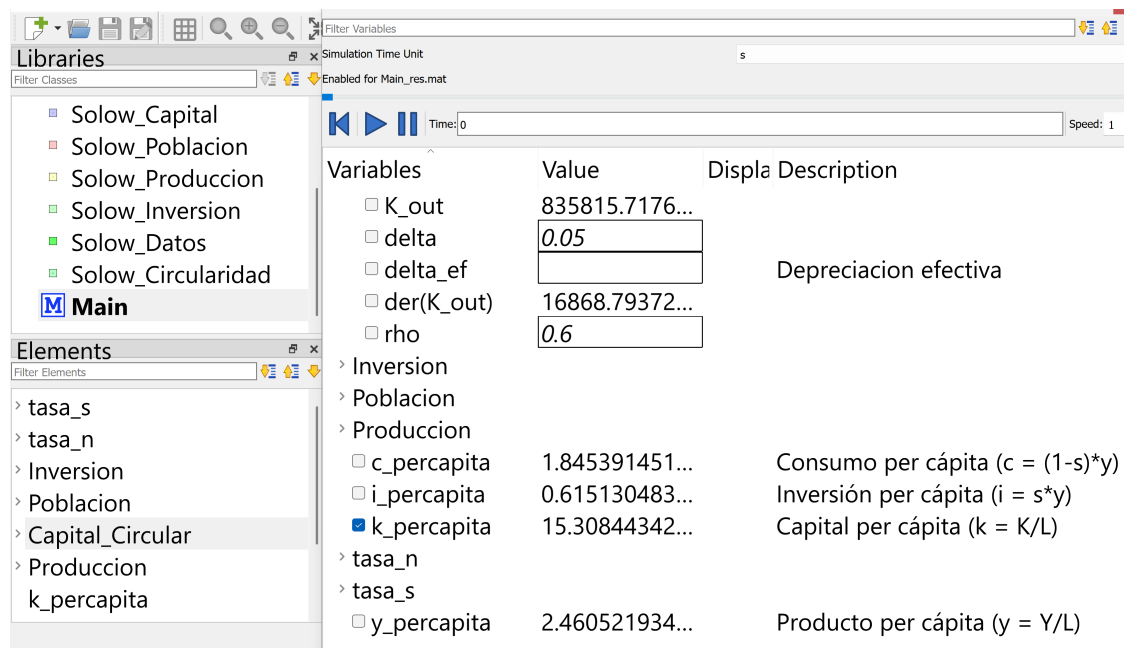


Figura 5.21: Simulación OpenModelica: Solow con Economía Circular, $\rho = 0,6$ (circularidad alta). El nuevo estado estacionario es sustancialmente mayor y la convergencia es algo más lenta, al estar el objetivo más alejado del punto de partida.

La Tabla 5.15 compara el valor de k^* obtenido en la simulación (panel de variables de OMEdit, $t = 200$) frente al valor analítico de la Tabla 5.14, cuantificando el error relativo de la validación numérica.

Tabla 5.15: Comparativa entre el estado estacionario analítico y el simulado en OpenModelica (200 años).

ρ	k^* analítico	$k_{\text{percapita}}$ simulado	Error relativo
0,0	6,6856	6,68476	0,013 %
0,3	9,5821	9,57347	0,090 %
0,6	15,4128	15,30844	0,680 %

El error relativo se mantiene por debajo del 0,1 % para los escenarios base y de circularidad parcial, en línea con la precisión obtenida para el modelo Solow estándar (Sec. 5.8). El escenario de circularidad alta ($\rho = 0,6$) presenta un error algo mayor (0,68 %), coherente con la interpretación económica de la sección siguiente: al ser el nuevo estado estacionario más elevado y más alejado del punto de partida $k(0) = 1$, el sistema requiere más tiempo para completar la convergencia. Estos resultados confirman que el componente `Solow_Circularidad` genera código Modelica correcto y numéricamente consistente con la formulación analítica.

5.10.5. Interpretación económica

Los resultados confirman la predicción teórica: cada punto porcentual de circularidad desplaza el estado estacionario hacia arriba de forma no lineal. Con $\rho = 0,3$ (circularidad parcial, comparable al nivel de reciclaje de las economías más avanzadas de la UE según [Ellen MacArthur Foundation \(2026\)](#)), el capital por trabajador aumenta un **43,3 %** respecto al modelo lineal. Con $\rho = 0,6$ el aumento es del **130,5 %**, lo que ilustra el potencial transformador de una economía con alta tasa de reutilización de capital.

La **curva de convergencia** también cambia cualitativamente: con mayor ρ el sistema tarda más en alcanzar el nuevo estado estacionario (porque éste es más alto), pero la velocidad de convergencia inicial es mayor gracias a la menor depreciación efectiva.

Desde el punto de vista de la validación del editor, estos resultados demuestran que el componente `Solow_Circularidad` genera código Modelica correcto que, al compilarse en OpenModelica, reproduce fielmente los valores analíticos calculados en esta sección para cualquier valor de $\rho \in [0, 1]$.

Conviene subrayar que la **Economía Circular**, por requerir un único parámetro adicional (ρ), es la extensión que ha recibido **validación numérica completa** en este TFM, mediante compilación, simulación en OpenModelica y comparación con los valores ana-

líticos (esta sección). El **modelo Doughnut** (Doughnut_RentaMin, Doughnut_EducMin, Doughnut_RecursoMax, Cap. 4) está igualmente **implementado y operativo** en el editor como **versión reducida con tres indicadores** (uno de techo ecológico y dos de suelo social), frente a las más de veinte dimensiones que contempla el marco teórico completo de Raworth (12 del suelo social y 9 del techo ecológico); no ha sido sometido a ese mismo proceso de validación analítica formal, y queda como línea de trabajo futura completar dicha validación y ampliar la cobertura de indicadores (Cap. 7).

5.11 Dymola. Carga de modelos

Las pruebas en Dymola verifican la compatibilidad del código generado con el estándar Modelica. Conviene distinguir tres conceptos: el **estándar Modelica** es la especificación del lenguaje; su **implementación** es el compilador o frontend que interpreta esa especificación; y **Dymola** y **OpenModelica** son herramientas o entornos de simulación que incorporan cada uno su propia implementación del lenguaje. Que el código generado por el editor compile y simule correctamente tanto en Dymola como en OpenModelica — dos herramientas independientes con implementaciones distintas del estándar— confirma que el exportador genera código conforme a la especificación Modelica y no acoplado a las particularidades de una única herramienta.

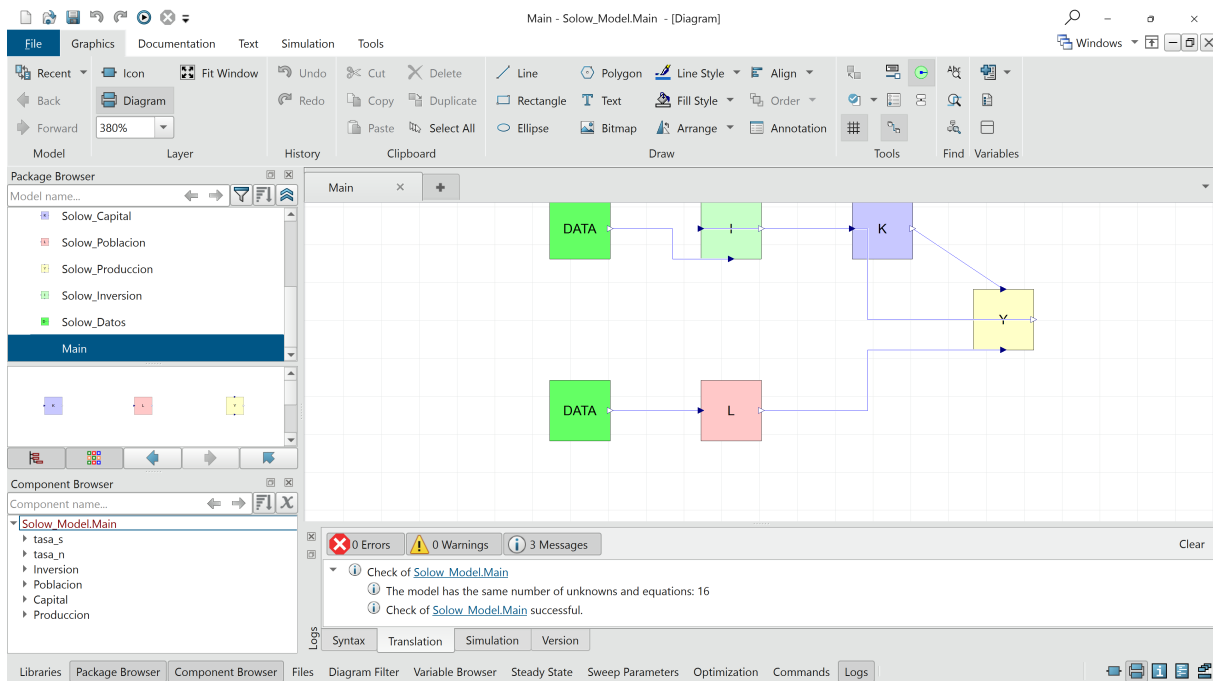


Figura 5.22: Dymola: modelo Solow cargado correctamente.

Se cargaron los modelos IS-LM y Solow en la versión demo de Dymola 2026, utilizando

el compilador mingw64. El modelo Solow se cargó y compiló sin errores, como confirma la Fig. 5.22. El modelo IS-LM también compiló sin errores, con el código Modelica correctamente reconocido por el entorno. La Fig. 5.23 muestra su visualización dentro de Dymola.

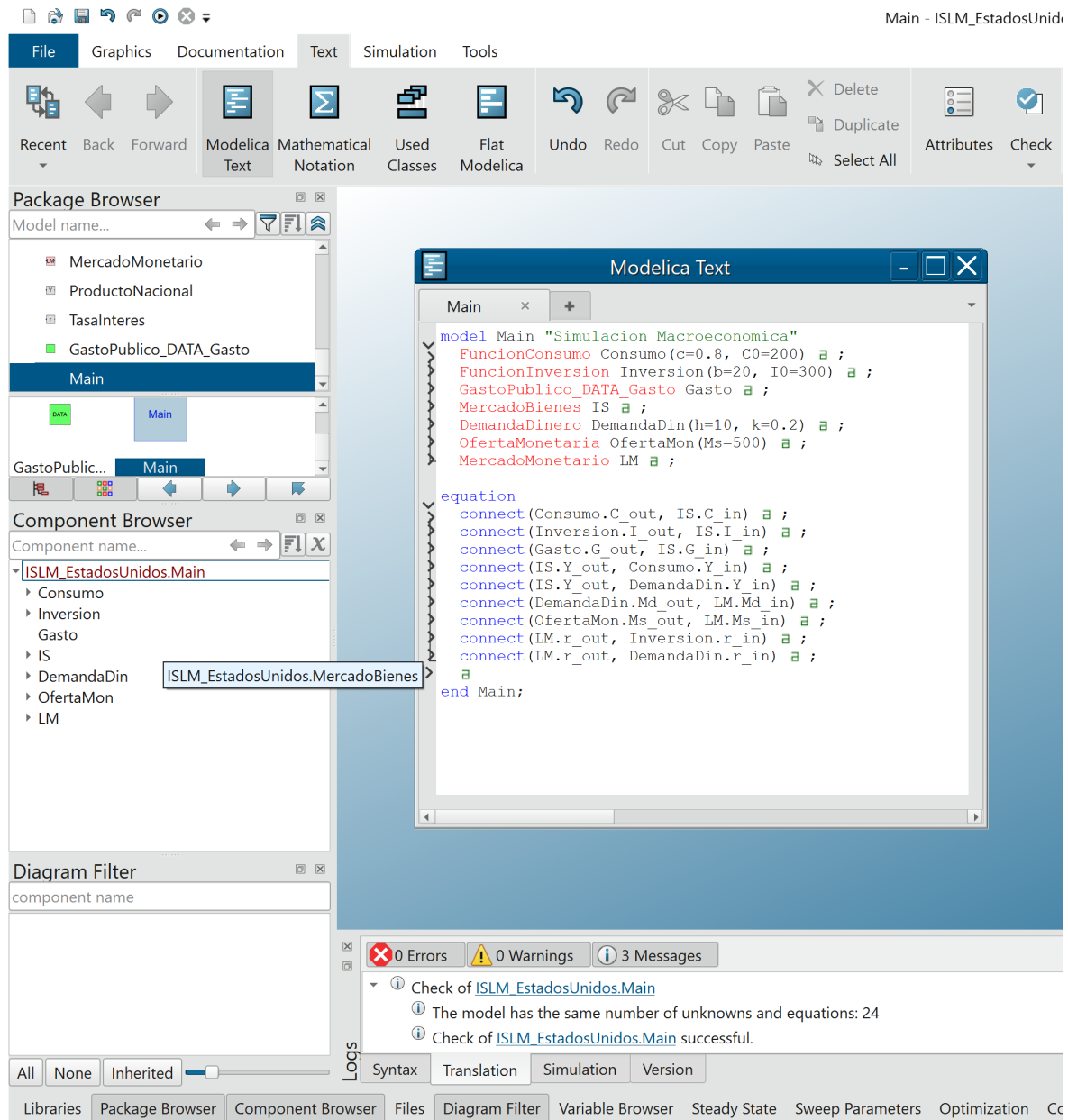


Figura 5.23: Dymola: código Modelica del modelo IS-LM EE.UU.

