Editor de modelos eléctricos en lenguaje Modelica

Trabajo Fin de Master

Master en Ingeniería Informática

Curso 2022-2023

Autor:

Joaquín Sánchez Higueras

Directores:

Alfonso Urquía Moraleda

Carla Martín Villalba

Escuela Técnica Superior de Ingeniería Informática

Universidad Nacional de Educación a Distancia (UNED)

Sánchez Higueras, Joaquín (2023). *Editor de modelos eléctricos en lenguaje Modelica*. Trabajo de Fin de Master: Universidad Nacional de Educación a Distancia.

RESUMEN

Cuando estudiamos un sistema físico, recurrimos a expresarlo mediante un modelo matemático. La simulación sobre este modelo nos permite extraer resultados o conclusiones en función de los parámetros que gobiernen el sistema analizado. El modelado y la simulación juegan un papel muy importante en el desarrollo de los procesos de ingeniería, facilitando el diseño y la evaluación de diferentes escenarios.

Modelica es un lenguaje de modelado de sistemas complejos y dinámicos, basado en ecuaciones y orientado a objetos. Está concebido para describir modelos compuestos por ecuaciones algebraico diferenciales y eventos, y es un referente en el modelado de sistemas del mundo físico, que pueden ayudarnos a analizar cómo se comportan componentes eléctricos, térmicos, hidráulicos o mecánicos entre otros.

Este trabajo consiste en el desarrollo de una herramienta, la cual proporcionará una traducción de un modelo gráfico diseñado por el usuario a lenguaje Modelica. La aplicación proporcionará una codificación del modelo que posibilitará la simulación para analizar y predecir el comportamiento de un sistema, identificando problemas o puntos de optimización en el diseño del modelo además de facilitar la comprensión de estudiantes y profesionales. El diseño del modelo gráfico y la interfaz de usuario se desarrollarán mediante tecnologías de programación web modernas.

En nuestra aplicación usaremos el dominio eléctrico, ofreciendo una serie de componentes analógicos, distribuidos sobre una paleta, que podrán seleccionarse e interrelacionarse para construir un modelo completo mediante una interfaz gráfica con objetos conectables. Los componentes podrán ser parametrizados modificando sus propiedades, además, este modelo tendrá que superar una serie de comprobaciones para verificar su corrección. Una vez comprobadas estas reglas, el modelo podrá describirse como un modelo matemático o modelo atómico, o bien, como un modelo de objetos o modelo compuesto en lenguaje Modelica, generando todo el código necesario con el que podremos trabajar desde cualquier editor de Modelica realizando las simulaciones pertinentes.

Hemos de distinguir entre Modelica, el lenguaje de modelado en sí, y los entornos de trabajo donde llevaremos a cabo las simulaciones. Entre estos entornos destacaremos OpenModelica, herramienta de código abierto, muy extendida, capaz de asignar la causalidad computacional y la ordenación del modelo, esto permitirá la traducción de Modelica a un sistema de ecuaciones.

La solución implementada abarca el uso de la plataforma .NET persistiendo los modelos y la información necesaria en bases de datos SQL server. Usaremos ASP.NET y JavaScript como lenguajes en el diseño de la interfaz de usuario y dotaremos de una capa de abstracción en C# para la lógica de traducción de un modelo gráfico a lenguaje Modelica. La aplicación se ha desarrollado usando metodologías ágiles, realizando un desarrollo iterativo incremental.

Los resultados obtenidos nos permiten constatar cómo funciona cada componente dentro del sistema, además de dar una visión de cómo trabajan los entornos en Modelica al traducir modelos compuestos a modelos atómicos.

Palabras claves

Lenguaje Modelica, Aplicaciones Web, Modelado de sistemas físicos, Dominio eléctrico, Editor de modelos, Lenguaje de modelado, Tecnologías .NET.

Sánchez Higueras, Joaquín (2023). *Electric models editor in Modelica language*. . Dissertation final.: National University of Distance Education.

ABSTRACT

When we study a physical system, we resort to expressing it through a mathematical model. The simulation on this model allows us to draw results or conclusions based on the parameters that govern the analyzed system. Modeling and simulation play a very important role in the development of engineering processes, facilitating the design and evaluation of different scenarios.

Modelica is an object-oriented, equation-based modeling language for complex and dynamic systems. It is designed to describe models composed of differential algebraic equations and events, and is a benchmark in modeling systems of the physical world, which can help us analyze how electrical, thermal, hydraulic, or mechanical components behave, among others.

This work consists in the development of a tool, which obtained a translation of a graphic model designed by the user into Modelica language. The application will allow a coding of the model that will allow the simulation to analyze and predict the behavior of a system, identifying problems or points of optimization in the design of the model in addition to facilitating the understanding of students and professionals. The design of the graphic model and the user interface will be developed using modern web programming technologies.

In our application we will use the electrical domain, offering a series of analog components, distributed on a palette, which can be selected and interrelated to build a complete model through a graphical interface with connectable objects. The components can be parameterized by modifying their properties, in addition, this model will have to pass a series of validations to verify its correctness. Once validated, the model can be translated into a mathematical model or flat model, or into an object model or composite model in Modelica language, generating all the necessary code with which we can work from any Modelica editor performing the relevant simulations.

We must distinguish between Modelica, the modeling language itself, and the work environments where we will carry out the simulations. Among these environments we will highlight OpenModelica, a widely used open-source tool, capable of assigning computational

causality and ordering of the model, this will allow the translation of Modelica into a system of equations.

The implemented solution covers the use of the .NET platform, persisting the models and the necessary information in SQL server databases, making use of ASP.NET and JavaScript as languages in the design of the user interface and providing an abstraction layer. in C# for the translation logic of a graphical model to Modelica language. The application has been developed using agile methodologies, performing an incremental iterative development.

The results obtained allow us to verify how each component works within the system, in addition to giving a vision of how the environments in Modelica work when translating compound models into flat models.

Keywords:

Modelica Language, Web Applications, Physical Systems Modeling, Electrical Domain, Model Editor, Modeling Language, .NET technologies.

Índice

1 Introducción, objetivos y estructura 15

1.1 Introducción 15

1.2 Objetivos 16

1.3 Estructura 19

2 Revisión del estado del arte 20

2.1 Introducción 20

2.2 Clasificación de los modelos 20

2.3 Modelado y simulación de tiempo continuo 22

2.4 Entornos de modelado en Modelica 29

2.4.1 OpenModelica 32

2.4.2 Dymola 34

2.5 Herramientas de transcripción de diagramas en código 35

2.5.1 GoJS 36

2.5.2 mxGraph 36

2.6 Arquitectura de desarrollo 37

2.6.1 Patrón MVC 38

2.7 Entorno de desarrollo 39

2.7.1 Visual Studio 39

2.7.2 SQL Server Management Studio 41

2.8 Lenguajes de programación 41

2.8.1 C# y Asp.NET 41

2.8.2 HTML, JavaScript y JQuery 42

2.9 Conclusiones 42

3 Análisis y planificación del proyecto 44

3.1 Introducción 44

3.2 Catálogo de requisitos 44

3.3 Análisis y especificaciones 46
3.4 Planificación 48
3.5 Conclusiones 51
4 Definición y traducción de los modelos 52
4.1 Introducción 52
4.2 Modelo XML mxGraph 54
4.3 Modelo de clases interno 57
4.4 Traducción a modelo atómico en Modelica 58
4.4.1 Declarar los voltajes y corrientes 59
4.4.2 Declarar los parámetros 59
4.4.3 Describir las ecuaciones en los nodos 60
4.4.4 Describir las relaciones constitutivas 63
4.5 Generación del modelo compuesto en Modelica 64
4.6 Comprobaciones sobre el modelo 68
4.7 Conclusiones 74
5 Construcción del frontend 75
5.1 Introducción 75
5.2 Lienzo y propiedades del entorno 75
5.3 Definición de paleta de herramientas 76
5.4 Propiedades de los elementos 76
5.5 Barra de herramientas 78
5.6 Editor de código 79
5.7 Accesibilidad y usabilidad 80
5.8 Conclusiones 81
6 Fase de pruebas en OpenModelica 82
6.1 Introducción 82
6.2 Modelo de Test 1 82

6.3 Modelo de Test 2 88
6.4 Modelo de Test 3 93
6.5 Modelo de Test 4 98
6.6 Modelo de Test 5 103
6.7 Modelo de Test 6 109
6.8 Modelo de Test 7 113
6.9 Validación de requisitos 120
6.10 Conclusiones 121
7 Conclusiones y trabajos futuros 122
7.1 Introducción 122
7.2 Conclusiones 122
7.3 Trabajos futuros 123
Bibliografía 125
Glosario 128
Anexo A: Manual de usuario 130
A-1 Instalación 130
A-2 Gestión de usuarios 131
A-3 Gestión de proyectos 132
A-4 Interfaz de edición de circuitos 133
A-5 Diagnosis de errores 135
A-6 Generación de código 136
Anexo B: Código fuente 138
B-1 ModelicaController.cs 138
B-2 FlatService.cs 152
B-3 CompositeService.cs 166

Índice de figuras

Figura 1-1 Vista general del editor de modelos 17
Figura 1-2 Vista general del generador de código 18
Figura 2-1 Clasificación de los modelos matemáticos (Urquía & Villalba, 2016) 22
Figura 2-2 Cronología de los distintos hitos de la simulación (PEC 2020 de la asignatura Métodos de modelado y simulación) 23
Figura 2-3 Modelo de trayectoria de un cohete realizado en CSSL (Huntsinger, 1988) 24
Figura 2-4 Modelo mecánico desarrollado en SIMULINK, (Alonso, 2012) 26
Figura 2-5 Ejemplo de código de circuito eléctrico realizado en Modelica 28
Figura 2-6 Ejemplo de simulación de sistema mecánico en SimulationX 29
Figura 2-7 Traducción de objetos en OpenModelica a modelo compuesto 31
Figura 2-8 Definición del objeto "Resistor" 31
Figura 2-9 Interfaz de OpenModelica 33
Figura 2-10 Entorno de modelado de Dymola 34
Figura 2-11 Entorno de simulación de Dymola 35
Figura 2-12 Ejemplo de un diagrama BPMN en GoJS 36
Figura 2-13 Modelo con servidor de aplicaciones 38
Figura 2-14 Modelo MVC 38
Figura 2-15 Visual Studio -Creación de un proyecto MVC nuevo 40
Figura 2-16 Visual Studio - Estructura de un nuevo proyecto MVC 40
Figura 2-17 SQL Server Management Studio 41
Figura 3-1 Modelo de tablas generado por Identity 46
Figura 3-2 Diagrama conceptual de proyectos y circuitos 47
Figura 3-3 Casos de uso de la aplicación 48
Figura 4-1 Esquema de modelos y transformaciones entre ellos usados en la aplicación 52
Figura 4-2 Propiedades con valores iniciales de cada componente 54
Figura 4-3 Modelo simple conectando 3 elementos 56
Figura 4-4 Modelo de clases interno 57
Figura 4-5 Instancia de la clase ModelItem simulando un generador sinusoidal 57
Figura 4-6 Configuración del atributo fixed 59

Figura 4-7 Ejemplo de la primera Ley de Kirchhoff 61
Figura 4-8 Nodo con nombre “a” conectando un generador con un inductor 61
Figura 4-9 Circuito modelado con varios elementos en paralelo 62
Figura 4-10 Iconos de tipos de representación en OpenModelica 68
Figura 4-11 Representación gráfica de un modelo en OpenModelica 68
Figura 4-12 Comprobación 01 69
Figura 4-13 Comprobación 04 70
Figura 4-14 Mensajes de comprobación de OpenModelica al escribir dos resistencias con el mismo nombre 71
Figura 4-15 Panel de propiedades donde se asigna el nombre a cada elemento 72
Figura 4-16 Comprobación 06, donde vemos como falta una conexión en el inductor L 72
Figura 4-17 Comprobación 08, fuentes de corriente en serie 73
Figura 4-18 Fuentes de tensión en paralelo 74
Figura 5-1 Aspecto general del editor dentro de la aplicación. 75
Figura 5-2 Paleta de componentes del editor 76
Figura 5-3 Ventana de propiedades de componente 77
Figura 5-4 Barra de funciones 78
Figura 5-5 Vista del editor de código 79
Figura 6-1 Circuito de Test 1 82
Figura 6-2 Corriente en la resistencia R3 83
Figura 6-3 Voltaje en el condensador de los modelos atómico y compuesto del Test1 83
Figura 6-4 Variación del modelo de test1 87
Figura 6-5 Modelo de Test 2 88
Figura 6-6 Valores de la simulación sobre el voltaje en R1 en Test 2 91
Figura 6-7 Valores sobre la simulación en el voltaje de R2 y R3 del Test 2 92
Figura 6-8 Valores de corriente en R1 y R2 sobre la simulación del Test 2 92
Figura 6-9 Modelo de Test 3 y su equivalente Thévenin reducido 93
Figura 6-10 Propiedades de la simulación de los modelos para Test 3 94
Figura 6-11 Model del Test 4 98
Figura 6-12 Cambios de estado en el interruptor 98
Figura 6-13 Valor del voltaje en el condensador C2 99

Figura 6-14 Valor del voltaje en el condensador C1 99
Figura 6-15 Diagrama del modelo en OpenModelica 100
Figura 6-16 Model de Test 5 103
Figura 6-17 Voltaje en el nodo u1 105
Figura 6-18 Voltaje en los nodos u3 y u4 106
Figura 6-19 Modelo de Test 6 109
Figura 6-20 Voltaje de distintos nodos con el modelo atómico y compuesto superpuestos 113
Figura 6-21 Modelo de test 7 113
Figura 6-22 Voltaje en el condensador C2 119
Figura 6-23 Voltaje en el nodo u5 119
Figura A-1 Página de bienvenida 130
Figura A-2 Página de registro de usuario nuevo 131
Figura A-3 Página de login 132
Figura A-4 Página de visualización de proyectos 132
Figura A-5 Página de creación de proyectos 133
Figura A-6 Página de visualización de circuitos 133
Figura A-7 Editor de circuitos 134
Figura A-8 Propiedades de elementos del modelo 135
Figura A-9 Código atómico generado a partir del modelo en Modelica 136
Figura A-10 Código en formato compuesto del modelo generado en Modelica 137

Índice de tablas

Tabla 2-1 Tipos de modelos 20

Tabla 2-2 Comparación de la simulación analógica con el modelado mediante diagramas de bloques 25

Tabla 2-3 Herramientas de OpenModelica (OpenModelica, 2023) 33

Tabla 3-1 Requisitos funcionales 45

Tabla 3-2 Requisitos no funcionales 46

Tabla 3-3 Desglose de iteraciones y tareas del proyecto 49

Tabla 3-4 Diagrama de Gantt con planificación temporal 50

Tabla 4-1 Componentes eléctricos 53

Tabla 4-2 Mensajes de comprobación 69

Tabla 6-1 Verificación de requisitos funcionales 120

Tabla A-1 Comprobaciones sobre el modelo 135

1 Introducción, objetivos y estructura

1.1 Introducción

Este trabajo consiste en el desarrollo completo de una aplicación web que servirá para modelar sistemas eléctricos analógicos con cierta sencillez para un usuario y la capacidad de producir una descripción del sistema a código Modelica. La herramienta facilitará de una manera visual la comprensión de cómo trabaja un sistema del mundo real y como se definiría matemáticamente un circuito eléctrico.

Un editor gráfico de modelos Modelica es una herramienta utilizada para crear, modificar y analizar modelos matemáticos de sistemas dinámicos en el lenguaje Modelica. Los modelos creados en el editor pueden ser simulados y analizados para estudiar el comportamiento de los sistemas, lo que puede ser útil en una variedad de aplicaciones, incluyendo el diseño de sistemas mecánicos, eléctricos, termodinámicos y de control. Los editores gráficos de Modelica suelen incluir características como la capacidad de arrastrar y soltar componentes en el modelo, la generación automática de código y herramientas de análisis y visualización de resultados.

El editor es concebido como una prueba de concepto, dispondrá de las clases de componentes del dominio eléctrico más habituales y sencillos (resistencias, condensadores, diodos, fuentes de corriente, etc.). Con esto queremos reseñar que pretendemos comprobar la viabilidad técnica del concepto de traducir un modelo gráfico a un modelo matemático, no de realizar una aplicación que compita con los entornos de modelado y simulación actuales más extendidos.

Esta aplicación tiene dos características que destacan sobre las tradicionales, y es que las aplicaciones basadas en web usan menos recursos que programas instalados, además, no necesitan canales de distribución como el software tradicional lo que permite su accesibilidad a un público más amplio. Su carácter online la hace, además, una herramienta colaborativa en la que se pueden desarrollar modelos por varios usuarios en el mismo momento. Además, la mayoría de los entornos de simulación de Modelica nos mostrarán la transcripción de un modelo compuesto basado en clases y objetos, nuestra herramienta es capaz de descomponer un diagrama eléctrico hasta ofrecer un modelo atómico basado en las ecuaciones elementales del sistema.

1.2 Objetivos

El objetivo principal de este trabajo es desarrollar una aplicación que permite al usuario componer circuitos eléctricos, pinchando y arrastrando los componentes de una paleta de clases de componentes eléctricos predefinidos, y conectar dichos componentes gráficamente, generando automáticamente la descripción en lenguaje Modelica del circuito compuesto por el usuario. Para ello nos serviremos de una metodología de trabajo y de una serie de herramientas de programación que nos llevarán al producto final.

Estamos hablando de un desarrollo que abarca múltiples tecnologías, como ya veremos, y en el que se pueden explicar todas las etapas clásicas de cualquier metodología de desarrollo: análisis, diseño, desarrollo y pruebas.

Los funcionales desarrollados son importantes para dar sentido a la aplicación final, serán analizados en las posteriores secciones del trabajo pero a modo esquemático podríamos resumirlo en estos 3 puntos.

- Construir una aplicación software que permita a los usuarios, mediante un editor gráfico, crear modelos eléctricos descritos en el lenguaje de modelado Modelica.
- El editor debe disponer de una paleta de componentes predefinidos.
- El editor aplicará reglas para determinar si el modelo está correctamente definido y generará la descripción en lenguaje Modelica tanto para un modelo atómico como compuesto. El modelo compuesto se definirá instanciando las correspondientes clases de la Librería Estándar de Modelica versión 4.

Los usuarios podrán registrarse e identificarse en la aplicación mediante un formulario en el que se especificará un email y contraseña. Un usuario identificado podrá crear y gestionar carpetas de proyectos donde almacenar sus modelos. Cada vez que un usuario cree o edite un modelo, lo realizará desde una pantalla de edición especialmente pensada para trabajar con componentes conectables.

Como se muestra en la Figura 1-1, la vista general del editor de modelos, contaremos con un lienzo donde podremos añadir componentes desde la paleta de elementos eléctricos que se muestra a la izquierda de la pantalla. En la misma figura podemos observar cómo debemos disponer de una barra de herramientas para interactuar con el modelo mediante varias opciones, como pueden ser opciones de zoom, de carga o guardado de otros modelos y opciones para generar el código en Modelica.

Entrando más en detalle, el editor gráfico será construido con un framework de desarrollo de JavaScript que nos permitirá la conexión de elementos gráficos. El framework proporcionará

una transcripción en XML de esta colección de elementos y conexiones. Este modelo junto con otros datos de la interfaz como la gestión de usuarios o el conjunto de trabajos de cada usuario serán guardados directamente en una base de datos de tipo relacional.