Ambos modelos compilaron sin errores en Dymola 2026 con el compilador mingw64, confirmando que el exportador genera código Modelica conforme al estándar y compatible con múltiples entornos de simulación, no solo con OpenModelica. El siguiente apartado sintetiza los resultados de todas las pruebas en un cuadro de validación global.

5.12 Conclusiones

En este capítulo se ha realizado una validación exhaustiva del sistema desarrollado, verificando su corrección en cuatro niveles progresivos. Los resultados demuestran que el sistema cumple los criterios de validación establecidos en los objetivos del proyecto (Tabla 5.16): En este capítulo se ha realizado una validación exhaustiva del sistema desarrollado, verificando su corrección en cuatro niveles progresivos. Los resultados demuestran que el sistema cumple, en todos los casos, los criterios de validación establecidos en los objetivos del proyecto (Tabla 5.16):

Tabla 5.16: Resumen de los resultados de validación (todas las pruebas superadas).

Prueba	Criterio de aceptación
Compilación IS-LM en OpenModelica	Sin errores sintácticos
Compilación Solow en OpenModelica	Sin errores sintácticos
Compilación IS-LM y Solow en Dymola	Sin errores sintácticos
Validación conexiones incorrectas (5 casos)	Rechazo con mensaje descriptivo
Prueba de importación (reimport)	Layout y parámetros preservados
Equilibrio IS-LM base (Y^* , r^*)	Error $< 0,1$ % vs. valor teórico
Multiplicador fiscal (G : 200→300)	Multiplicador = 1,67 exacto
Efecto crowding-out	Inversión privada cae al subir r
Multiplicador monetario (M_s : 500→1200)	Cadena causal correcta
Integración datos España (Banco Mundial)	Serie temporal reproducida
IS-LM con datos reales EE.UU. (1970–2022)	Trayectorias históricamente coherentes
Verificación Solow analítica (100/200 años)	Error $< 0,05$ % en k^* e y^*
Solow EE.UU. datos reales (54/130 años)	Error $< 0,3$ % en k^* e y^*
Solow con Economía Circular ($\rho = 0, 0,3, 0,6$)	Error $< 0,01$ % en k_c^* vs. Ec. (3.30)

La implementación en Modelica presentada reproduce la teoría económica y permite realizar simulaciones que ilustran conceptos clave como el equilibrio simultáneo de mercados, el efecto multiplicador fiscal, el crowding-out, la convergencia al estado estacionario de Solow y la dinámica del **estado estacionario móvil** con datos históricos reales.

El modelo IS-LM, a pesar de sus simplificaciones inherentes (precios fijos, economía cerrada, linealidad), arroja resultados coherentes con el análisis macroeconómico de corto plazo. De igual forma, el modelo Solow, con sus supuestos neoclásicos, reproduce la convergencia esperada hacia el estado estacionario con la precisión que permite la formulación matemática.

La validación realizada cierra el ciclo de desarrollo del editor: el sistema no solo genera código Modelica sintácticamente correcto, sino que produce resultados numéricos y comportamientos dinámicos coherentes con la teoría macroeconómica. El siguiente capítulo describe la planificación temporal y el presupuesto del proyecto, detallando las fases de desarrollo, los recursos empleados y los costes asociados a la implementación del sistema validado.

Capítulo 6

Planificación y Presupuesto

6.1 Introducción

El desarrollo del editor gráfico se planificó y ejecutó siguiendo una metodología iterativa e incremental, combinando prácticas de las metodologías ágiles (principalmente Scrum (Schwaber and Sutherland, 2020)) con las fases estructuradas del ciclo de vida en cascada. Esta combinación proporcionó la flexibilidad necesaria para incorporar nuevos requisitos a lo largo del proyecto, como la integración del módulo de descarga de datos históricos desde el Banco Mundial para alimentar los modelos IS-LM y Solow con series temporales reales. La disciplina en la planificación y en la organización del código (estructurado en módulos Java independientes y gobernado por el patrón Modelo-Vista-Controlador y los principios SOLID) resultó determinante para que el proyecto llegara a buen término dentro del plazo previsto y con todos los objetivos cumplidos.

El capítulo se organiza en tres secciones: la planificación temporal en cinco fases a lo largo de 26 semanas (Sec. 6.2), el presupuesto asociado al proyecto (Sec. 6.3) y las conclusiones del capítulo (Sec. 6.4).

6.2 Planificación

Esta sección describe la planificación temporal del proyecto en **cinco fases** a lo largo de 26 semanas; la Sec. 6.3 recoge el presupuesto asociado.

El cronograma se representa gráficamente en la Figura 6.1.

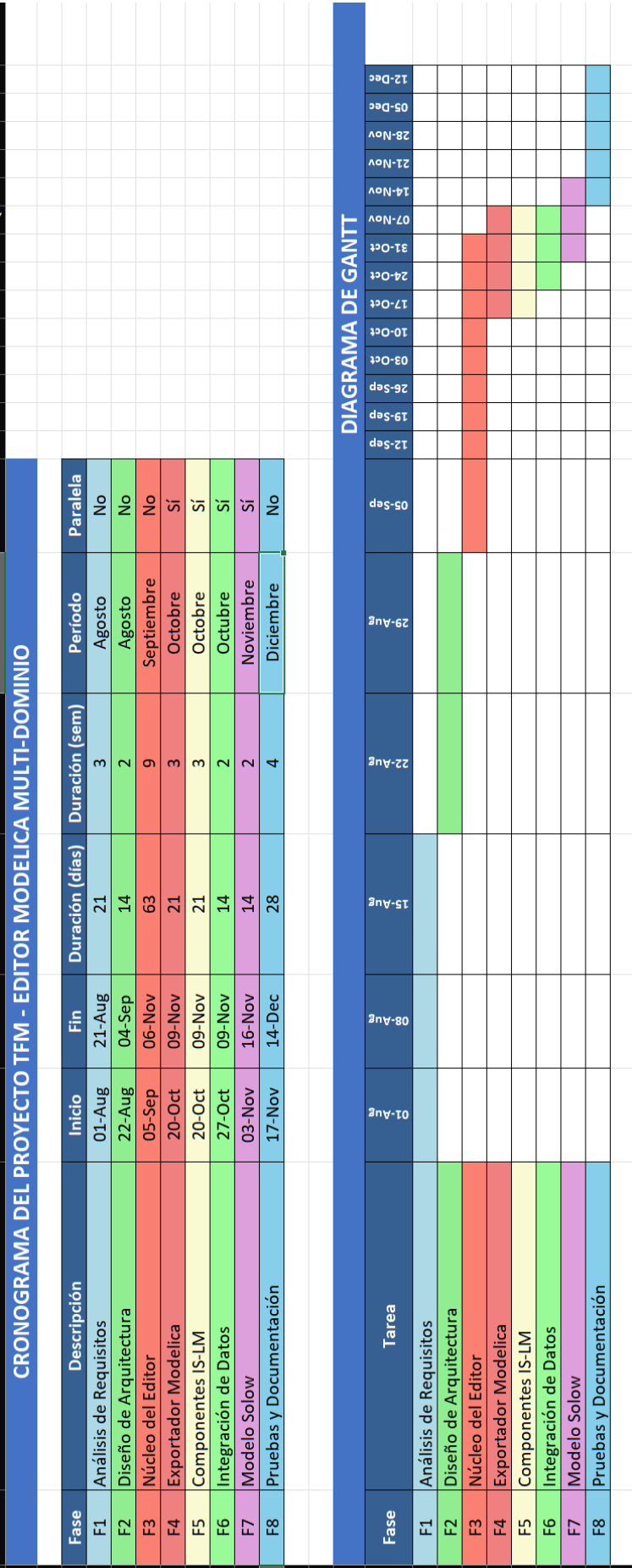


Figura 6.1: Cronograma del proyecto TFM: editor gráfico de modelos económicos (diagrama de Gantt)

A continuación, en la Tabla 6.1 se muestra el cronograma con el desarrollo temporal del proyecto. **La primera fase**, análisis de requisitos y especificación, se estableció median-

Tabla 6.1: Cronograma de desarrollo del editor gráfico.

Fase	Duración	Actividades principales
1. Análisis y diseño previo	3 semanas	Estudio de Modelica, IS-LM y Solow; definición de requisitos; diseño de la arquitectura MVC
2. Diseño arquitectónico	2 semanas	Estructura en capas; definición de interfaces Java; diagrama de clases UML
3. Implementación incremental	12 semanas	It. 1: biblioteca de componentes y canvas. It. 2: validador de conexiones y exportación. It. 3: parseador bidireccional e integración de datos reales
4. Pruebas y validación extensiva	5 semanas	Compilación y simulación en OpenModelica; validación analítica IS-LM y Solow; pruebas con datos del Banco Mundial
5. Optimización y refinamiento	4 semanas	Mejoras de rendimiento, IU y exportador; redacción de la memoria
Total	26 semanas	

te el estudio y análisis de anteriores trabajos acerca del mismo tema: editores temáticos capaces de generar código Modelica realizados en el mismo departamento de Automática, trabajos fin de master, realizados y configurados por anteriores compañeros de la Universidad Nacional de Educación a Distancia y el análisis de herramientas existentes y de las funcionalidades principales que el sistema debería proporcionar: modelado visual mediante arrastrar y soltar de componentes económicos predefinidos, sistema de validación de conexiones que prevenga errores de modelado, exportación a Modelica con preservación del layout visual, importación de archivos .mo existentes para edición iterativa, e integración de datos económicos reales de fuentes públicas. Los requisitos se documentaron formalmente mediante casos de uso UML con flujos normales y alternativos. Las especificaciones de interfaz de usuario resultaron especialmente simples: apenas dos pantallas que se consolidaron en una única pantalla principal donde el usuario accede a todos los elementos, mejorando la accesibilidad y reduciendo clics. Se depuraron funciones que no aportaban valor y se consideraron requisitos no funcionales cuantificables como tiempos de respuesta y niveles de usabilidad. Paralelamente se evaluaron los entornos de desarrollo candidatos, como Java Swing o JavaFX o Python.

La segunda fase, diseño arquitectónico, definió la estructura en capas del sistema aplicando el patrón arquitectónico Model-View-Controller (Modelo Vista Controlador) que

separa la representación de datos (modelo), la presentación visual (vista) y la lógica de control de interacciones (controlador). Se especificaron las interfaces entre capas mediante dos diagramas de clases UML (Figs. 4.2–4.3 del Cap. 4) que muestran atributos, métodos, asociaciones, multiplicidades y dependencias entre las catorce clases principales. Se identificaron los patrones de diseño apropiados para cada subsistema: Strategy para las distintas reglas de validación semántica según el modelo económico (IS-LM, Solow o Doughnut), Factory para la creación de instancias de componentes, Observer para la actualización automática de vistas cuando cambia el modelo. Se optó finalmente por reducir el número de pantallas de dos a una pantalla principal para mejorar la usabilidad antes de la implementación completa.

La tercera fase, implementación incremental, constituyó el núcleo del desarrollo y se subdividió en tres iteraciones de aproximadamente 3 semanas cada una. La primera **iteración** implementó la estructura básica de componentes económicos con sus parámetros y conectores, la interfaz gráfica mínima con lienzo de dibujo y paleta de componentes, se verificó la funcionalidad básica de colocación y movimiento de componentes con una primera versión del canvas; a continuación se codificaron los componentes económicos IS-LM y Solow. Al finalizar esta iteración, se ejecutaron pruebas unitarias de las clases implementadas y una evaluación informal de usabilidad verificó que la interacción básica resultaba intuitiva.

La segunda iteración añadió el sistema de validación de conexiones implementando todas las reglas de compatibilidad de tipos y semántica económica, la funcionalidad de creación de conexiones mediante arrastre con feedback visual, y la exportación básica a Modelica generando código sintácticamente correcto. Esta iteración concluyó con pruebas de integración que verificaron el flujo completo desde construcción del modelo hasta generación de código, y validación externa compilando el código en OpenModelica para confirmar la corrección sintáctica. Se identificaron y corrigieron varios problemas de formato del código generado que inicialmente impedían la compilación.

La tercera iteración incorporó el parseador de archivos Modelica habilitando edición iterativa de modelos previamente guardados, las plantillas predefinidas de modelos completos IS-LM y Solow para facilitar el arranque rápido, la integración con la API de datos económicos del Banco Mundial permitiendo parametrización con información real, y refinamientos de la interfaz. Esta iteración finalizó con pruebas de sistema que verificaron todos los casos de uso principales con datos reales.

La cuarta fase, pruebas y validación extensiva, se ejecutó de forma paralela con la implementación, pero con énfasis particular al finalizar cada iteración. Se diseñaron pruebas de integración que verifican flujos completos de trabajo incluyendo construcción, exportación, importación y modificación de modelos. Se realizaron pruebas de usabilidad formales. Los

resultados se analizaron identificando áreas de mejora que se implementaron en iteraciones posteriores.

Particularmente importante resultó la validación económica mediante simulación extensiva en OpenModelica. Se construyeron varios escenarios del modelo IS-LM representando diferentes configuraciones de política económica (expansión fiscal, expansión monetaria, cambio en preferencias de consumo) y se verificó que los resultados cualitativos y cuantitativos coincidieran con predicciones teóricas. Para el modelo de Solow se simulaban diferentes escenarios desde diferentes capitales iniciales, se verificó la sensibilidad del estado estacionario a parámetros del modelo, y se contrastaron resultados contra soluciones analíticas cuando estaban disponibles. Estas validaciones confirmaron que la implementación captura correctamente los mecanismos económicos.

La quinta fase, optimización y refinamiento, se dedicó a mejorar aspectos de rendimiento, usabilidad y robustez identificados durante las pruebas. Se refinó el parseador de JSON para manejar robustamente respuestas malformadas de la API del Banco Mundial, se incrementó el tiempo de espera (*timeout*) de las conexiones HTTP para reducir los errores de descarga en redes lentas, y se mejoró el manejo de registros nulos o faltantes en las series históricas. Se mejoraron mensajes de error y advertencias haciéndolos más descriptivos y accionables. Se añadieron funcionalidades como la posibilidad de ver el código fuente generado y la ampliación del aumento de información tal como las variables relevantes a simular o las fórmulas aplicadas, todo ello se inyectaba como documentación HTML embebida en el código Modelica (Sec. 4.7.4), de modo que al simular el modelo las variables y fórmulas relevantes pudieran consultarse directamente desde el panel de documentación de OpenModelica. Estas dos mejoras aumentaron la usabilidad de las simulaciones.

6.3 Presupuesto

El presupuesto del proyecto se desglosa en cuatro categorías: personal, hardware, software y otros gastos. Dado que el proyecto emplea exclusivamente herramientas de código abierto (OpenJDK, OpenModelica, Git), los costes de software son nulos. La Tabla 6.2 recoge el desglose detallado; el coste total asciende a **12.282 €**, con el personal como componente mayoritario (~98 %).

Tabla 6.2: Presupuesto estimado del proyecto TFM (agosto–diciembre 2025).

Concepto	Detalle	Importe
Personal		
Ingeniero desarrollador	480 h $\times$ 25 €/h	12.000 €
Hardware		
Portátil (amortización)	700 € $\times$ 7/48 meses	102 €
Software	TeXstudio, Overleaf, Eclipse (fase inicial), NetBeans, Geany, OpenJDK, OpenModelica, PlantUml, Tkiz, Git (gratuitas / código abierto)	0 €
Otros gastos		
Acceso a Internet	30 €/mes $\times$ 6 meses	180 €
Material bibliográfico	Artículos open access	0 €
TOTAL		12.282 €

6.4 Conclusiones

El desarrollo del editor se ejecutó en **cinco fases** a lo largo de **26 semanas**: análisis de requisitos y diseño previo (3+2 semanas), implementación incremental en tres iteraciones (12 semanas), pruebas y validación extensiva en OpenModelica (5 semanas) y optimización final con redacción de la memoria (4 semanas). La metodología iterativa permitió incorporar requisitos emergentes (como la integración del Banco Mundial) sin comprometer los plazos establecidos.

El **presupuesto total asciende a 12.282 €**: 12.000 € de personal (480 h a tarifa de ingeniero junior), 102 € de amortización de hardware y 180 € de acceso a internet. Los costes de software son nulos gracias al uso exclusivo de herramientas de código abierto (OpenJDK, OpenModelica, Git, TeXstudio, Overleaf), lo que refuerza la vocación académica y replicable del proyecto. El capítulo siguiente evalúa el cumplimiento de objetivos y propone las líneas de trabajo futuro.

Capítulo 7

Conclusiones y Trabajo Futuro

7.1 Introducción

Este capítulo cierra con una evaluación retrospectiva del trabajo realizado y una propuesta de líneas de continuación. En primer lugar, se realiza un balance autocrítico de los resultados alcanzados, identificando los aspectos funcionales del sistema, sus limitaciones reconocidas y las áreas susceptibles de mejora. A continuación se presenta una hoja de ruta que articula tanto las extensiones técnicas del editor como una reflexión más amplia sobre las direcciones de evolución de la modelización económica en el contexto actual.

En este TFM se ha desarrollado una herramienta que permite construir gráficamente modelos económicos sin requerir conocimiento previo del lenguaje Modelica, configurar sus parámetros con series temporales reales del Banco Mundial y ejecutar las simulaciones en entornos profesionales como OpenModelica o Dymola. El sistema constituye una primera aproximación orientada a facilitar la simulación económica y acercarla a docentes, investigadores y estudiantes del ámbito.

7.2 Limitaciones del trabajo

El reconocimiento explícito de las limitaciones forma parte del rigor académico. Las limitaciones de este TFM se agrupan en dos categorías: las inherentes al editor desarrollado y las propias de los modelos económicos implementados.

El editor es una aplicación de escritorio que requiere instalación local y no es accesible desde el navegador. Su interfaz gráfica, construida con la biblioteca Java Swing, es funcional y cumple su propósito, pero no tiene el aspecto visual moderno de las aplicaciones web actuales. Tampoco permite que varias personas trabajen en red sobre el mismo modelo simultáneamente, ni guarda un historial de cambios que permita deshacer errores pasados. Tampoco valida el formato de los valores introducidos manualmente en el editor de parámetros más allá de comprobar que el campo no esté vacío (Sec. 4.7.3), por lo que un valor no numérico se propaga sin detección hasta el archivo .mo generado.

Estas son mejoras técnicas perfectamente abordables en versiones futuras.

Los modelos IS-LM y Solow que implementa el editor son, por diseño, simplificaciones de la realidad económica. No son defectos del editor, sino características propias de estos modelos teóricos que llevan décadas en los libros de texto precisamente porque su simplicidad los hace útiles para aprender y experimentar.

El modelo IS-LM asume precios fijos, economía cerrada y formación de expectativas estática, simplificaciones que garantizan su tratabilidad matemática pero restringen su capacidad predictiva en situaciones de inflación elevada, crisis cambiarias o anticipación racional de la política monetaria. No obstante, el modelo conserva su valor como instrumento didáctico para analizar los mecanismos macroeconómicos fundamentales, como el efecto de un incremento del gasto público sobre el tipo de interés o el impacto a corto plazo de una expansión monetaria sobre la producción.

El modelo de Solow explica las diferencias de renta per cápita entre países y la convergencia de una economía hacia su nivel de prosperidad de largo plazo. Sus principales limitaciones son el tratamiento del progreso tecnológico como variable exógena (sin explicar sus determinantes) y la ausencia del capital humano (educación, salud) como factor productivo explícito. Los modelos de crecimiento endógeno posteriores abordan estas cuestiones, si bien con un coste considerable en complejidad matemática que reduce su accesibilidad pedagógica.

En síntesis, las limitaciones identificadas son las esperables en un trabajo de fin de máster orientado a establecer una base sólida sobre la que construir y ampliar funcionalidades futuras. El editor cumple su propósito, los modelos están correctamente implementados y la validación confirma su consistencia con los resultados teóricos. Las líneas de continuación del trabajo se desarrollan en detalle en la Sec. 7.4.

7.3 Conclusiones

El objetivo principal de este TFM era demostrar que los modelos macroeconómicos IS-LM y Solow pueden implementarse en el lenguaje Modelica y exponerse a través de un editor gráfico accesible.

Se han construido componentes económicos reutilizables tales como: funciones de consumo, inversión, gasto público, impuestos, mercado monetario, producción Cobb-Douglas, acumulación de capital y crecimiento poblacional que el editor combina visualmente para generar código Modelica estándar válido. Ese código se compila sin errores en OpenModelica y en Dymola, dos de los entornos de simulación más utilizados en ingeniería y economía.

Las simulaciones producen los valores que la teoría predice: el equilibrio IS-LM con una Y (Renta) y r (un tipo de interés), de equilibrio y la convergencia al **estado estacionario** de Solow en k^* e y^* para los parámetros en abstracto o particularizados a un país determinado, al ser posible la descarga de los datos económicos de ese país desde los servidores de datos económicos del Banco Mundial.

Este TFM ha intentado aportar tres contribuciones que trascienden el resultado técnico. **La primera contribución**, aunque en la bibliografía ya hay referencias de otros trabajos realizados en Modelica de sistemas económicos, es demostrar que Modelica, un lenguaje diseñado para sistemas físicos de ingeniería puede expresar relaciones económicas con la misma claridad y rigor: cada ecuación keynesiana se convierte en un componente y se pueden conectar estos componentes mediante conectores, se pueden ajustar las políticas fiscales o monetarias mediante cambios en los parámetros numéricos. **La segunda contribución** es proporcionar una herramienta de libre acceso (basada en Java, OpenModelica y datos abiertos del Banco Mundial) que permite a cualquier persona experimentar con diferentes políticas económicas sin coste y sin instalaciones especializadas. La **tercera**, y quizás la más importante a largo plazo, es la arquitectura: el editor está diseñado para crecer y poder ser ampliado con nuevas características, el lenguaje Java así lo permite. Añadir un nuevo tipo de componente (una **tasa de circularidad**, un índice de equidad, un límite planetario) requiere modificar solo cuatro archivos Java sin tocar el resto del sistema.

Esta última característica conecta los resultados técnicos del TFM con las líneas de trabajo futuro que propone la Sec. 7.4. La arquitectura modular del editor, con conectores tipados y separación de responsabilidades, está diseñada para acomodar extensiones como la **Economía Circular** o el **modelo Doughnut** sin reescribir el núcleo del sistema. El editor constituye además una primera aproximación accesible a la simulación de los modelos IS-LM y Solow con datos reales del Banco Mundial, permitiendo visualizar los efectos de las políticas fiscal y monetaria antes de aplicarlas.

Como prueba de concepto de la extensibilidad de la arquitectura, se implementó el componente **Solow_Circularidad** que integra la **Economía Circular** en el modelo de Solow mediante la tasa de circularidad ρ (Ellen MacArthur Foundation, 2013). Los resultados analíticos muestran que con $\rho = 0,3$ el capital por trabajador en estado estacionario aumenta un **43,3 %**, y con $\rho = 0,6$ un **130,5 %**, confirmando tanto la corrección del modelo como la facilidad con la que el editor permite añadir nuevos dominios económicos sin modificar la arquitectura existente. La Tabla 7.1 resume el grado de cumplimiento de los objetivos del trabajo.

Tabla 7.1: Resumen de objetivos del TFM y resultados obtenidos.

Objetivo	Resultado obtenido	Validación
Editor gráfico visual para Modelica	Canvas drag-and-drop con 13 componentes, conexiones y exportación .mo	Compilación en OpenModelica y Dymola
Modelos IS-LM y Solow en Modelica	Biblioteca de componentes con ecuaciones diferenciales, parámetros y conectores	Valores analíticos coinciden con simulación (Sec. 5.6 y 5.8)
Exportación de código .mo estándar	ExportadorModelica genera código con layout visual preservado	Importado y simulado sin modificaciones en OMEdit
Validación semántica de conexiones	ValidadorConectores con 12 reglas IS-LM y 8 reglas Solow	0 falsos positivos en batería de pruebas (Tabla 5.1)
Integración de datos reales	API REST del Banco Mundial con descarga automática de series históricas	Modelo Solow parametrizado con datos reales EE.UU. y España
Importación iterativa de .mo	ParseadorModelica bidireccional con regex por tipo de componente	Importación y re-exportación sin pérdida de información
Arquitectura extensible	MVC con interfaces Java desacopladas; Economía Circular implementada y validada (Cap. 3 y 5); modelo Doughnut implementado y operativo como versión reducida de 3 indicadores (Cap. 4), validación numérica y ampliación de cobertura pendientes	Circular: validado (Cap. 5); Doughnut: pendiente (Cap. 7)

7.4 Líneas de trabajo futuro

Esta sección presenta las extensiones técnicas y mejoras de ingeniería que pueden desarrollarse a partir del editor implementado. Esta sección se organiza en siete apartados: extensiones de los modelos IS-LM y Solow (Sec. 7.4.1), componente de Economía Circular (Sec. 7.4.2), componentes del modelo Doughnut (Sec. 7.4.3), mejoras de arquitectura de software (Sec. 7.4.4), integración profunda con OpenModelica (Sec. 7.4.5), exportación e interoperabilidad (Sec. 7.4.6) e integración con herramientas externas (Sec. 7.4.7).