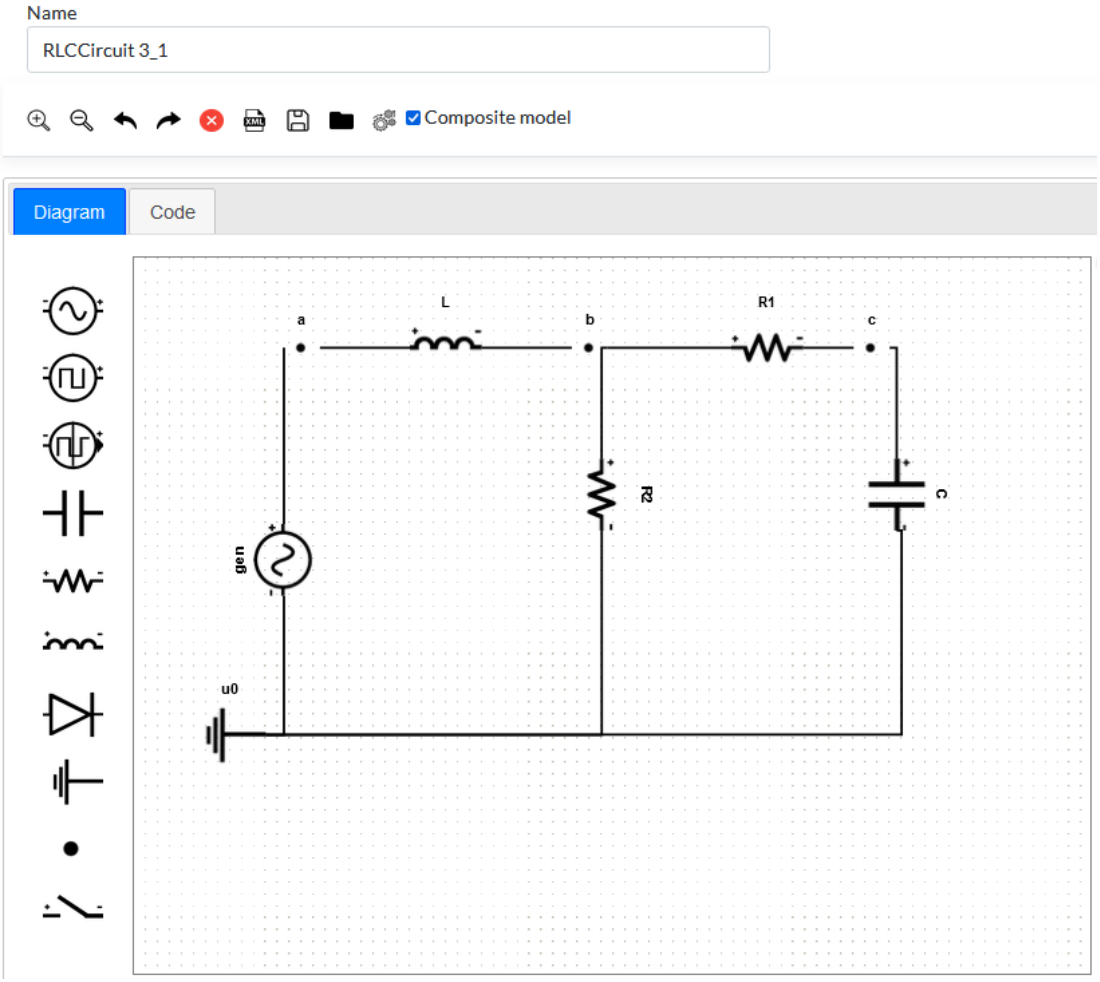


Figura 1-1 Vista general del editor de modelos

Antes del proceso que transcriba el modelo a lenguaje Modelica se deben tener en cuenta varios factores. El primero es que el modelo debe ser correcto y para ello debe pasar una serie de comprobaciones teniendo en cuenta las características del modelo eléctrico en su conjunto y de cada componente por separado. El modelo debe ser coherente y las conexiones deben estar hechas correctamente.

El segundo factor es que habrá que parsear el modelo XML de la interfaz gráfica y trabajar con él en un lenguaje que nos permita definir cada componente como un objeto con atributos y métodos, el candidato para este lenguaje es C#.

Por último, es necesario, crear un algoritmo que teniendo en cuenta las relaciones de cada componente con el resto, realice la descripción en lenguaje Modelica. Para un modelo atómico, es necesario desglosar las ecuaciones constitutivas de cada componente y las ecuaciones en los nodos,

es decir las ecuaciones que relacionan un componente con otro teniendo en cuenta el voltaje y la corriente. Para definir el modelo compuesto, será necesario asociar cada componente con los objetos de la librería standard de Modelica o MSL y establecer las conexiones entre ellos.

En la Figura 1-2 podemos observar como el código generado se debe mostrar en un editor de código en una pestaña anexa al editor de modelos.

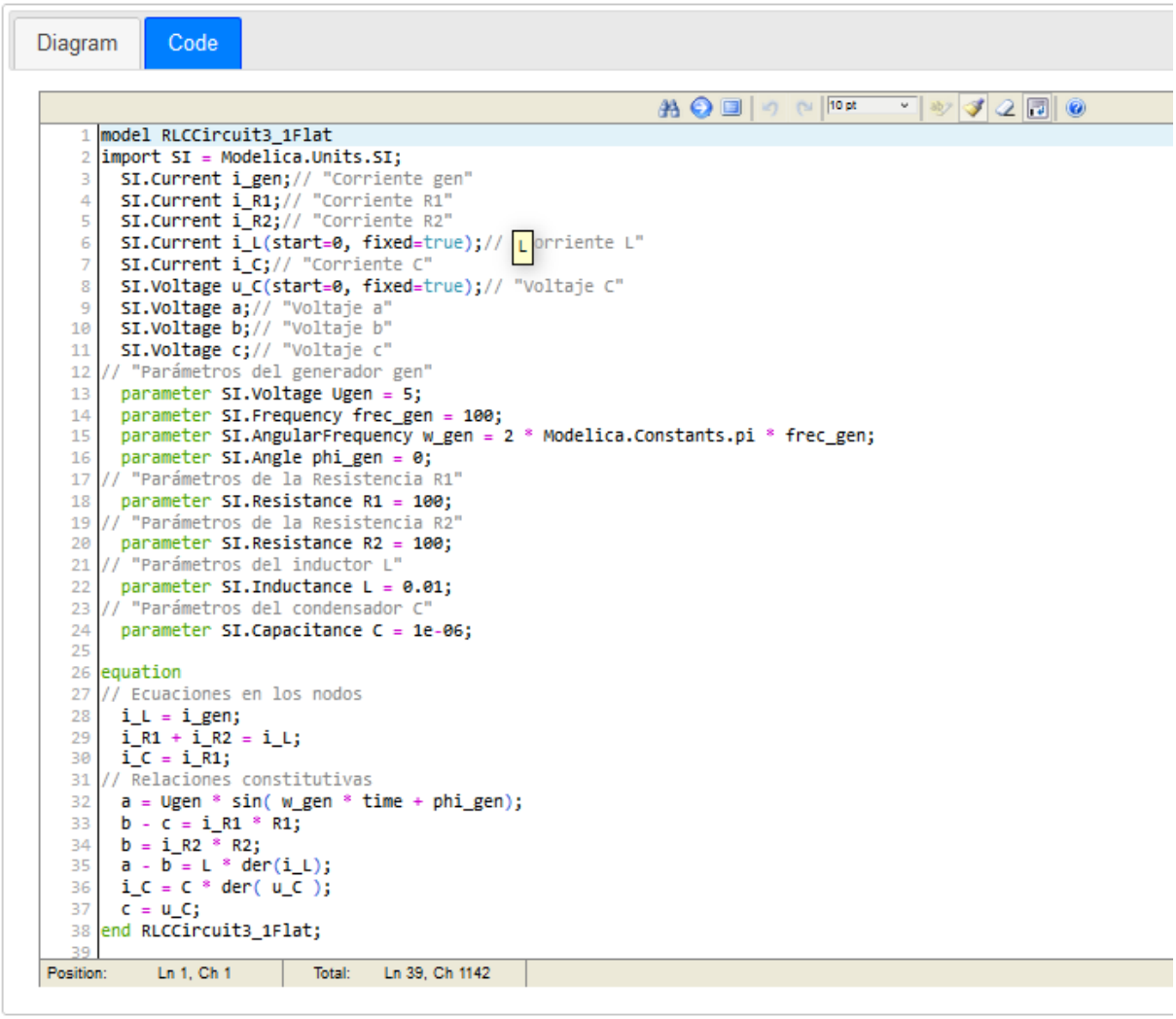


Figura 1-2 Vista general del generador de código

Finalmente el código generado por nuestra aplicación podrá ser utilizado para realizar simulaciones en otros entornos como OpenModelica o Dymola. Estos entornos son capaces de interpretar el código Modelica y traducirlo a un sistema de ecuaciones, resolviendo numéricamente el problema y aportando información mediante gráficas del comportamiento de los distintos componentes.

1.3 Estructura

El presente documento se estructura de la siguiente manera:

El *segundo capítulo* contiene los precedentes tecnológicos que existen en el mundo del modelado y la simulación de tiempo continuo, la evolución de la técnica, los lenguajes y los entornos tradicionalmente usados. En segundo lugar definiremos los elementos de nuestra aplicación, las distintas tecnologías que se usarán y daremos una visión completa del problema a resolver.

El *tercer capítulo* se centrará en la etapa de análisis y planificación, desglosando los requisitos y analizando las distintas tareas a realizar para el desarrollo.

El *cuarto capítulo* se centrará en los distintos modelos de datos que se manejan en las distintas capas de la aplicación, tanto modelos lógicos como físicos que son persistidos a nivel de una base de datos. A la vez explicaremos como estos modelos se utilizan para realizar los procesos de "backend" de traducción de un modelo a otro. Los algoritmos para generar el modelo atómico y compuesto serán explicados en este capítulo.

En el *quinto capítulo* queremos profundizar en la parte "frontend" de la aplicación. Explicaremos de una manera más detallada como el usuario va a interactuar con la aplicación y los distintos componentes de la interfaz gráfica.

En el *sexto capítulo* veremos cómo lo explicado en los capítulos anteriores, modelos, frontend y procesos de backend encajarán como un engranaje y producirá el código que podrá usarse para simulación en un editor de Modelica. Veremos varios casos de pruebas y verificaremos como el código generado para las simulaciones, corresponde con la resolución numérica de los modelos analizados.

El *séptimo capitulo*, hará una reflexión sobre todo el proceso software así como del estado final de la aplicación.

Finalmente incluiremos dos anexos. El *Anexo A* desarrollará un manual de la aplicación para el usuario. En el *Anexo B* incluiremos el código fuente de la aplicación que hemos considerado más relevante.

2 Revisión del estado del arte

2.1 Introducción

En este capítulo, en primer lugar, vamos a realizar una revisión de los lenguajes y herramientas de simulación y modelado y su evolución histórica. En segundo lugar, realizaremos un análisis de los componentes que forman nuestra aplicación, para determinar la interacción y dependencia entre ellos. Expondremos soluciones de arquitectura alternativas al problema y el por qué se han descartado o elegido unas u otras.