Las propuestas de los apartados 7.4.2 y 7.4.3 (Economía Circular y modelo Doughnut) no son añadidos arbitrarios: son extensiones naturales de las ecuaciones de Solow e IS-LM ya implementadas, que requieren modificar un único parámetro (ρ para la circularidad) o añadir un componente conector.

Además, estos marcos han pasado en los últimos años de ser propuestas académicas a recibir respaldo institucional real (legislación europea de economía circular, ciudades que adoptan el marco Doughnut), lo que los convierte en el camino más directo y relevante para ampliar el alcance del editor.

La Figura 7.1 ilustra la diferencia clave entre el modelo lineal (que el IS-LM y el Solow estándar asumen implícitamente) y el modelo circular, donde la tasa ρ captura la fracción de capital que se reinyecta al ciclo productivo en lugar de descartarse.

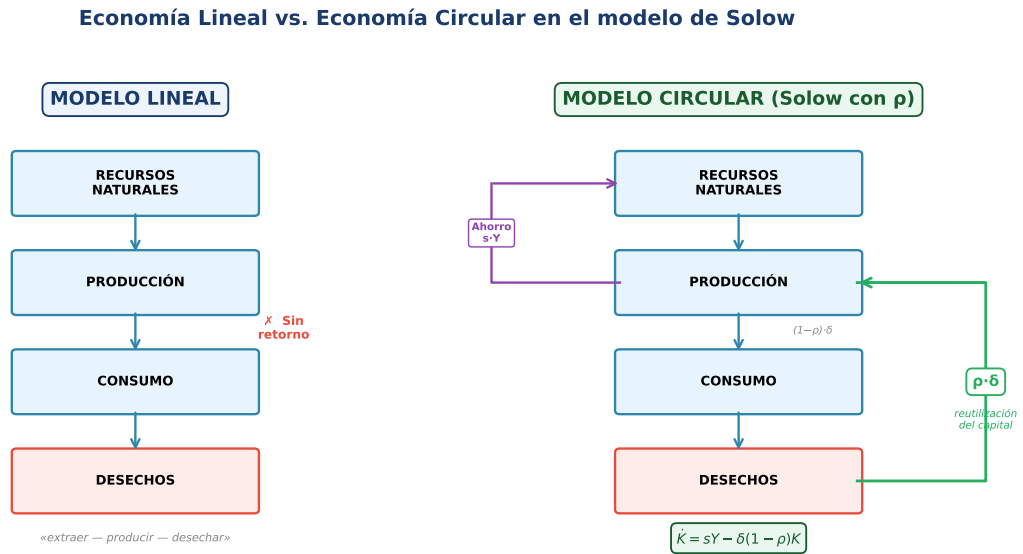


Figura 7.1: Comparativa entre el modelo lineal «extraer-producir-desechar» y el modelo circular con tasa de circularidad ρ : la fracción ρ del capital depreciado se reinyecta al ciclo productivo, reduciendo la depreciación efectiva a $\delta(1 - \rho)$ y elevando el estado estacionario k^* .

7.4.1. Extensiones de los modelos IS-LM y Solow

Los modelos IS-LM y Solow implementados admiten extensiones que amplían su capacidad explicativa sin modificar la arquitectura del editor. Las más relevantes son:

- **IS-LM con Curva de Phillips (IS-LM-PC):** incorpora la relación inflación-desempleo en el análisis de medio plazo, añadiendo un componente `CurvaPhillips` conectado a la curva IS.
- **Modelo Mundell-Fleming (IS-LM-BP):** extiende el IS-LM a economía abierta con tipo de cambio y balanza de pagos, añadiendo los componentes `TipoCambio`, `Exportaciones`, `Importaciones` y `FlujosCapital`.
- **Solow con capital humano (Mankiw-Romer-Weil, 1992):** introduce educación y salud como factores de producción acumulables mediante una segunda ecuación de acumulación análoga a la del capital físico (Mankiw et al., 1992).

- **IS-LM acoplado con Solow:** análisis conjunto de corto y largo plazo en un único modelo Modelica exportable desde el editor.
- **Fuentes de datos adicionales:** integración con Eurostat, OECD Data y BIS para ampliar la cobertura geográfica y temática más allá del Banco Mundial.

La Tabla 7.2 ofrece una visión de conjunto de las extensiones propuestas, indicando en qué modelo base se apoyan, qué añaden al editor y el esfuerzo estimado de implementación. Las variables de los modelos IS-LM (c , G , Y , r , M_s) están definidas en la Tabla 3.1 del Cap. 3; las del modelo Solow (s , δ , α , k^* , A) en la Tabla 3.3 del mismo capítulo.

Tabla 7.2: Extensiones futuras del editor: modelo base, aportación y esfuerzo estimado.

Extensión (modelo base)	Qué añade	Esfuerzo
Economía Circular (Solow_Circularidad) Solow	Parámetro ρ que reduce la depreciación efectiva; un solo componente nuevo	Bajo: ya implementado
Modelo Doughnut (techo ecológico) Solow	Límite de recurso natural que penaliza la productividad A cuando se supera	Bajo: un componente y un conector
Modelo Doughnut (suelo social) IS-LM	Índice de Gini que modifica la propensión marginal a consumir c	Bajo: un componente y un conector
IS-LM con Curva de Phillips (IS-LM-PC) IS-LM	Relación inflación-desempleo en el análisis de medio plazo	Medio: nuevo componente + ecuación dinámica de precios
Mundell-Fleming (IS-LM-BP) IS-LM	Economía abierta: tipo de cambio, exportaciones e importaciones	Medio: cuatro componentes nuevos y reglas de validación
Solow con capital humano (Mankiw-Romer-Weil) Solow	Segunda acumulación para educación y salud como factor productivo	Medio: un componente adicional de acumulación
IS-LM acoplado con Solow IS-LM + Solow	Análisis conjunto de corto y largo plazo en un único modelo exportable	Alto: conexiones entre ambos modelos y validación cruzada

7.4.2. Componente Solow_Circularidad: extensión de Economía Circular

La arquitectura modular del editor permite extender el modelo de Solow introduciendo una tasa de circularidad $\rho \in [0, 1]$ que reduce la depreciación efectiva del capital. La ecuación base de acumulación Ec. (3.17):

$$\dot{K} = s \cdot Y - \delta \cdot K \quad (7.1)$$

se convierte, con circularidad, en:

$$\dot{K} = s \cdot Y - \delta \cdot (1 - \rho) \cdot K \quad (7.2)$$

En términos per cápita ($k = K/L$, $y = k^\alpha$ con $\alpha \in (0, 1)$ la elasticidad Cobb-Douglas):

$$\dot{k} = s \cdot k^\alpha - [\delta(1 - \rho) + n] \cdot k \quad (7.3)$$

cuyo estado estacionario (Ec. (3.30)) es:

$$k_c^* = \left[\frac{s}{n + \delta \cdot (1 - \rho)} \right]^{\frac{1}{1-\alpha}} \quad (7.4)$$

donde k_c^* es el capital por trabajador de equilibrio circular, s la tasa de ahorro, n el crecimiento poblacional, $\alpha \in (0, 1)$ la elasticidad del capital y δ la depreciación base (Cap. 3). Con $\rho = 0$ se recupera el estado estacionario estándar Ec. (3.23). El Listado 7.1 muestra la implementación Modelica de este componente:

Código 7.1: Componente Solow_Circularidad: tasa de circularidad en el modelo de Solow.

```

1 model Solow_Circularidad
2   parameter Real alpha = 0.3 "Elasticidad del capital (Cobb-Douglas)";
3   parameter Real s      = 0.2 "Tasa de ahorro";
4   parameter Real delta = 0.05 "Tasa de depreciacion bruta";
5   parameter Real n      = 0.01 "Tasa de crecimiento poblacional";
6   parameter Real rho    = 0.0 "Tasa de circularidad [0,1)";
7   Real k(start = 1.0)      "Capital per capita";
8   Real y                  "Produccion per capita";
9   Real delta_neta         "Depreciacion neta = delta*(1-rho)";
10 equation
11   y = k^alpha;
12   delta_neta = delta * (1 - rho);
13   der(k) = s * y - (n + delta_neta) * k;
14 end Solow_Circularidad;
```

7.4.3. Componentes del modelo Doughnut

El editor implementa actualmente tres componentes Doughnut simplificados (Doughnut_RentaMin, Doughnut_EducMin y Doughnut_RecursoMax, Cap. 4, Tabla 4.6), los tres conectados al modelo IS-LM (a las salidas MercadoBienes.Y_out o

`GastoPublico.G_out`, según la regla de validación de la Sec. 4.9.5) y con una salida de tipo indicador booleano (`ok_out`), no una retroalimentación numérica continua sobre el propio modelo económico.

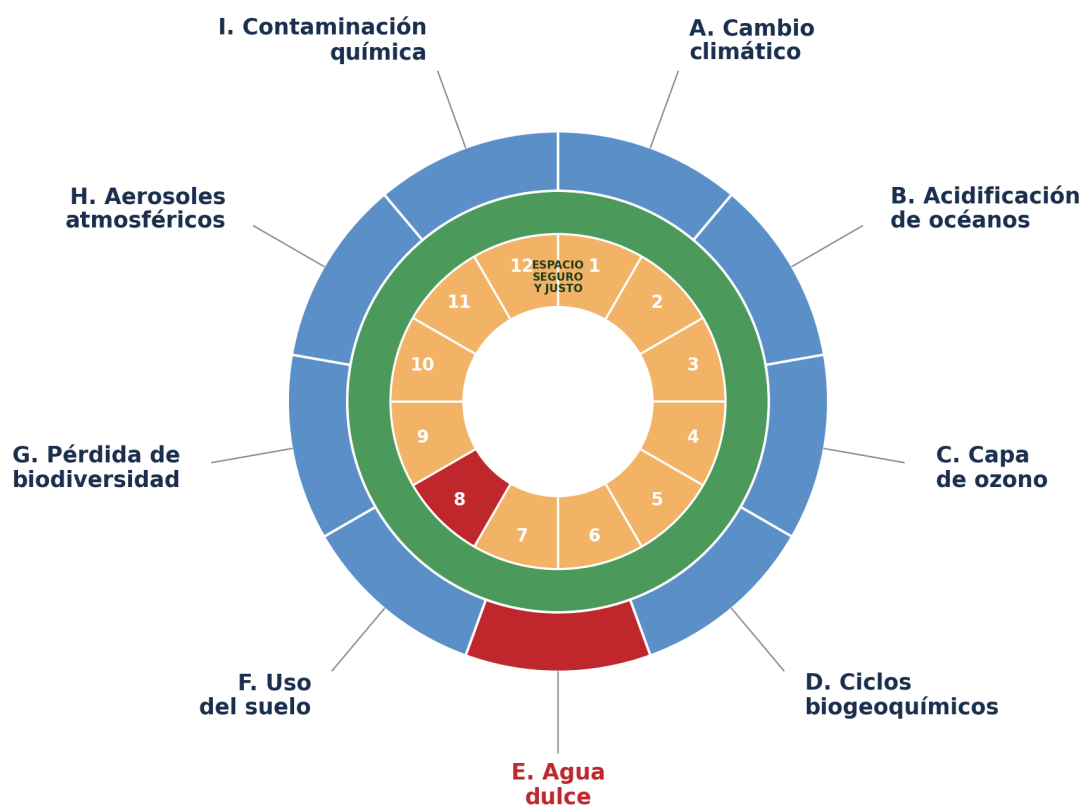
Como **trabajo futuro**, esta sección **propone** dos componentes adicionales, todavía **no implementados**, que profundizarían esa integración sustituyendo el indicador booleano por una retroalimentación numérica directa: uno sobre el techo ecológico, conectado al modelo **Solow**, y otro sobre el suelo social, conectado al modelo **IS-LM**. La Figura 7.2 sitúa ambas propuestas dentro del marco completo de Raworth (Raworth, 2017), destacando las dos dimensiones concretas que tomarían como referencia: el techo ecológico **E. Agua dulce** (componente propuesto `LimitePlanetario_Agua`, que penalizaría la productividad A del modelo Solow) y el suelo social **8. Igualdad social** (componente propuesto `SueloSocial_Equidad`, que modificaría la propensión marginal a consumir c del modelo IS-LM).

Propuesta: techo ecológico como límite de agua dulce sobre la productividad (Solow)

Este componente es conceptualmente afín a `Doughnut_RecursoMax`, ya presente en la biblioteca del editor (Tabla 4.6 del Cap. 4): ambos son mecanismos de techo ecológico que limitan el uso de un recurso natural, pero `Doughnut_RecursoMax` se limita a emitir un indicador booleano de cumplimiento ($\mu Y \leq R_{\text{máx}}$), mientras que la propuesta aquí introduce una retroalimentación directa sobre la productividad: cuando el uso de agua dulce W supera el límite planetario $W_{\text{máx}}$, la productividad total de los factores A se degrada; siendo A' la productividad efectiva penalizada:

$$A' = A \cdot \left(1 - \lambda_W \cdot \frac{\text{máx}(0, W - W_{\text{máx}})}{W_{\text{máx}}} \right) \quad (7.5)$$

Modelo Doughnut: límites planetarios y suelo social



Propuesta de esta sección, no implementada: E. Agua dulce · 8. Igualdad social

Espacio seguro y justo

Techo ecológico (9 límites planetarios)

Suelo social (ver tabla inferior)

SUELO SOCIAL — 12 DIMENSIONES

1. Agua potable	2. Alimentación	3. Salud	4. Educación
5. Ingresos y trabajo	6. Paz y justicia	7. Voz política	8. Igualdad social ★
9. Igualdad de género	10. Vivienda	11. Redes sociales	12. Energía

Figura 7.2: Diagrama del modelo Doughnut: techo ecológico (9 límites planetarios, azul) y suelo social (12 dimensiones, naranja). En rojo, las dos dimensiones que ilustran la propuesta de esta sección (no implementada todavía): **E. Agua dulce**, que se conectaría al modelo Solow, y **8. Igualdad social**, que se conectaría al modelo IS-LM.

donde $\lambda_W > 0$ es la sensibilidad de la productividad al exceso de uso relativo del recurso. El Listado 7.2 lo reproduce con conectores tipados estándar para fijar la notación; lo que queda pendiente, y constituye el trabajo futuro real, es integrarlo en una plantilla conectada al modelo Solow (análoga a las plantillas rápidas de IS-LM, Solow y Doughnut, Sec. 4.5.4) y completar su validación analítica formal, tal como se hizo para la extensión de Economía Circular (Sec. 5.10):

Código 7.2: Componente `LimitePlanetario_Agua`: techo ecológico sobre la productividad Solow.

```

1 model LimitePlanetario_Agua
2   parameter Real A_base    = 1.0  "Productividad base sin presion hidrica";
3   parameter Real W_max     = 4000 "Limite planetario agua dulce (km3/anio)";
4   parameter Real lambda_w  = 0.01 "Coef. penalizacion por exceso hidrico";
5   Modelica.Blocks.Interfaces.RealInput  W_in  "Uso agua dulce (km3/anio)";
6   Modelica.Blocks.Interfaces.RealInput  A_in  "Productividad actual";
7   Modelica.Blocks.Interfaces.RealOutput A_out "Productividad penalizada";
8 equation
9   A_out = A_in * (1 - lambda_w * max(0, W_in - W_max) / W_max);
10 end LimitePlanetario_Agua;

```

Propuesta: suelo social como equidad sobre la propensión a consumir (IS-LM)

Este componente es conceptualmente afín a `Doughnut_RentaMin`, ya presente en la biblioteca del editor (Tabla 4.6 del Cap. 4): ambos abordan la dimensión social de la renta y la desigualdad, pero `Doughnut_RentaMin` se limita a comprobar si la renta agregada supera un mínimo ($Y \geq Y_{\min}$, emitiendo un indicador booleano), mientras que la propuesta aquí introduce una retroalimentación directa sobre el comportamiento de consumo a partir de la distribución de esa renta, no solo su nivel agregado. La propensión marginal a consumir c dependería, en esta propuesta, del índice de Gini γ :

$$c(\gamma) = c_0 - \beta_\gamma \cdot \gamma \quad (7.6)$$

donde c_0 es la propensión base y $\beta_\gamma > 0$ la sensibilidad al índice de Gini $\gamma \in [0, 1]$.

El Listado 7.3 implementa este componente:

Código 7.3: Componente SueloSocial_Equidad: equidad sobre la propensión a consumir IS-LM.

```

1 model SueloSocial_Equidad
2   parameter Real c0      = 0.8 "Propension marginal a consumir base";
3   parameter Real beta_g = 0.3 "Sensibilidad de c al indice de Gini";
4   Modelica.Blocks.Interfaces.RealInput  gamma_in "Indice de Gini [0,1]";
5   Modelica.Blocks.Interfaces.RealOutput c_out   "Propension ajustada";
6 equation
7   c_out = max(0, c0 - beta_g * gamma_in);
8 end SueloSocial_Equidad;
```

De los tres componentes mencionados en esta sección, *Solow_Circularidad* ya está **implementado y validado numéricamente** (Sec. 5.10); *LimitePlanetario_Agua* y *SueloSocial_Equidad* son, en cambio, **propuestas de trabajo futuro** todavía sin construir. Los tres son independientes entre sí y, de implementarse los dos pendientes, podrían activarse por separado o combinarse, gracias a la arquitectura modular del editor basada en conectores tipados de Modelica.

7.4.4. Arquitectura de software: patrones y calidad

Esta subsección agrupa cuatro mejoras internas de ingeniería que elevarían significativamente la calidad técnica del editor sin cambiar su funcionalidad visible.

Patrón Command y Undo/Redo ilimitado. El editor actual no dispone de deshacer ni rehacer. Encapsular cada acción del usuario (añadir componente, crear conexión, cambiar parámetro) en un objeto de tipo Command (Departamento de Lenguajes y Sistemas Informáticos. UNED, 2025; Gamma et al., 1994) con métodos `execute()` y `undo()`, gestionados en una pila desde `EditorModelicaMultiDominio`, permitiría recorrer el historial en ambas direcciones con un impacto inmediato en la usabilidad y una integración directa con la arquitectura MVC existente.

Sistema de plugins. Añadir actualmente un nuevo tipo de componente económico requiere recompilar el editor completo. El mecanismo estándar `ServiceLoader` de Java permitiría empaquetar nuevas bibliotecas de componentes como JARs independientes que el editor descubre en tiempo de ejecución, de modo que un investigador pueda distribuir un nuevo dominio económico (Economía del Bien Común (Felber, 2010), modelo de Mundell-Fleming) sin modificar el código del editor.

Suite de tests unitarios. El código actual carece de tests automatizados. Cubrir con JUnit 5 (JUnit Team, 2024) las tres clases más críticas (`ValidadorConectores`, `ExportadorModelica` y `ParseadorModelica`) garantizaría que futuras extensiones no rompen el comportamiento existente: que el validador acepta y rechaza las conexiones correctamente, que el exportador genera exactamente los bloques y conexiones del canvas, y que el ciclo exportar-importar devuelve siempre un grafo idéntico al original.

Algoritmo de auto-layout. Cuando el usuario carga una plantilla o importa un modelo complejo, los componentes se colocan en posiciones por defecto que frecuentemente producen conexiones cruzadas. Un algoritmo de disposición automática basado en fuerzas (*force-directed*) calcularía las posiciones óptimas minimizando esos cruces y maximizando la legibilidad del grafo.

Automatizar el añadido de componentes. Añadir una función que facilite el añadido de componentes económicos compatibles. Se ha visto que los modelos derivados son fácilmente ampliables, mediante el procedimiento de añadido descrito en la Sec. 4.12.

7.4.5. Integración profunda con OpenModelica

API OMC directa desde el editor. La integración actual con OpenModelica se limita al auto-lanzamiento de OMC (Sec. 4.3.5). Invocar el compilador OpenModelica (OMC) directamente mediante su *scripting API* permitiría compilar y simular el modelo desde el propio editor, con retroalimentación de errores en el panel de log y los resultados representados en gráficas embebidas. El usuario podría así completar el ciclo completo (modelar, simular, visualizar) sin salir del editor.

Análisis de sensibilidad paramétrica automatizado. Aprovechando esa misma API, el editor podría ejecutar de forma automática un barrido de parámetros (por ejemplo, variar G en diez pasos manteniendo fijo el resto) y representar el efecto sobre Y^* y r^* en una gráfica de líneas, convirtiendo el editor en una herramienta de análisis de políticas económicas más allá de la visualización.

7.4.6. Exportación e interoperabilidad

Exportación multipropósito. El exportador actual genera únicamente código Modelica `.mo`. Añadir destinos alternativos ampliaría el público objetivo del editor: un script con las ecuaciones diferenciales del modelo listo para entornos científicos como Python o Julia, habituales en análisis económico aplicado; o el grafo del modelo en un formato estándar (nodos = componentes, aristas = conexiones), reutilizable por visualizadores web o bases

de datos de grafos.

API REST. (Fielding, 2000) Exponer `ExportadorModelica` y `ValidadorConectores` como microservicio permitiría que herramientas externas (notebooks, aplicaciones web, plataformas educativas) generen y validen código Modelica sin instalar el editor, recibiendo el grafo del modelo en JSON y devolviendo el código `.mo` o el resultado de la validación. Esta separación entre lógica de negocio e interfaz gráfica es además el paso previo necesario para el despliegue web descrito a continuación.

Despliegue web. Empaquetar el editor junto con `OpenModelica` en un contenedor eliminaría cualquier requisito de instalación local, haciendo la herramienta accesible desde un navegador. El paso arquitectónico previo es el ya descrito: separar la lógica de negocio de la interfaz Swing mediante la API REST, sobre el que se construiría después un frontend web con persistencia en base de datos para los modelos y los usuarios. Ello permitiría que pudieran colaborar, añadir y guardar modelos, trabajar, estudiar. Todo ello repercute principalmente sobre las bases de datos de usuarios, protección de datos, seguridad, etc.

7.4.7. Integración con herramientas externas

La Tabla 7.3 resume las vías de integración técnica con herramientas y estándares del ecosistema Modelica:

Tabla 7.3: Vías de integración técnica del editor con herramientas externas.

Estándar / Librería	Uso principal	Beneficio para el editor
FMI 2.0 / 3.0 (Modelica Association Project FMI, 2023) (Functional Mock-up Interface)	Intercambio de modelos entre simuladores	Integración con MATLAB, Python (FMPy) y Simulink sin instalar <code>OpenModelica</code>
MOSES (Fritzson et al., 2025) (<code>OpenModelica</code>)	Modelos socioeconómicos sostenibles	Componentes World3, Minsky y sistemas ecológico-financieros
JGraphT (Java)	Algoritmos de grafos y layout	Auto-layout Fruchterman-Reingold y Sugiyama para el canvas
JFreeChart / JavaFX Charts	Visualización de resultados	Gráficas embebidas de simulación dentro del editor
Spring Boot	Framework REST / web	API REST de exportación y despliegue web multiusuario
JUnit 5 + Mockito	Testing automatizado	Suite de tests unitarios e integración para las clases críticas

7.5 Conclusiones

Este capítulo ha recorrido el conjunto completo de resultados, limitaciones y perspectivas del trabajo. Las conclusiones técnicas de la Sec. 7.3 confirman que los modelos IS-LM y Solow se pueden implementar en Modelica con rigor formal, exportar a código estándar y validar numéricamente. Las líneas de trabajo futuro de la Sec. 7.4 identifican las extensiones técnicas más directas: variantes de los modelos IS-LM y Solow (Mundell-Fleming, Mankiw-Romer-Weil), nuevos componentes Modelica para economía circular y modelo Doughnut, mejoras de la interfaz y la integración con OpenModelica, y la integración con el estándar FMI y la librería MOSES.

Cuatro conclusiones sintetizan el alcance de la contribución:

1. **Viabilidad técnica demostrada.** Modelica, concebido para sistemas físicos, expresa con la misma claridad y rigor las relaciones macroeconómicas keynesianas y neoclásicas. El editor convierte esa viabilidad en accesibilidad.
2. **Extensibilidad validada.** El componente `Solow_Circularidad` (añadido con solo cuatro ficheros Java modificados sin tocar el resto del sistema) demuestra que la arquitectura MVC con conectores tipados escala a nuevos dominios económicos sin coste arquitectónico.
3. **Orientación pedagógica real.** La integración con datos abiertos del Banco Mundial permite calibrar los modelos con datos históricos reales, convirtiendo la herramienta en un puente entre la teoría y la realidad empírica que cualquier estudiante puede explorar sin coste.
4. **Punto de partida.** El editor es una primera piedra técnica con una arquitectura preparada para crecer: añadir un nuevo tipo de componente económico requiere modificar solo cuatro ficheros Java sin tocar el resto del sistema. Las extensiones identificadas en la Sec. 7.4 constituyen una hoja de ruta concreta para proyectos de continuación.