2.2 Clasificación de los modelos

El avance de la tecnología y el software ha hecho posible que los sistemas sean construidos reduciendo la necesidad de experimentos que encarecen su diseño y fabricación, además de dar soporte desde su concepción hasta su mantenimiento.

Para construir estos sistemas se hacen necesarios los modelos. Los modelos son prototipos que sirven de referencia para el diseño de un producto. Podemos decir que son representaciones de la realidad que explican un fenómeno y tienen la finalidad de explicar cómo funciona un sistema. Según Fritzson (2011) "Un modelo de un sistema es cualquier cosa a la que se puede aplicar un experimento, con el fin de responder a preguntas respecto del sistema".

Podríamos definir una clasificación de los tipos de modelos. En la Tabla 2-1 podemos ver un resumen comparativo de ellos.

Modelo mental	Modelo físico	Modelo formal
Subjetivo	Modelo tangible	Objetivo
Se obtiene de la observación y de inferir conclusiones sobre lo estudiado	Reconstrucción física del sistema	A partir de un modelo mental, y datos empíricos obtenemos un modelo matemático La base matemática puede ser ecuaciones

Tabla 2-1 Tipos de modelos

Los modelos pueden ser **mentales**, cuando construimos una aproximación de una idea o concepto en nuestro cerebro. Según Kenneth Craik (2007) un modelo mental es "Un mecanismo a escala del pensamiento mediante el cual un ser humano, intenta explicar cómo funciona el mundo

real. Es un tipo de representación de la realidad externa." Este concepto da lugar siempre a un modelo subjetivo ya que se forma a partir de juicios de valor y el conocimiento empírico del sujeto.

En contraposición tenemos los **modelos físicos**, que son tangibles y que nos dan una reconstrucción física del sistema. Según Fritzson (2011), "se trata de un objeto físico que reproduce algunas propiedades de un sistema real, para ayudarnos a responder preguntas del sistema", en este caso estamos hablando de prototipos o maquetas.

Por último podemos definir los tipos de modelos que se aplican en el caso de estudio, los **modelos matemáticos** o **modelos formales**, que definen mediante ecuaciones, algoritmos y parámetros de entrada y de salida. Según Morten Blomhoj (2004) "un modelo matemático es la relación existente entre ciertos objetos matemáticos y sus conexiones, por un lado, y por el otro, una situación o fenómeno de naturaleza no matemática". Los modelos matemáticos pueden ser muy simples o complejos, dependiendo de la naturaleza del sistema que se esté modelando y del propósito del modelo.

Los modelos matemáticos a su vez se pueden clasificar en varios tipos. Para empezar podemos clasificarlos según la variable tiempo, aquellos modelos en los que interviene el tiempo los denominaremos **modelos dinámicos**, el resto, **modelos estáticos**.

Según el instante de tiempo en el que cambian sus variables podemos clasificar los modelos dinámicos en 4 tipos adicionales.

- Modelos de tiempo discreto: las variables cambian en instantes predefinidas de tiempo en intervalos regulares. El resto del tiempo estas variables permanecen constantes.
- Modelos de eventos discretos: igual que los modelos de tiempo discreto, pero en este caso, los intervalos no son regulares ni predefinidos, si no que responden a un evento.
- Modelos de tiempo continuo: el valor de las variables cambia de manera continua a lo largo de todo el tiempo.
- Modelos híbridos: son aquellos que tienen una parte de tiempo continuo y otra de tiempo o eventos discretos.

En el caso de este trabajo, vamos a centrarnos en los **modelos de tiempo continuo y en los modelos híbridos**. En los modelos de tiempo continuo, nos centraremos concretamente, en los modelos de parámetros concentrados, los cuales la única derivada en cualquier ecuación es respecto al tiempo. En contraposición están los modelos de parámetros distribuidos, donde además de derivadas respecto al tiempo encontramos respecto a las coordenadas espaciales. La Figura 2-

1 es un ejemplo de la clasificación de los modelos matemáticos que acabamos de describir. En el siguiente apartado entraremos más en detalle con los distintos tipos de modelos de parámetros concentrados.

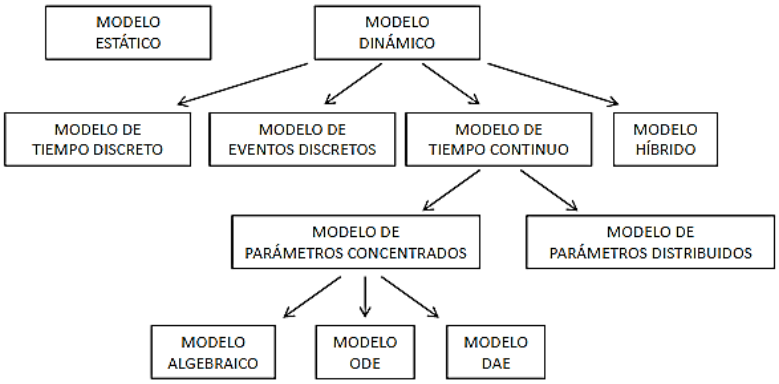


Figura 2-1 Clasificación de los modelos matemáticos (Urquía & Martín, 2016)

2.3 Modelado y simulación de tiempo continuo

En esta sección vamos a realizar una revisión de la historia de la simulación de tiempo continuo basándonos en Åström et al. (1998). Esta revisión es necesaria para entender el contexto en el que se desarrolla nuestra aplicación y conocer cómo han evolucionado estas disciplinas a lo largo del tiempo.

Las primeras herramientas de modelado y simulación fueron analógicas. Permitían a los ingenieros predecir el comportamiento de sistemas físicos complejos. Por ejemplo, máquinas y sistemas de producción, modelando sistemas en términos de **ecuaciones diferenciales ordinarias** (ODE) y posteriormente construyendo un prototipo o dispositivo físico basado en estas ecuaciones.

Las ecuaciones ODE relacionan la función desconocida de una única variable independiente con sus derivadas. En contraste las **ecuaciones algebraico diferenciales** (DAE) es una forma general de ecuación diferencial en la que las derivadas no se expresan explícitamente. Las ecuaciones algebraico diferenciales combinan ecuaciones algebraicas con ecuaciones diferenciales ordinarias, con derivadas únicamente con respecto al tiempo.

La **simulación analógica** pretende definir un modelo mediante ecuaciones diferenciales ordinarias, descomponiendo el problema en partes (componentes) que llevan asociados ecuaciones, un flujo de entrada y otro de salida. Los flujos de entrada son los datos que se utilizan para inicializar y controlar la simulación, mientras que los flujos de salida son los resultados y las observaciones que se obtienen después de que la simulación ha sido ejecutada.

Con anterioridad a la década de los 1960, solo se podía experimentar en el campo de los modelos ODE, pero el avance tecnológico provocó la proliferación de herramientas de modelado y simulación. En la Figura 2-2 las **clasifica** en 3 grupos:

- Herramientas de propósito general
- Herramientas de propósito específico
- Librerías para lenguajes de programación

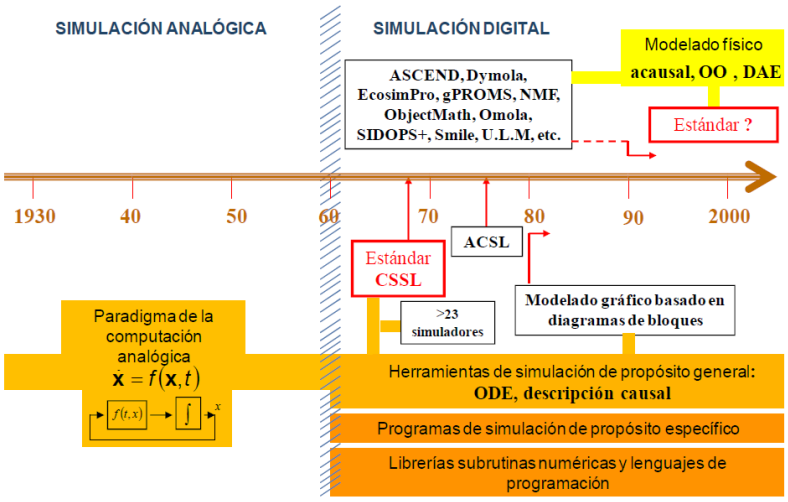


Figura 2-2 Cronología de los distintos hitos de la simulación (PEC 2020 de la asignatura Métodos de modelado y simulación)

La Figura 2-2 es una representación cronológica de todas las etapas en las que se ha desarrollado la simulación de tiempo continuo. Se destaca un **hito** importante en el cual se puede ver el cambio de paradigmas: el paso de la simulación analógica a la digital en los años 60, producido por la creación de los primeros ordenadores digitales. Este cambio dio lugar a diferentes lenguajes y tecnologías de modelado y simulación que 3 décadas después originarían el modelado físico.

Inicialmente, a principios de la década de 1960, fueron desarrollados una serie de lenguajes en ensamblador y en otros lenguajes no estandarizados lo que los hacía lenguajes no portables. Estas aplicaciones se desarrollaban en máquinas dedicadas y difícilmente podía ser compartido entre la comunidad científica.

El estándar CSSL (Continuous System Simulation Languages) fue un primer intento de unificar los conceptos para un lenguaje general, pero la citada expansión de distintas herramientas y tecnologías hace que hoy en día no haya un estándar como tal. CSSL fue diseñado para ayudar

a ingenieros y matemáticos a describir y resolver problemas del mundo físico. En la Figura 2-3 se observa la definición de la dinámica de un cohete.

```
Rocket Dynamics (CSSL-IV)
PROGRAM ROCKET
INITIAL
  CONSTANT  K=0.008,G=32.17, ...
  TOFF=60.0,TFIN=100.0,YØ=0.0,...
  DYØ=0.0,THRUST=7000.0
END $"OF INITIAL SECTION"
DYNAMIC
   CINTERVAL  CINT=1.0
DRIVATIVE  ROCK
  ALGORITHM  IALGO=5, ALGO=5
  SWITCH=TOFF-T
  W=SWIN(SWITCH,3000.0-400*TOFF,3000.0-40.0*T)
  THRUST=SWIN(SWITCH,0.0,THRUST)
  DRAG=K*DY*ABS(DY)
  D2Y=G*(THRUST-DRAG)/W-G
  DY=INTEG(D2Y,DYØ)
   Y=INTEG(DY,YØ)
END $ "OF DERIVATIVE SECTION"
   TERMT(T .GE. TFIN)
   PREPAR Y,DY,D2Y,W,THRUST
 END $ "OF DYNAMIC"
TERMINAL
END
END $ "OF PROGRAM IN CSSL-IV"
```

Figura 2-3 Modelo de trayectoria de un cohete realizado en CSSL (Huntsinger, 1988)

En 1967 apareció ACSL (Advance Continuos Simulation Language) como lenguaje derivado de CSSL, lenguaje basado en sentencias que a partir de un fichero fuente con extensión .csl, se traducía a FORTRAN y posteriormente se compilaba generando un ejecutable.