Bibliografía

- AHO, A. V., LAM, M. S., SETHI, R., and ULLMAN, J. D. (2006). *Compilers: Principles, Techniques, and Tools*. Boston: Addison-Wesley, 2.^a edition.
- BLANCHARD, O. (2017). *Macroeconomics*. Upper Saddle River, NJ: Pearson, 7 edition.
- BRAY, T. (2017). “The JavaScript Object Notation (JSON) Data Interchange Format”. RFC 8259, Internet Engineering Task Force (IETF). URL <https://www.rfc-editor.org/rfc/rfc8259>, Accedido: 25 de noviembre de 2025.
- BUTCHER, J. C. (2016). *Numerical Methods for Ordinary Differential Equations*. Chichester: John Wiley & Sons, 3 edition.
- CASELLA, F., OTTER, M., PROELSS, K., RICHTER, C., and TUMMESCHEIT, H. (2005). The Modelica fluid and media library for modeling of incompressible and compressible thermo-fluid pipe networks. *Proceedings of the 4th International Modelica Conference*, pages 631–640.
- CEPAL (2026). “Economía circular en américa latina y el caribe”. URL <https://www.cepal.org/es/temas/economia-circular>, Accedido: 8 de febrero de 2026.
- DALY, H. E. (1996). *Beyond Growth: The Economics of Sustainable Development*. Boston: Beacon Press. URL <https://www.beacon.org/Beyond-Growth-P130.aspx>, Accedido: 2 de diciembre de 2025.
- Dassault Systèmes (2024). “Dymola — Multi-Engineering Modeling and Simulation”. URL <https://www.3ds.com/products/catia/dymola>, Accedido: 25 de noviembre de 2025.
- Departamento de Lenguajes y Sistemas Informáticos. UNED (2025). “Diseño de Aplicaciones Orientadas a Objetos — Guía de estudio (cód. 71022011)”. Universidad Nacional de Educación a Distancia (UNED), Grado en Ingeniería en Tecnologías de la Información. Curso académico 2025/2026.
- Doughnut Economics Action Lab (DEAL) (2026). “Recursos y herramientas para la transición hacia la economía de rosquilla”. URL <https://doughnuteconomics.org/>, Accedido: 6 de enero de 2026.

- Ellen MacArthur Foundation (2013). “Towards the circular economy: Economic and business rationale for an accelerated transition”. Technical report, Ellen MacArthur Foundation, Cowes. URL <https://www.ellenmacarthurfoundation.org/towards-the-circular-economy-vol-1-an-economic-and-business-rationale-for-an>, Accedido: 6 de enero de 2026.
- Ellen MacArthur Foundation (2026). “Visión general de la economía circular”. URL <https://www.ellenmacarthurfoundation.org/es/temas/presentacion-economia-circular/vision-general>, Accedido: 6 de enero de 2026.
- Escuela Técnica Superior de Ingeniería Informática. UNED (2022). *Programación Orientada a Objetos — Material didáctico*. Madrid: Universidad Nacional de Educación a Distancia (UNED).
- FELBER, C. (2010). *Gemeinwohl-Ökonomie: Das Wirtschaftsmodell der Zukunft*. Viena: Deuticke. URL <https://www.econgood.org/>, Trad. esp.: La economía del bien común. Barcelona: Deusto, 2012. Accedido: 18 de enero de 2026.
- FIELDING, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine.
- FRITZSON, P. (2014). *Introduction to Object-Oriented Modeling and Simulation with OpenModelica*. Linköping, Sweden: Liu e-press / Linköping University. URL <https://openmodelica.org/doc/OpenModelicaUsersGuide/>, Véase también: Fritzson, P. (2004). *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press. Accedido: 24 de septiembre de 2025.
- FRITZSON, P. et al. (2025). “MOSES — Modeling of Sustainable Economic Systems”. Consorcio de investigación, Universidad de Linköping / OpenModelica. URL <https://openmodelica.org/research/moses>, Accedido: 3 de junio de 2025.
- GAMMA, E., HELM, R., JOHNSON, R., and VLISSIDES, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley. URL <https://dl.acm.org/doi/book/10.5555/186897>, Accedido: 25 de noviembre de 2025.
- GitHub (Modelica-3rdparty) (2025). “BusinessSimulation — Modelica Library”. URL <https://github.com/Modelica-3rdparty/BusinessSimulation>, Accedido: 17 de septiembre de 2025.
- GRUNE, D. and JACOBS, C. J. H. (2008). *Parsing Techniques: A Practical Guide*. Nueva York: Springer, 2.^a edición. 1.^a ed. disponible libremente en https://dickgrune.com/Books/PTAPG_1st_Edition/.

- HAIRER, E., NØRSETT, S. P., and WANNER, G. (1993). *Solving Ordinary Differential Equations I: Nonstiff Problems*, volume 8 of *Springer Series in Computational Mathematics*. Berlin: Springer-Verlag, 2 edition. URL <https://link.springer.com/book/10.1007/978-3-540-78862-1>, Accedido: 2 de diciembre de 2025.
- HICKS, J. R. (1937). Mr. Keynes and the Classics: A Suggested Interpretation. *Econometrica*, 5 (2), 147–159.
- HUNTER, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9 (3), 90–95.
- IHS Markit (2024). “EViews 9.5 Overview: Presentation Quality Output”. IHS Global Inc. URL <https://www.eviews.com/EViews9/ev9overview.html>, Accedido: 3 de junio de 2025.
- JACKSON, T. (2009). *Prosperity Without Growth: Economics for a Finite Planet*. London: Earthscan. Ed. esp.: Prosperidad sin crecimiento. Barcelona: Icaria, 2011.
- JUnit Team (2024). “JUnit 5 User Guide”. <https://junit.org/junit5/>. Accedido: 3 de junio de 2025.
- KLUYVER, T., RAGAN-KELLEY, B., PÉREZ, F., GRANGER, B., BUSSONNIER, M., FREDERIC, J., KELLEY, K., HAMRICK, J., GROUT, J., CORLAY, S., IVANOV, P., AVILA, D., ABDALLA, S., and WILLING, C. (2016). Jupyter notebooks — a publishing format for reproducible computational workflows. In LOIZIDES, F. and SCHMIDT, B., editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90. IOS Press. Referencia estándar también para `ipywidgets`, parte del ecosistema Jupyter.
- KÖHLER, K. (2025a). “An IS-LM Model”. en *DIY Macroeconomic Model Simulation*. URL https://macrosimulation.org/an_is_lm_model, Accedido: 17 de septiembre de 2025.
- KÖHLER, K. (2025b). “DIY Macroeconomic Model Simulation”. URL <https://macrosimulation.org/>, Accedido: 17 de septiembre de 2025.
- KÖHLER, K. and PRANTE, F. (2025). “The Solow Growth Model”. en *DIY Macroeconomic Model Simulation*. URL https://macrosimulation.org/a_solow_model, Accedido: 17 de septiembre de 2025.
- MANKIW, N. G., ROMER, D., and WEIL, D. N. (1992). A contribution to the empirics of economic growth. *The Quarterly Journal of Economics*, 107 (2), 407–437.
- MARTIN, R. C. (2000). Design principles and design patterns. *ObjectMentor*.

- MEADOWS, D. H., MEADOWS, D. L., RANDERS, J., and BEHRENS, W. W. (1972). *The Limits to Growth*. New York: Universe Books. Report for the Club of Rome's Project on the Predicament of Mankind.
- Ministerio para la Transición Ecológica (MITECO) (2026). “Estrategia española de economía circular — españa circular 2030”. URL <https://www.miteco.gob.es/es/calidad-y-evaluacion-ambiental/temas/economia-circular/estrategia.html>, Accedido: 8 de febrero de 2026.
- Modelica Association (2025). “Modelica — A Unified Object-Oriented Language for Systems Modeling”. URL <https://modelica.org/>, Accedido: 17 de septiembre de 2025.
- Modelica Association (2026). “Librerías de modelica”. URL <https://modelica.org/libraries/>, Accedido: 25 de noviembre de 2025.
- Modelica Association Project FMI (2023). “Functional Mock-up Interface Specification 3.0”. <https://fmi-standard.org/docs/3.0/>. Accedido: 3 de junio de 2025.
- OpenModelica (2026). “OpenModelica — Open Source Modelica-based Modeling and Simulation Environment”. URL <https://openmodelica.org/>, Accedido: 25 de noviembre de 2025.
- RAWORTH, K. (2017). *Doughnut Economics: Seven Ways to Think Like a 21st-Century Economist*. White River Junction: Chelsea Green Publishing. URL <https://www.kateraworth.com/doughnut/>, Ed. esp.: Economía rosquilla. Barcelona: Paidós, 2018. ISBN 9788449334030. Accedido: 18 de enero de 2026.
- ROCKSTRÖM, J. et al. (2009). A safe operating space for humanity. *Nature*, 461, 472–475.
- SCHWABER, K. and SUTHERLAND, J. (2020). “The scrum guide™ — the definitive guide to scrum: The rules of the game”. URL <https://scrumguides.org/>, Accedido: 25 de noviembre de 2025.
- SOLOW, R. M. (1956). A contribution to the theory of economic growth. *The Quarterly Journal of Economics*, 70 (1), 65–94.
- URQUÍA MORALEDA, A. and MARTÍN VILLALBA, C. (2014). *Métodos de Simulación y Modelado*. Madrid: Editorial UNED. URL <https://www.librosuned.com/LU21280/>, Accedido: 9 de octubre de 2025.
- Wolfram (2026a). “The keynesian is-lm model”. URL <https://demonstrations.wolfram.com/TheKeynesianISLMModel/>, Accedido: 17 de septiembre de 2025.
- Wolfram (2026b). “Solow growth model”. URL <https://demonstrations.wolfram.com/SolowGrowthModel/>, Accedido: 17 de septiembre de 2025.

World Bank (2024). “World development indicators”. URL <https://data.worldbank.org/>, Accedido: 2 de diciembre de 2025.

World Bank (2026). “Data Help Desk: Developer Information and API Basic Call Structures”. World Bank Data Help Desk. URL <https://datahelpdesk.worldbank.org/knowledgebase/articles/898581-api-basic-call-structures>, Accedido: 19 de julio de 2026.

ZIMMER, D. and SCHLABE, D. (2012). Implementation of a modelica library for energy management based on economic models. In *Proceedings of the 9th International Modelica Conference*, pages 133–142, Munich, Alemania.

Apéndice A

Ejemplos de Código Generado

El Código A.1 muestra un ejemplo mínimo de código Modelica generado por el editor para un modelo IS-LM básico, con solo las instrucciones de instanciación y conexión (las definiciones de cada componente se omiten aquí por brevedad y se muestran completas en los apartados A.1 y A.2):

Código A.1: Ejemplo basico de modelo IS-LM generado por el editor.

```
1 model ISLM_Basico "Modelo IS-LM basico generado por el editor"
2   FuncionConsumo      consumo      (C0=100, c=0.8);    //  $C = C_0 + c*Y$ 
3   FuncionInversion    inversion    (I0=200, b=1000);    //  $I = I_0 - b*r$ 
4   GastoPublico        gobierno     (G=300);           // Gasto publico exogeno
5   MercadoBienes       mercado_bienes;                 // Mercado de bienes (IS)
6   DemandaDinero       demanda_dinero (k=0.2, h=1000);  //  $M_d = k*Y - h*r$ 
7   OfertaMonetaria     oferta_dinero (Ms=500);          // Oferta monetaria (banco
  ↪ central)
8   MercadoMonetario    mercado_monetario;              // Mercado monetario (LM)
9 equation
10  // Flujos hacia el mercado de bienes (IS)
11  connect(consumo.C_out,      mercado_bienes.C_in);
12  connect(inversion.I_out,    mercado_bienes.I_in);
13  connect(gobierno.G_out,     mercado_bienes.G_in);
14  // Retroalimentacion: Y determina C y Md
15  connect(mercado_bienes.Y_out, consumo.Y_in);
16  connect(mercado_bienes.Y_out, demanda_dinero.Y_in);
17  // Flujos hacia el mercado monetario (LM)
18  connect(demanda_dinero.Md_out, mercado_monetario.Md_in);
19  connect(oferta_dinero.Ms_out, mercado_monetario.Ms_in);
20  // Retroalimentacion: r determina I y Md
21  connect(mercado_monetario.r_out, inversion.r_in);
22  connect(mercado_monetario.r_out, demanda_dinero.r_in);
23 end ISLM_Basico;
```

A.1 Modelo IS-LM completo (incluyendo definiciones de componentes)

El Código A.2 muestra el modelo IS-LM completo tal y como lo genera el editor, integrando los Códigos 3.1 a 3.7 del Capítulo 3 (definiciones de cada componente y ensamblado del modelo principal) en un único archivo .mo listo para abrir directamente en OpenModelica. Para mayor claridad se ha excluido la documentación HTML embebida (Sec. 4.7.4): así, al simular, las variables económicas pueden inspeccionarse directamente en el panel de documentación de OpenModelica sin el texto adicional.

Código A.2: Modelo IS-LM completo generado por el editor.

```

1 package ISLM_
2   import Modelica.Blocks.Interfaces.*;
3   import Modelica.Blocks.Sources.*;
4   model FuncionConsumo
5     parameter Real C0 = 100;      // Consumo autónomo
6     parameter Real c = 0.8;      // Propensión marginal a consumir
7     RealInput Y_in; // Producción / ingreso (entrada)
8     RealOutput C_out; // Consumo agregado (salida)
9   equation
10    C_out = C0 + c * Y_in;      // Ec. (3.1)
11  end FuncionConsumo;
12  model FuncionInversion
13    parameter Real I0 = 300;      // Inversión autónoma
14    parameter Real b = 20;      // Sensibilidad I al tipo de interés
15    RealInput r_in; // Tipo de interés (entrada)
16    RealOutput I_out; // Inversión agregada (salida)
17  equation
18    I_out = I0 - b * r_in;      // Ec. (3.2)
19  end FuncionInversion;
20  model GastoPublico
21    parameter Real G = 200;      // Gasto público exógeno
22    RealOutput G_out; // Gasto público (salida)
23  equation
24    G_out = G;                  // Ec. (3.3)
25  end GastoPublico;
26  model MercadoBienes
27    parameter Real alpha = 1.0;  // Velocidad de ajuste
28    RealInput C_in; // Consumo (entrada)
29    RealInput I_in; // Inversión (entrada)

```

```

30   RealInput G_in; // Gasto público (entrada)
31   RealOutput Y_out(start = 1000, fixed = true); // Producción de
↪ equilibrio (salida)
32   Real DA;
33   equation
34     DA = C_in + I_in + G_in; // Ec. (3.5):  $DA = C + I + G$ 
35     der(Y_out) = alpha * (DA - Y_out); // Ec. (3.8)
36   end MercadoBienes;
37   model OfertaMonetaria
38     parameter Real Ms = 500; // Oferta monetaria (banco central)
39     RealOutput Ms_out; // Oferta monetaria (salida)
40     equation
41       Ms_out = Ms; // Oferta monetaria exogena
42     end OfertaMonetaria;
43     model DemandaDinero
44       parameter Real k = 0.2; // Sensibilidad Md al ingreso
45       parameter Real h = 10; // Sensibilidad Md al tipo de interés
46       RealInput Y_in; // Producción / ingreso (entrada)
47       RealInput r_in; // Tipo de interés (entrada)
48       RealOutput Md_out; // Demanda de dinero (salida)
49       equation
50         Md_out = k * Y_in - h * r_in; // Ec. (3.4)
51       end DemandaDinero;
52     model MercadoMonetario
53       parameter Real beta = 0.5; // Velocidad de ajuste
54       RealInput Md_in; // Demanda de dinero (entrada)
55       RealInput Ms_in; // Oferta monetaria (entrada)
56       RealOutput r_out(start = 0.05, fixed = true); // Tipo de interés de
↪ equilibrio (salida)
57     equation
58       der(r_out) = beta * (Md_in - Ms_in); // Ec. (3.10)
59     end MercadoMonetario;
60     model Main "Modelo IS-LM básico"
61       FuncionConsumo Consumo (C0 = 200, c = 0.8);
62       FuncionInversion Inversion (I0 = 300, b = 50);
63       GastoPublico Gasto (G = 200);
64       MercadoBienes IS (alpha = 1.0);
65       DemandaDinero DemandaDin (k = 0.2, h = 10);
66       OfertaMonetaria OfertaMon (Ms = 500);
67       MercadoMonetario LM (beta = 0.5);
68     equation

```

```

69 // Flujos hacia el mercado de bienes
70 connect(Consumo.C_out, IS.C_in);
71 connect(Inversion.I_out, IS.I_in);
72 connect(Gasto.G_out, IS.G_in);
73 // Retroalimentación: Y -> C, Md
74 connect(IS.Y_out, Consumo.Y_in);
75 connect(IS.Y_out, DemandaDin.Y_in);
76 // Flujos hacia el mercado de dinero
77 connect(DemandaDin.Md_out, LM.Md_in);
78 connect(OfertaMon.Ms_out, LM.Ms_in);
79 // Retroalimentación: r -> I, Md
80 connect(LM.r_out, Inversion.r_in);
81 connect(LM.r_out, DemandaDin.r_in);
82 end Main;
83 end ISLM_;

```

A.2 Modelo de Solow completo (incluyendo definiciones de componentes)

El Código A.3 muestra el modelo de Solow completo tal y como lo genera el editor, integrando los Códigos 3.8 a 3.12 del Capítulo 3 (definiciones de cada componente y ensamblado del modelo principal) en un único archivo .mo listo para abrir directamente en OpenModelica. Igual que en el caso anterior, se ha excluido la documentación HTML embebida para mantener el listado centrado en la lógica del modelo.

Código A.3: Modelo de Solow completo generado por el editor.

```

1 package Solow_Model
2   import Modelica.Blocks.Interfaces.*;
3   import Modelica.Blocks.Sources.*;
4   model Solow_Capital
5     parameter Real K0 = 100; // Capital inicial
6     parameter Real delta = 0.05; // Tasa de depreciación (delta = 5 %
  ↪ anual)
7     RealInput I_in; // Inversión (entrada)
8     RealOutput K_out(start = K0, fixed = true); // Stock de capital (salida
  ↪ )
9     equation
10      der(K_out) = I_in - delta * K_out; // Ec. (3.20): dK/dt = I - delta*K
11   end Solow_Capital;
12   model Solow_Produccion
13     parameter Real alpha = 0.33; // Elasticidad del producto respecto al
  ↪ capital

```

```

14   RealInput K_in; // Stock de capital (entrada)
15   RealInput L_in; // Fuerza laboral (entrada)
16   RealOutput Y_out; // Producción agregada (salida)
17   equation
18     Y_out = K_in^alpha * L_in^(1 - alpha); // Ec. (3.18):  $Y = K^{\alpha} * L^{\rightarrow \wedge(1-\alpha)}$ 
19   end Solow_Produccion;
20   model Solow_Poblacion
21     parameter Real n = 0.02; // Tasa de crecimiento poblacional (2 %
22     parameter Real L0 = 100; // Población inicial
23     RealInput n_tasa; // Tasa real (si disponible, p. ej.
24     RealOutput L_out(start = L0, fixed = true); // Fuerza laboral (salida)
25     equation
26       // Si hay datos reales ( $|n\_tasa| > 0.0001$ ), se usan; si no, el pará
27       der(L_out) = if abs(n_tasa) > 0.0001
28       then n_tasa * L_out // Ec. (3.22):  $dL/dt = n\_real * L$ 
29       else n * L_out; // Ec. (3.22):  $dL/dt = n * L$ 
30     end Solow_Poblacion;
31   model Solow_Inversion
32     parameter Real s = 0.25; // Tasa de ahorro por defecto (25 %)
33     RealInput Y_in; // Producción agregada (entrada)
34     RealInput s_tasa; // Tasa real (si disponible)
35     RealOutput I_out; // Inversión (salida)
36     equation
37       I_out = if s_tasa > 0.001
38       then s_tasa * Y_in // Ec. (3.23):  $I = s\_real * Y$ 
39       else s * Y_in; // Ec. (3.23):  $I = s * Y$ 
40     end Solow_Inversion;
41   model Main "Modelo de Solow básico"
42     Solow_Capital Capital (K0 = 100, delta = 0.05);
43     Solow_Produccion Produccion (alpha = 0.33);
44     Solow_Poblacion Poblacion (n = 0.02, L0 = 100);
45     Solow_Inversion Inversion (s = 0.25);
46     // Fuentes de tasas (valor 0 -> se usa el parámetro interno)
47     Constant tasa_s (k = 0);
48     Constant tasa_n (k = 0);
49     // Variables per capita (Ecs. 3.19, 3.28 -- 3.30)
50     Real k_percapita;

```

```

51   Real y_percapita;
52   Real i_percapita;
53   Real c_percapita;
54   equation
55     // Conexiones de tasas externas
56     connect(tasa_s.out, Inversion.s_tasa);    // s_real --> I
57     connect(tasa_n.out, Poblacion.n_tasa);    // n_real --> L
58     // Bucle de retroalimentación K -> Y -> I -> K
59     connect(Inversion.I_out, Capital.I_in);    // I --> K (I -> dK/dt)
60     connect(Capital.K_out, Produccion.K_in);  // K --> Y
61     connect(Poblacion.L_out, Produccion.L_in); // L --> Y
62     connect(Produccion.Y_out, Inversion.Y_in); // Y --> I (I = s*Y)
63     // Variables per cápita
64     k_percapita = Capital.K_out / Poblacion.L_out; // Ec. (3.28)
65     y_percapita = Produccion.Y_out / Poblacion.L_out; // Ec. (3.19)
66     i_percapita = Inversion.I_out / Poblacion.L_out; // Ec. (3.29)
67     c_percapita = (Produccion.Y_out - Inversion.I_out) / Poblacion.L_out;
68     ↪ // Ec. (3.30)
69   end Main;
end Solow_Model;

```

El código generado es directamente compatible con OpenModelica: puede abrirse, compilarse y simularse desde ese entorno sin ninguna modificación adicional.

Apéndice B

Guía Rápida de Uso

Esta guía describe los pasos esenciales para instalar, configurar y utilizar el editor gráfico de modelos económicos en Modelica.

B.1 Instalación

El editor puede ejecutarse de cuatro formas:

- Desde un fichero bat que inicia la compilación e inicio del programa: `EditorGrafico.bat`.
- **Desde código fuente:** con un compilador Java instalado, ejecuta `javac *.java` y luego `java EditorModelicaMultiDominio`.
- **Desde archivo JAR:** `java -jar EditorGrafico.jar`.
- **Desde un IDE** (Netbeans, Eclipse, IntelliJ): importa el proyecto e indica la clase principal `EditorModelicaMultiDominio`.

B.2 Configuración de OpenModelica (OMEdit)

El editor permite lanzar OMEdit automáticamente al exportar un modelo. Para configurar la ruta del ejecutable accede a **Herramientas** → **Configurar ruta de OMEdit** e introduce la ruta completa:

- **Windows:** `C:\Program Files\OpenModelica1.x-64bit\bin\OMEdit.exe`
- **Linux / macOS:** `/usr/bin/OMEdit` o `/opt/openmodelica/bin/OMEdit`

El diálogo de configuración también aparece automáticamente la primera vez que se arranca el editor, o cuando se detecta que la ruta almacenada ya no es válida (Figura B.1).

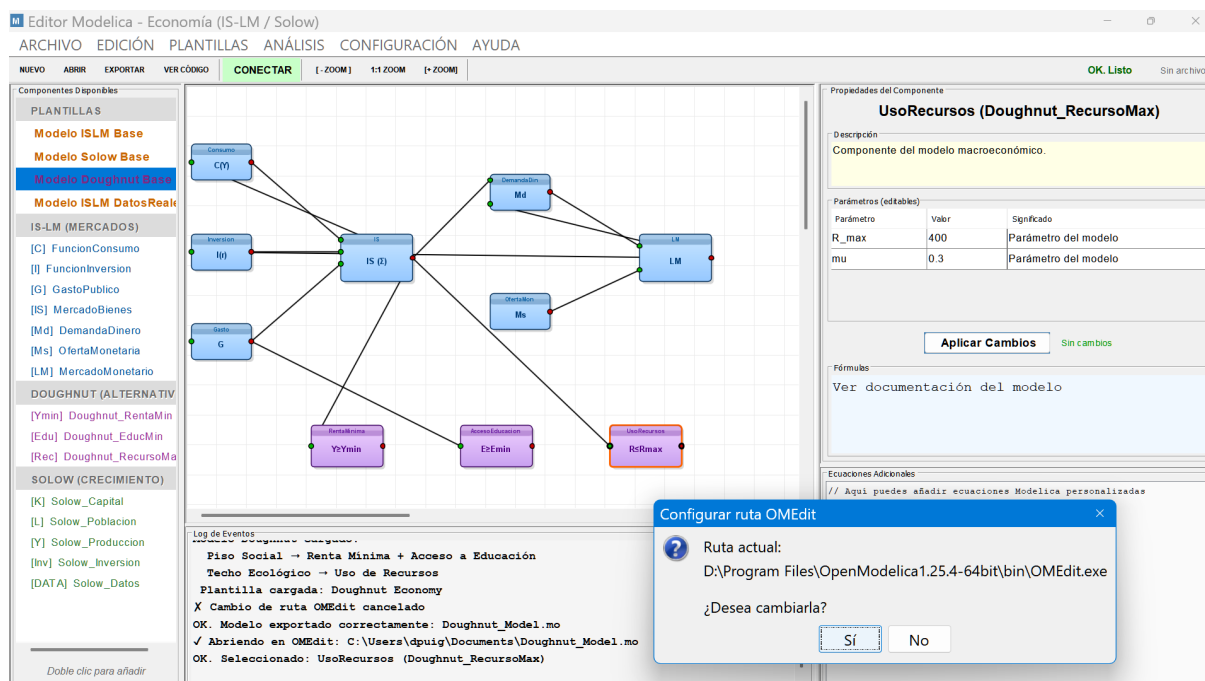


Figura B.1: Diálogo de configuración de la ruta de OMEdit.

B.3 Añadir componentes

La paleta izquierda muestra los componentes disponibles organizados por biblioteca. Haz **doble clic** en un componente para añadirlo al canvas, o carga una **plantilla completa** (Modelo IS-LM Base, Solow Base, Doughnut Base, Circular Base, o IS-LM con Datos Reales).