ACSL validaba el fichero antes de pasarlo a FORTRAN, buscando errores de estructura, luego era capaz de ordenar las ecuaciones, pero no las manipulaba simbólicamente, por lo cual había que escribir las ecuaciones con la causalidad computacional explícita. ACSL sólo resolvía ecuaciones implícitas con una única incógnita. Es decir, no permitía resolver lazos algebraicos compuestos por varias ecuaciones e incógnitas. Según Urquía y Villalba (2016) la **causalidad computacional** es una propiedad global del modelo, y se refiere a la relación que debe emplearse para calcular cada incógnita dentro del conjunto de ecuaciones. La conexión entre los componentes de un modelo y las condiciones de contorno condicionan esta causalidad computacional.

Por otro lado, el **modelado basado en diagramas de bloques,** que se inició en la década de 1980, aborda el problema de la misma manera pero gráficamente. Mediante distintas librerías emula el comportamiento de componentes a los cuales se les asocia una serie de ecuaciones que definen su comportamiento. Esta metodología puede resultar compleja para elementos con numerosas ecuaciones dinámicas, por ello, muchas herramientas desarrollan bloques que

incorporan todos los elementos de una sola vez. En la Tabla 2-2 podemos ver un resumen comparativo de estos dos paradigmas.

Simulación analógica	Modelado mediante diagramas de bloques
• Modelar un sistema en términos de ecuaciones diferenciales ordinarias ODE • Hacer que un dispositivo físico obedezca estas ecuaciones • Dispositivo inicializado con valores iniciales • La ecuación diferencial se presenta en términos de operaciones básicas (sumas, multiplicaciones y funciones generadoras)	• Se presentan los modelos descompuestos por componentes con entradas y salidas • Usado ampliamente en la industria (terminales, simuladores aéreos, etc.) • Flujo de datos unidireccional

Tabla 2-2 Comparación de la simulación analógica con el modelado mediante diagramas de bloques

El **modelado físico** es producto de la descomposición del problema en subsistemas. Cada subsistema tendrá una magnitud asociada (voltaje, masa, momento, etc.) y unas interfaces y se puede descomponer en niveles más básicos. No confundir el paradigma del modelado físico con el tipo de modelo físico explicado en la clasificación de los modelos en la sección anterior, este segundo se refiere a los modelos tangibles que podemos construir como maquetas. El modelado físico nos conduce a la posibilidad de definir modelos DAE con ecuaciones algebraico diferenciales. La simulación analógica tenía limitaciones con respecto al modelado físico, mientras que ambos trabajan descomponiendo el problema, solo el modelado físico es capaz de resolver modelos matemáticos DAE, la simulación analógica solo trabaja con ecuaciones diferenciales ordinarias ODE.

Una vez introducido el modelado físico, podemos hablar de los distintos lenguajes de modelado y simulación de tiempo continuo como Modelica, entre otros.

Dentro de estos lenguajes de modelado podríamos clasificarlos en dos grupos, los lenguajes orientados a bloques y los orientados a sentencias, la mayoría de las aplicaciones hoy en día son capaces de combinar los dos aspectos dando la capacidad de jugar con una interfaz gráfica y a la vez de codificarla para lograr el mismo objetivo.

Un ejemplo de lenguaje orientado a bloques, SIMULINK (que originalmente fue llamado SIMULAB) y se integra con MATLAB. Apareció en 1991 (Grace, 1991). Está especialmente diseñado para trabajar con diagramas de bloques usando MATLAB para el análisis dinámico del sistema.

Los diagramas de bloques de SIMULINK pueden ser definidos como una ecuación de estado, es decir, cada bloque tiene un conjunto específico de propiedades y parámetros que se pueden ajustar para adaptar el comportamiento del bloque al sistema en cuestión. Un bloque puede ser una función, un sumador, un integrador, una señal, un sensor, etc. Es ampliamente usado en Ingeniería por ejemplo para trabajar con procesamiento de señales.

En SIMULINK, los usuarios crean modelos arrastrando y soltando bloques que representan diferentes componentes o subsistemas en un editor gráfico. En la Figura 2-4 podemos ver como se modela un sistema mecánico que simula la amortiguación de un vehículo. Vemos como el origen es un bloque de tipo escalón y los componentes muelle y amortiguador modifican la velocidad y la posición. En SIMULINK el usuario es el encargado de definir el comportamiento y las ecuaciones de cada bloque. SIMULINK no tenía la capacidad de ordenar el modelo y asignar la causalidad computacional, esto lo hacía directamente el usuario estudiando en profundidad el modelo desarrollado para preparar su simulación.

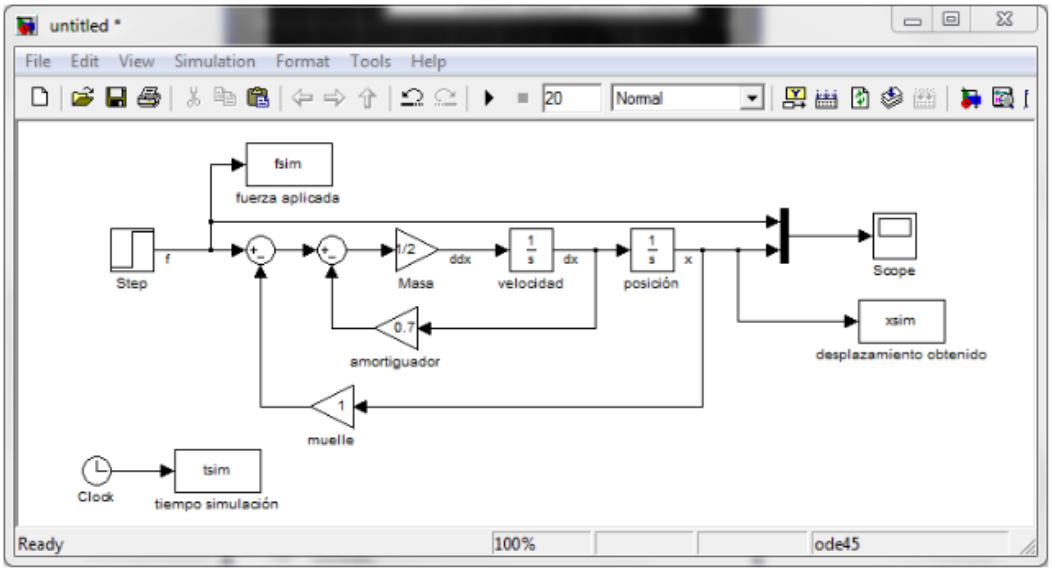


Figura 2-4 Modelo mecánico desarrollado en SIMULINK, (Alonso, 2012)

Modelica es un lenguaje de modelado declarativo, orientado a objetos y basado en componentes que modelan el mundo físico. Se basa en la idea de que los sistemas físicos se pueden representar mediante componentes individuales que interactúan entre sí. Cada componente se modela mediante un conjunto de ecuaciones matemáticas que describen su comportamiento. Los componentes se pueden combinar para formar sistemas más complejos, y las interacciones entre los componentes se describen mediante conectores. Apareció en 1997 y su objetivo fue estandarizar un formato para reutilizar sistemas dinámicos. Modelica se basa en un principio de funcionamiento claro, en el cual se descompone un sistema en distintos elementos, se define la interacción entre cada elemento y se define el comportamiento en función de ecuaciones

matemáticas. En contraposición con SIMULINK, los entornos de simulación de Modelica son no causales, es decir, si son capaces de asignar la causalidad computacional al modelo sin que el usuario establezca un orden en sus ecuaciones.

Actualmente Modelica se encuentra en su versión 3.5 y tiene una librería de 1400 modelos y 1200 funciones genéricos a lo largo de varios dominios. La librería más destacable es la Modelica Standard Library (MSL), soportada por una entidad sin ánimo de lucro, la Modelica Association.

Desde su lanzamiento en 1997, ha habido varias versiones del lenguaje Modelica que han añadido diferentes funcionalidades y mejoras.

La versión 1.0, lanzada en 1997, incluía un lenguaje de modelado basado en ecuaciones, una librería estándar de componentes y una interfaz de usuario.

La versión 2.0, lanzada en 2000, añadió características como el soporte para modelos paralelos, la capacidad de importar y exportar modelos a otros formatos y mejoras en el rendimiento de la simulación.

La versión 3.0, lanzada en 2006, incluyó una mayor flexibilidad en el diseño de componentes, la capacidad de crear jerarquías de componentes y mejoras en la librería estándar.

La versión 3.2, lanzada en 2010, introdujo mejoras en la estabilidad y en la velocidad de la simulación, así como una mayor compatibilidad con otros programas de simulación.

La versión 3.3, lanzada en 2017, incluyó una mayor facilidad para crear componentes de forma automatizada, una mayor compatibilidad con sistemas de control y mejoras en la librería estándar.

La versión 3.4, lanzada en 2020, incluyó la capacidad de modelar sistemas continuos y discretos juntos, mejoras en la compatibilidad con otros lenguajes y una librería estándar más completa.

En general, la evolución del lenguaje Modelica se ha centrado en mejorar la facilidad de uso, la flexibilidad en el diseño de componentes y la compatibilidad con otros programas de simulación.

El código de Modelica se estructura de una forma muy simple, como puede observarse en la Figura 2-5. Inicialmente se definen las variables que componen el modelo, para la definición de estas variables se pueden usar tipos de datos de la MSL que representan las distintas magnitudes del mundo físico. A continuación se definen los parámetros numéricos que gobiernan el modelo. Tanto variables como parámetros se pueden declarar en cualquier orden. En una segunda parte del código Modelica se definen las ecuaciones que interrelacionan cada elemento, por un lado las

ecuaciones que definen cada elemento, y por otro, la conexión entre ellos o también llamadas ecuaciones constitutivas, no es necesario declararla en un orden específico.

```
model Test1Flat
import SI = Modelica.Units.SI;
  SI.Current i_gen;// "Corriente gen"
  SI.Current i_R;// "Corriente R"
  SI.Current i_L(start=0, fixed=true);// "Corriente L"
  SI.Current i_C;
  SI.Voltage u_C(start=0, fixed=true);// "Voltaje C"
  SI.Voltage a;// "Voltaje a"
  SI.Voltage b;// "Voltaje b"
// "Parámetros del generador gen"
  parameter SI.Voltage Ugen = 5;
  parameter SI.Frequency frec_gen = 100;
  parameter SI.AngularFrequency w_gen = 2 * Modelica.Constants.pi * frec_gen;
  parameter SI.Angle phi_gen = 0;
// "Parámetros de la Resistencia R"
  parameter SI.Resistance R = 100;
// "Parámetros del inductor L"
  parameter SI.Inductance L = 0.1;
// "Parámetros del condensador C"
  parameter SI.Capacitance C = 1e-06;
equation
// Ecuaciones en los nodos
  i_L = i_gen;
  i_C = i_L;
  i_R = i_C;
// Relaciones constitutivas
  a = Ugen * sin( w_gen * time + phi_gen);
  n = i_R * R;
  a - b = L * der(i_L);
  i_C = C * der( u_C );
  b - n = u_C;
end Test1_aFlat;
```

Figura 2-5 Ejemplo de código de circuito eléctrico realizado en Modelica

El código de Modelica se puede agrupar en funciones y clases siguiendo los patrones clásicos de la programación orientada a objetos. De esta manera se puede definir un elemento del modelo como una clase con sus parámetros y comportamiento intrínseco, luego se pueden usar estos elementos conectándolos en un sistema para formar un modelo completo.

Otra de las características de Modelica es que permite trabajar con eventos. Un evento es un cambio repentino en el estado de un sistema que ocurre en un instante específico. Los eventos pueden ser causados por un cambio en una variable de entrada o por un cambio en una variable interna del sistema, se utilizan para modelar fenómenos como el contacto entre dos objetos, el encendido de un interruptor, el cierre de una válvula, etc. En Modelica, los eventos se manejan

mediante el uso de ecuaciones discretas y un algoritmo de eventos que controla el orden en el que se resuelven las ecuaciones.

2.4 Entornos de modelado en Modelica

En el sitio web de Modelica Association (Modelica Association, 2023) podemos encontrar un catálogo extenso de herramientas, la mayoría relacionadas con modelado multidominio de sistemas complejos basados en leyes del mundo físico, entre ellas hay muchas características comunes. Todas ellas coinciden en el tipo de licencia privativa a excepción de OpenModelica, la mayoría posee integraciones con otros programas extendidos como pueden ser MATLAB, o aplicaciones de CAD, CFD o incluso entornos de bases de datos. Algunas ofrecen características de modelado en 3 dimensiones como por ejemplo SimulationX o MWorks, En la Figura 2-6 podemos ver el entorno de simulación de SimulationX, se aprecia como un modelo construido en base a un diagrama de bloques también puede tener una representación en tres dimensiones.

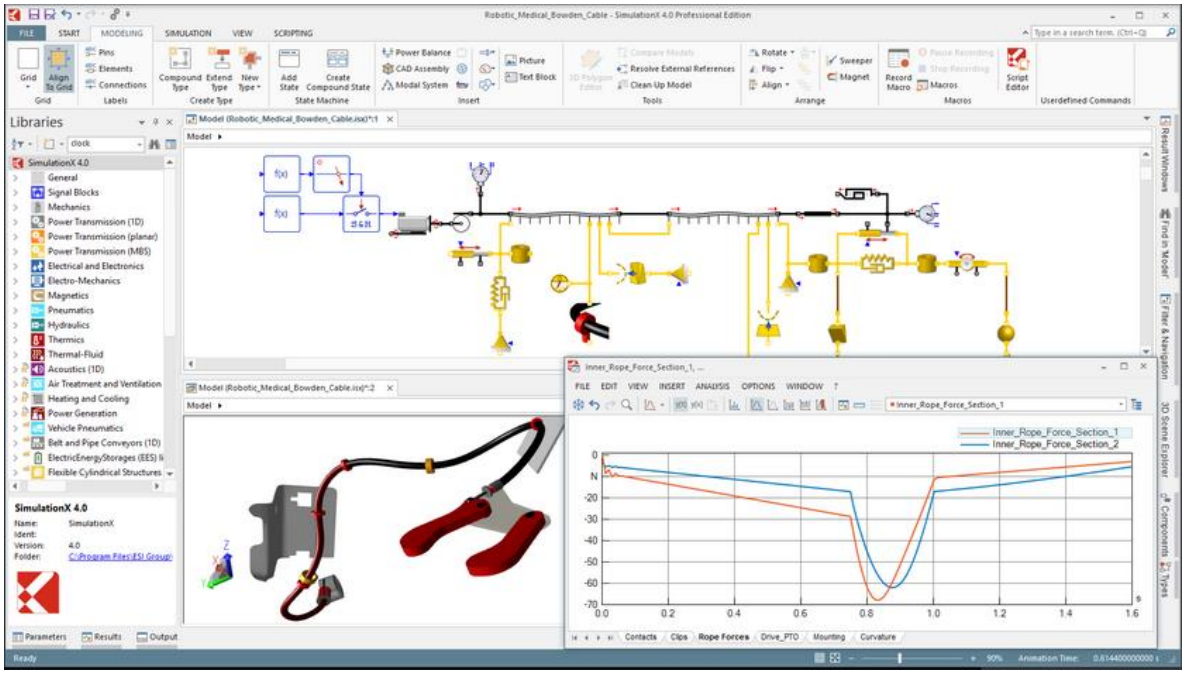


Figura 2-6 Ejemplo de simulación de sistema mecánico en SimulationX

Es muy habitual en estas herramientas que se trabaje con distintas librerías creadas para dominios concretos, por ejemplo SimulationX usa distintos paquetes según la característica del sistema a modelar, en el caso de trabajar con sistemas mecánicos, o lógica de fluidos o leyes de la termodinámica, además de permitir cargar cualquier librería en Modelica que cumpla las especificaciones de la versión de Modelica que se está usando en ese momento. Este es el principio de la mayoría de las aplicaciones de modelado y simulación. Modelica define paquetes que pueden

ser importados en cada proyecto, usando objetos predefinidos y clasificados según el dominio en el que se trabaja, esto favorece sobre todo la modularidad y la encapsulación.

Si quisiéramos detallar las características mínimas que necesita un entorno de simulación para simular un modelo descrito en Modelica podríamos incluir las siguientes características:

- Análisis de la causalidad computacional: determina la secuencia correcta de evaluación de las ecuaciones del modelo.
- Reducción del índice del DAE: para resolver sistemas de ecuaciones diferenciales algebraicas de manera eficiente.
- Resolución de sistemas de ecuaciones no lineales: para manejar modelos que contengan ecuaciones no lineales.
- Detección de eventos: manejo de los cambios discretos en el sistema.
- Integración de DAE: para resolver los sistemas de ecuaciones diferenciales algebraicas usando un método de integración numérica adecuado.
- Interfaz gráfica de usuario: para facilitar la interacción del usuario y el diseño de los modelos.

Todas estas herramientas suelen trabajar con otros estándares de mercado aparte de Modelica. Por ejemplo el estándar JMAG fue desarrollado en 1983 como un software de simulación para el desarrollo y diseño de dispositivos eléctricos o MATLAB que es un sistema para resolución de problemas estadísticos y numéricos, también es extendido el uso de otras herramientas CAD como AutoCAD, SolidWorks, QCAD para la importación de modelos.

En este caso de estudio vamos a centrarnos en dos herramientas concretamente que luego usaremos en la fase de validación y test, estas son OpenModelica y Dymola. Ambos entornos pueden trabajar mediante una interfaz de código como con una interfaz gráfica.

Los objetos generados como clases en Modelica, describen su comportamiento mediante unos parámetros de entrada y ecuaciones. Normalmente, al trabajar sobre la interfaz gráfica estos objetos se conectan entre sí. Al cambiar el tipo de visualización del modelo a la interfaz de código se puede ver como se instancian estos objetos pero la lógica de ecuaciones subyacente resulta transparente y oculta al usuario, no obstante, si nos vamos a la definición original de cada componente si podemos ver su parametrización y las ecuaciones que lo componen.

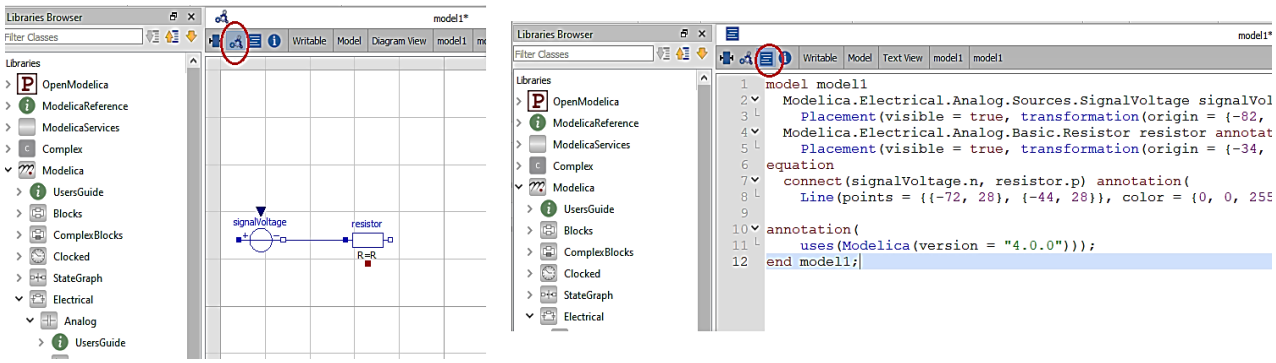


Figura 2-7 Traducción de objetos en OpenModelica a modelo compuesto

Podemos ver un ejemplo de lo descrito en la Figura 2-7. Creamos un modelo sencillo conectando una fuente a una resistencia, al cambiar al vista de visualización, vemos como se instancian las clases “SignalVoltage” y “Resistor”. Podemos observar cómo se conectan sus pines, pero no vemos las ecuaciones subyacentes. La Figura 2-8, muestra la definición del objeto “Resistor”, en este caso si podemos ver las ecuaciones en su definición.