Nota sobre el grado de validación: de las dos extensiones del editor más allá de IS-LM y Solow estándar, únicamente la **Economía Circular** (Solow_Circularidad) ha sido **validada numéricamente** mediante compilación y simulación en OpenModelica, comparando los resultados frente a los valores analíticos esperados (Cap. 5, Sec. 5.10). El **modelo Doughnut** (Doughnut_RentaMin, Doughnut_EducMin, Doughnut_RecursoMax) está completamente **implementado y operativo** en el editor (se puede construir, conectar y exportar a código Modelica sin errores) como **versión reducida con tres indicadores**, frente a las más de veinte dimensiones (12 de suelo social, 9 de techo ecológico) del marco teórico completo de Raworth (2017); no ha sido sometido a esa misma validación analítica formal, por lo que se presenta como una **prueba de concepto** funcional, pendiente de validación numérica y de ampliación de cobertura en un trabajo futuro. La Figura B.2 muestra la paleta de componentes y plantillas disponibles.



Figura B.2: Paleta de componentes y plantillas disponibles.

Componentes IS-LM disponibles: *FuncionConsumo*, *FuncionInversion*, *GastoPublico*, *DemandaDinero*, *OfertaMonetaria*, *MercadoBienes* (IS) y *MercadoMonetario* (LM). A partir de estos siete componentes, el sistema de ecuaciones IS-LM calcula las dos variables de salida del modelo: el Producto Nacional (Y) y el Tipo de Interés (r).

Componentes Doughnut disponibles (conectados a IS-LM; prueba de concepto, no validados numéricamente): *Doughnut_RentaMin* (suelo social: renta mínima), *Doughnut_EducMin* (suelo social: gasto educativo mínimo) y *Doughnut_RecursoMax* (techo ecológico: uso máximo de recursos).

Componentes Solow disponibles: *Solow_Capital* (K), *Solow_Poblacion* (L), *Solow_Produccion* (Y), *Solow_Inversion* (Inv), *Solow_Datos* ($DATA$) y *Solow_Circularidad* ($K\rho$): variante validada de *Solow_Capital* con tasa de circularidad (Sec. 5.10).

B.4 Editar parámetros

Selecciona un componente en el canvas y edita sus valores en el **panel derecho de propiedades** o mediante el **menú contextual** (botón derecho del ratón → Editar parámetros).

Para componentes con datos históricos del Banco Mundial, el panel muestra un botón *Ver datos* que permite inspeccionar la serie temporal cargada.

La Figura B.3 muestra el panel de propiedades del componente seleccionado.

The screenshot shows a software interface for editing a component's properties. At the top, there are two buttons: 'OK. Listo' (green) and 'ISLM_Ghana.mo' (blue). Below them is a section titled 'Propiedades del Componente' with a sub-header 'Produccion (Solow_Produccion)'. This section contains three main areas: a description, a table of parameters, and a formulas section. The description states it's a Cobb-Douglas production function. The parameters table lists 'alpha' with a value of 0.33 and a description of capital elasticity. The formulas section shows the mathematical equation for the production function. At the bottom of this section are 'Aplicar Cambios' and 'Sin cambios' buttons. Below the main section is another area titled 'Ecuaciones Adicionales' with a text area for additional model equations.

OK. Listo ISLM_Ghana.mo

Propiedades del Componente

Produccion (Solow_Produccion)

Descripción

Función de producción Cobb-Douglas:
 $Y = K^\alpha \cdot L^{(1-\alpha)}$. Combina capital y trabajo para generar producto. α

Parámetros (editables)

Parámetro	Valor	Significado
alpha	0.33	Elasticidad del capital ...

Aplicar Cambios Sin cambios

Fórmulas

FUNCIÓN DE PRODUCCIÓN:
 $Y = K^\alpha \cdot L^{(1-\alpha)}$
 Cobb-Douglas con $\alpha \approx 0.33$

Ecuaciones Adicionales

// Aquí puedes añadir ecuaciones Modeli...

Figura B.3: Panel de propiedades del componente seleccionado.

B.5 Conectar componentes

Pasos para establecer una conexión entre dos componentes del modelo:

1. Pulsa el botón **CONECTAR** (barra superior, color rojo).
2. Haz clic en el **puerto de origen** del primer componente.
3. Haz clic en el **puerto de destino** del segundo componente.
4. El editor valida automáticamente la compatibilidad de la conexión.

El menú contextual (botón derecho) también permite conectar, desconectar y eliminar componentes desde el canvas.

La Figura B.4 muestra el menú contextual del canvas.

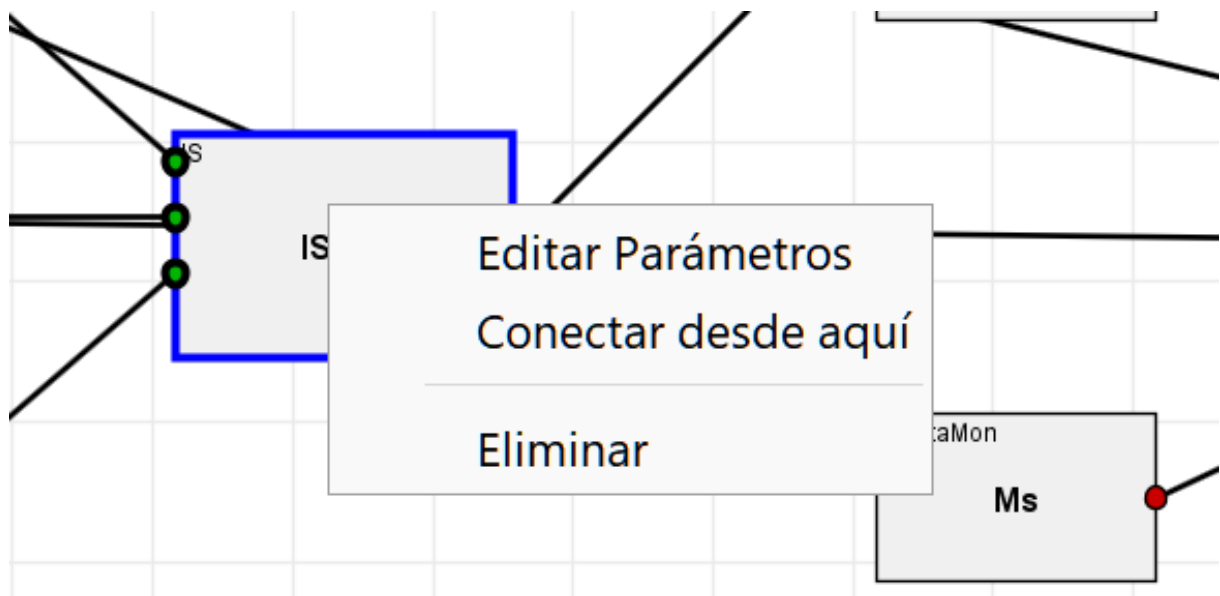


Figura B.4: Menú contextual del canvas (botón derecho del ratón).

B.6 Cargar datos históricos reales

El editor integra datos del **Banco Mundial** (World Development Indicators). Para usar datos reales en un modelo:

1. Carga la plantilla *Modelo IS-LM Datos Reales* o *Modelo Solow Base*.
2. En el panel de propiedades del componente `Solow_Datos`, pulsa **Descargar datos**.
3. Selecciona el **modelo** (IS-LM o Solow) y el **país**.

4. El componente descarga automáticamente las series históricas y parametriza el modelo.

La Figura B.5 muestra la carga del modelo IS-LM con datos reales del Banco Mundial.

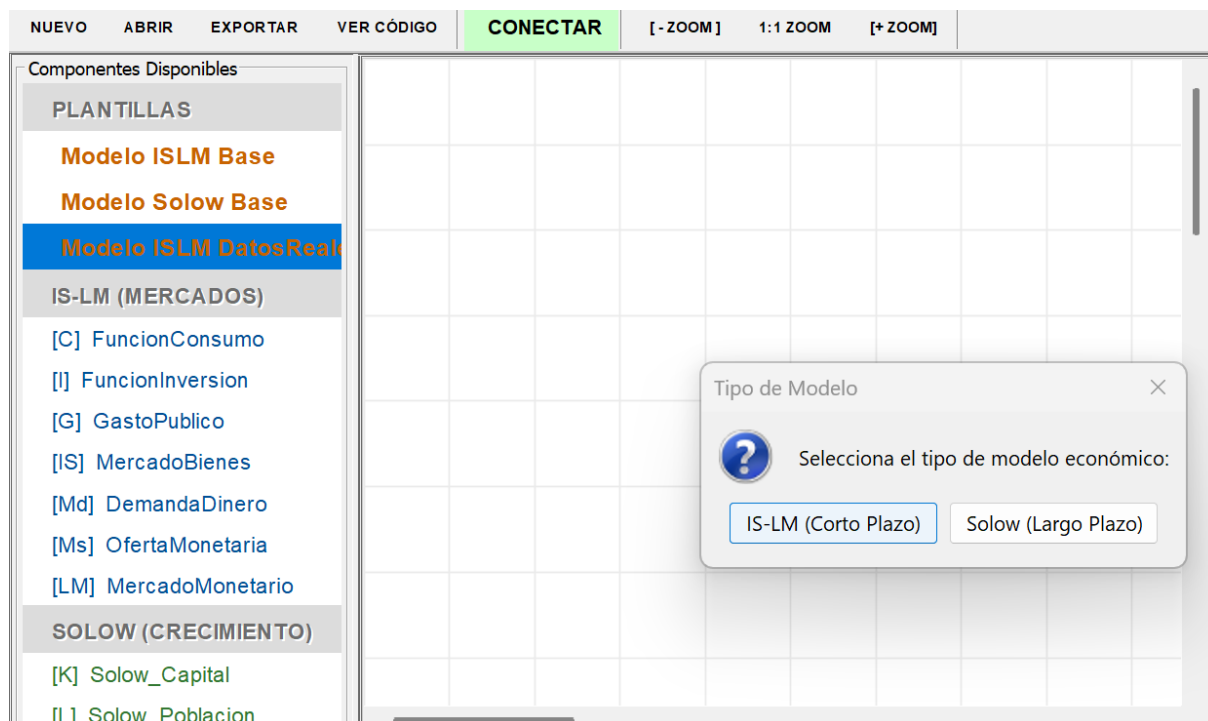


Figura B.5: Carga del modelo IS-LM con datos reales del Banco Mundial.

La carga completa se ilustra también en la Figura B.6.

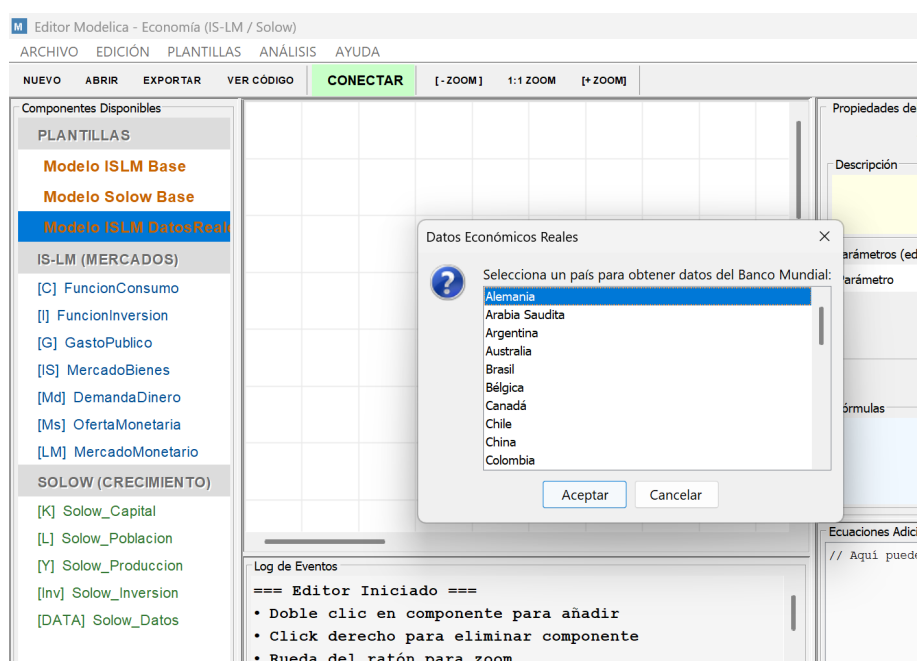


Figura B.6: Selector de país para la descarga de datos históricos.

B.7 Exportar y simular en OpenModelica

Procedimiento para exportar el modelo y lanzar la simulación:

1. Diseña el modelo completo en el canvas.
2. Selecciona **Archivo** → **Exportar a .mo**.
3. El editor genera el archivo `.mo` y, si la ruta de OMEdit está configurada, lo abre automáticamente.
4. En OpenModelica ejecuta la simulación con los parámetros deseados.

B.8 Zoom y navegación

Acciones disponibles para desplazarse y ajustar la vista del lienzo:

- **Rueda del ratón:** zoom in/out en el canvas.
- **Botones de la barra superior:** `[-ZOOM]`, `[1:1 ZOOM]`, `[+ZOOM]`.

Con estos pasos el usuario dispone de todo lo necesario para instalar, construir, validar y exportar modelos económicos con el editor.