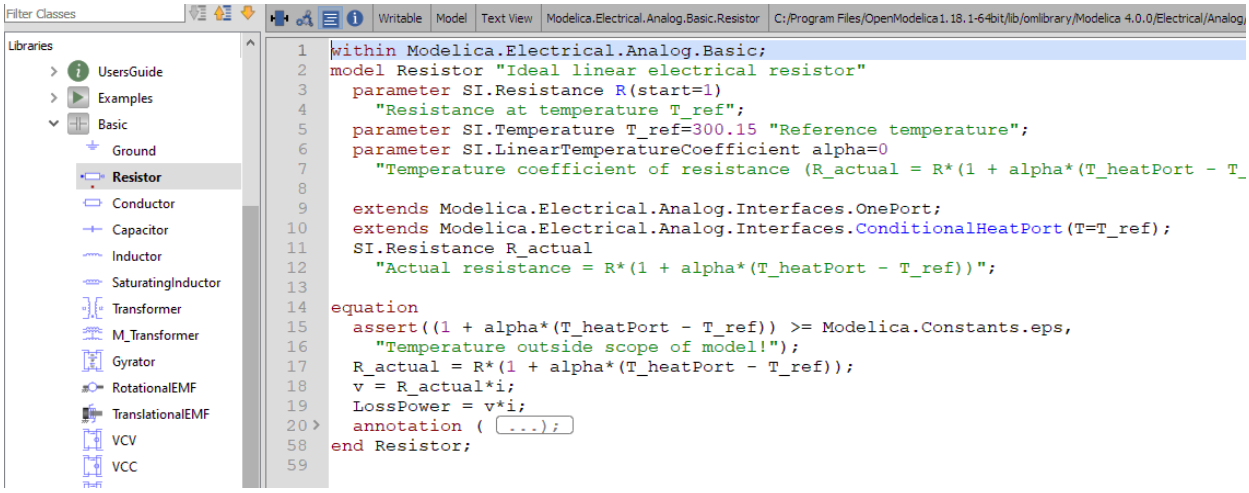


Figura 2-8 Definición del objeto "Resistor"

Estos entornos, realizan una conversión de estos objetos o de lo que llamamos modelo compuesto (Figura 2-7) a una descripción plana para realizar la compilación del modelo y preparar la simulación.

Un modelo plano en Modelica es una representación matemática de un sistema físico o proceso, donde las variables del sistema son descritas mediante ecuaciones, algoritmos y eventos. Estos modelos se utilizan para simular y analizar el comportamiento dinámico de sistemas físicos y se caracterizan por su simplicidad y rapidez de cálculo.

2.4.1 OpenModelica

EL *Open Source Modelica Consortium* o conocido por sus siglas OSMC desarrolló bajo una licencia GPL de software libre el entorno OpenModelica. Es multiplataforma lo que permite trabajar tanto con distribuciones de Linux como Windows. Proporciona una amplia gama de capacidades, incluyendo:

- Modelado de sistemas dinámicos mediante el lenguaje de modelado Modelica.
- Simulación numérica de modelos mediante el uso de diferentes algoritmos de integración.
- Análisis de resultados mediante herramientas de visualización de datos.
- Integración con otros herramientas de ingeniería, como Matlab y Scilab.
- Optimización de modelos mediante técnicas de optimización no lineal y algoritmos genéticos.
- Diseño automatizado mediante herramientas de control automático.
- Interoperabilidad mediante la exportación de modelos a diferentes formatos, como FMU y C code.

El entorno se compone de una interfaz gráfica mediante la cual se pueden modelar sistemas complejos realizando operaciones de “drag and drop”, o traducido del inglés, “arrastrar y soltar” con componentes de la librería estándar de Modelica o con componentes personalizados. OpenModelica proporciona una variedad de capacidades para la edición gráfica de modelos, como son: Editor gráfico integrado, posibilidad de creación de diagramas de bloques y diagramas de estructura para visualizar organización jerárquica del modelo, diagramas de estado, vistas previas en 3D y herramientas de dibujo.

En resumen OpenModelica ofrece una interfaz de edición gráfica intuitiva para la creación y modificación de modelos dinámicos, con herramientas para representar de forma visual los modelos en varios formatos, facilitando la comprensión y diseño del mismo.

A su vez dispone de una suite de herramientas que lo hacen una solución muy funcional para realizar cualquier trabajo.

Herramienta	Descripción
OMEdit (OpenModelica Connection Editor)	Es una interfaz gráfica que permite a los usuarios la creación de modelos editando conexiones. Es extensible, reutilizando modelos gráficos y textuales.
OMShell (Interactive OpenModelica Shell)	Interprete de consola que usa comandos y expresiones de Modelica para interactuar con el modelo y la simulación.
OMNotebook (OpenModelica Notebook)	Editor de texto sencillo
Equation Model Debugger	Es un depurador que nos permite localizar fallos en el código, además nos da un seguimiento de las transformaciones del código hasta ser compilado en C
OMOptim (Optimization subsystem)	Es un subsistema de optimización para la interfaz gráfica
MDT (Modelica Development Tooling)	Proporciona un navegador entre el sistema de clases y archivos.

Tabla 2-3 Herramientas de OpenModelica (OpenModelica, 2023)

Funciona compilando el lenguaje Modelica a lenguaje C, pero a su vez tiene componentes que permiten trabajar con una interfaz en Python o MATLAB, también dispone de un compilador a través de una API basada en CORBA que le permite integrarse en otros entornos como Eclipse. En la Figura 2-9 podemos ver un ejemplo de la interfaz gráfica que ofrece OpenModelica

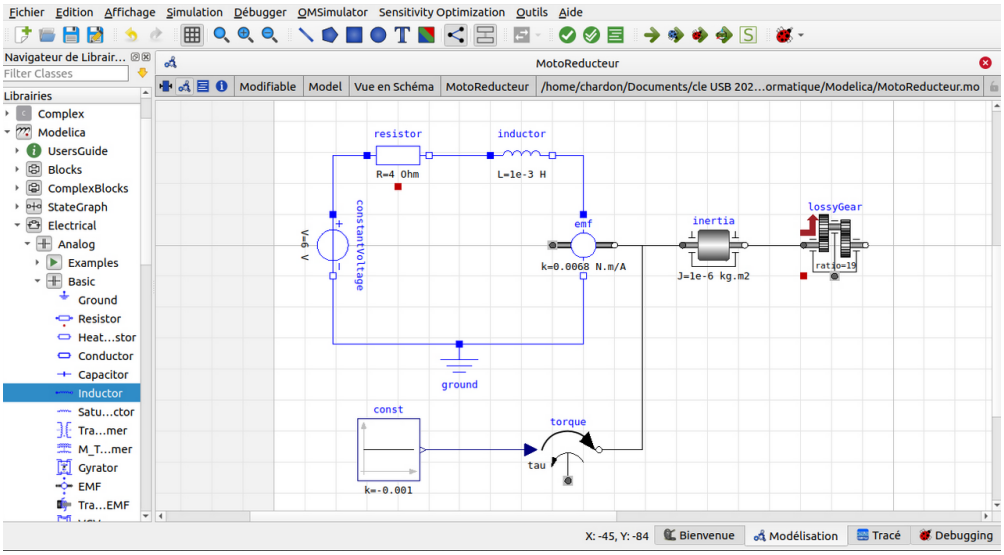


Figura 2-9 Interfaz de OpenModelica

2.4.2 Dymola

En 1977 Hilding Elmqvist creó SIMNON (SIMulation of NON-linear systems) (Elmqvist, 1977), cuando preparaba su tesis doctoral. En esta aplicación, escrita en FORTRAN, el usuario podía crear modelos introduciendo expresiones matemáticas, la gran novedad fue la simulación mixta de sistemas continuos y discretos en el tiempo. Elmqvist comprendió que era necesario un lenguaje orientado a objetos que permitiese la interacción entre submodelos, este fue el nacimiento de Dymola. En 1992, Elmqvist creó la empresa sueca Dynasim AB, y en 2006 la empresa francesa Dassault Systèmes adquirió Dynasim AB y comenzó a integrar Dymola en CATIA (Elmqvist, 2014).

Aunque el modelado realizado en el entorno se transcriba en lenguaje Modelica, se apoya en lenguaje C para realizar una traducción del modelo plano, es por ello por lo que necesita de un compilador de C para funcionar.

Dymola se distribuye bajo una licencia propietaria aunque dispone de una versión de evaluación con características limitadas, hoy en día es uno de los entornos más extendidos para modelado y simulación y es usado ampliamente en la industria del automóvil, aeroespacial o en equipamiento industrial (Dassault Systèmes, 2023).

Tiene una interfaz gráfica muy intuitiva, que divide el espacio de trabajo en dos entornos, uno de simulación y otro de modelado como podemos ver en las Figuras 2-10 y 2-11. A su vez cada entorno se divide en una ventana de propiedades, otra de librerías y un lienzo donde se puede visualizar el modelo gráficamente. Además de todas las funcionalidades gráficas que se describen en OpenModelica hay que añadir que Dymola dispone de una interfaz de usuario personalizable, adaptarse a las necesidades específicas del usuario.

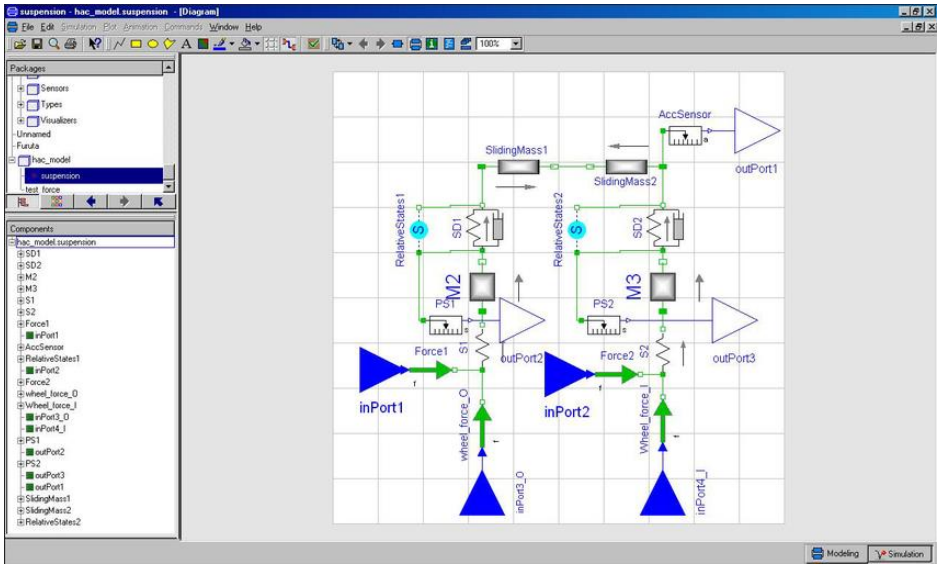


Figura 2-10 Entorno de modelado de Dymola

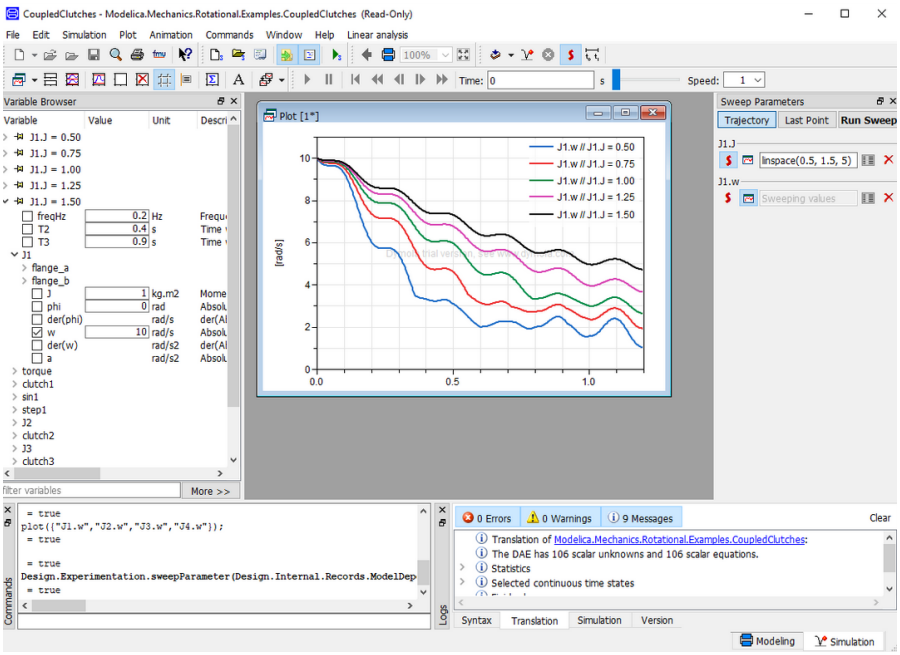


Figura 2-11 Entorno de simulación de Dymola

2.5 Herramientas de transcripción de diagramas en código

Para el componente de frontend se hace crítico tener una tecnología rápida y ligera que fuera capaz de generar una interfaz gráfica la cual pueda representar grafos o diagramas con elementos conectados y programables.

Actualmente la única tecnología de tipo web que es capaz de realizar actualizaciones dinámicas en un entorno web es JavaScript y todos sus variantes JQuery, Angular, React Native etc. Es por ello por lo que comenzamos un proceso de investigación de los mejores frameworks en JavaScript para realizar representaciones de diagramas.

- Draw.io: herramienta gratuita de diagramas que permite exportar e importar modelos.
- PlantText: es una herramienta online. Es usado para transcribir diagramas a código en Python, C++ y Java.
- WebSequenceDiagrams: es una herramienta online, al igual que PlantText para crear diagramas de secuencia y transcribirlos a código en varios lenguajes, como Python, Java y C++.
- yUML: es una herramienta gratuita que permite crear diagramas UML de forma online.

2.5.1 GoJS

Es un framework que permite implementación de diagramas interactivos con fácil navegación, gestión de eventos sobre los nodos, conexiones y ofrece multitud de plantillas con ejemplos interactivos. Es muy usada para hacer diagramas UML de definición de clases, diagramas de interacción, diagramas temporales, mapas de conceptos, o diagramas BPMN como el de la Figura 2-12.

GoJS dispone de una API documentada, en la cual se pueden aplicar configuraciones para obtener un tipo de diagrama muy concreto, pero tiene un gran contra y es la opacidad a la hora de implementar las funciones a las que se invoca, esto hace que no pueda ser totalmente versátil ni configurable a nivel del detalle.

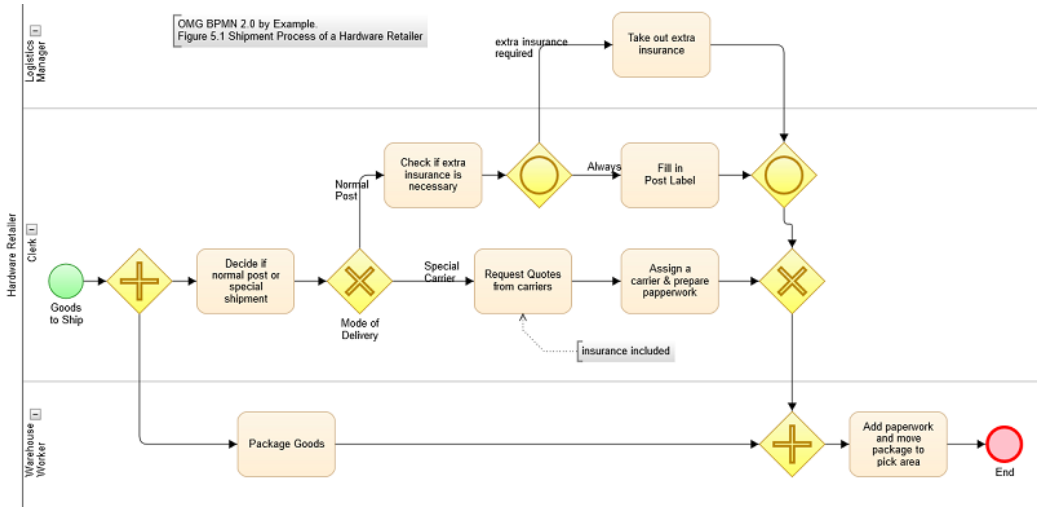


Figura 2-12 Ejemplo de un diagrama BPMN en GoJS

El primer prototipo realizado de la aplicación de estudio fue realizado con GoJS pero desechamos este primer bosquejo en pro de otro framework: mxGraph.

2.5.2 mxGraph

mxGraph es una librería en JavaScript que permite gobernar todos aquellos procesos que no eran accesibles ni editables con GoJS. La principal ventaja de este framework es que trabaja juntamente con HTML e imágenes SVG y tiene una gran comunidad que soporta actualizaciones, bugs y plantillas.

Actualmente mxGraph está en su versión 4.2.2 y está licenciado bajo Apache License 2.0 lo cual lo hace una tecnología de software libre. Aunque ahora hagamos una introducción de la tecnología detallaremos como se ha hecho la implementación de código más adelante.

2.6 Arquitectura de desarrollo

Cuando se realiza el análisis previo al diseño de cualquier aplicación se hace necesario empezar por la arquitectura global. Vamos a detallar como se estructurará en capas nuestra aplicación, como se va a persistir la información, las tecnologías que vamos a usar para implementarla y cómo será la interacción del usuario con la interfaz gráfica.

La propia definición del proyecto ya especifica que se va a tratar de una aplicación web. Esto hace que forzosamente tengamos que recurrir a un modelo concreto: cliente-servidor. La aplicación debe centralizarse en un nodo servidor, al cual llamen los clientes desde un navegador. El modelo cliente-servidor tiene dos ventajas principales:

- La **escalabilidad**: es un modelo en el cual se pueden realizar escalados tanto horizontales (añadir nodos que cooperen conjuntamente) como vertical (aumentando la capacidad del nodo de procesamiento).
- Fácil **mantenimiento**: siempre es más fácil reemplazar, reparar, actualizar o trasladar un servidor que un conjunto de recursos que están distribuidos de forma descentralizada y deslocalizada.

Ahora bien, hablemos de los componentes que debe tener el servidor para dar soporte a la funcionalidad deseada.

La web debe ser un portal en el cual los usuarios puedan interactuar creando distintos proyectos, por lo que debemos persistir al menos, información asociada a los usuarios, además de las características de cada modelo creado por ellos: necesitamos una base de datos. Es muy frecuente que el servidor de aplicaciones deba conectarse con una base de datos para obtener los datos solicitados durante la ejecución del código. Dicha base de datos puede residir en la misma máquina que el servidor web o en otro host conectado en red. Es a esto a lo que se llama arquitectura de 3 capas, según Luján-Mora (2002), el objetivo de aumentar el número de niveles en una aplicación distribuida es lograr una mayor independencia entre un nivel y otro, lo que facilita la portabilidad en entornos heterogéneos y la escalabilidad en caso de incorporación de nuevos clientes. La Figura 2-13 muestra un esquema de la arquitectura de 3 capas.

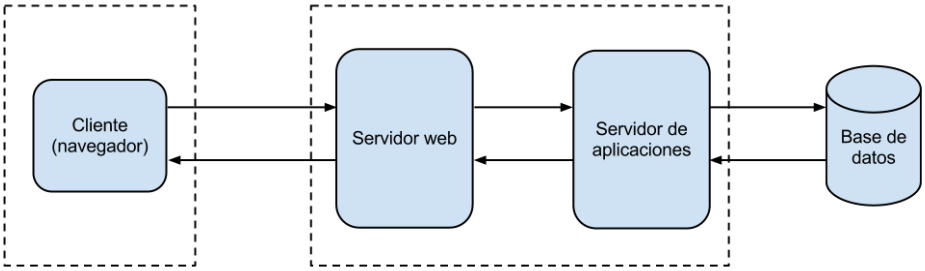


Figura 2-13 Modelo con servidor de aplicaciones

¿Cómo funcionaría internamente nuestra aplicación para dar una interfaz de usuario, mantener un modelo que interactúe directamente con una base de datos y a la vez pueda realizar procesos de transformación con esos datos? Hemos optado por el patrón de desarrollo Modelo Vista Controlador o MVC.

2.6.1 Patrón MVC

Model-View-Controller (MVC) es un patrón de arquitectura de software que separa una aplicación en tres componentes principales, tres capas acotadas con una responsabilidad concreta: el modelo, la vista y el controlador. En la Figura 2-14 podemos ver la interacción entre estos componentes.

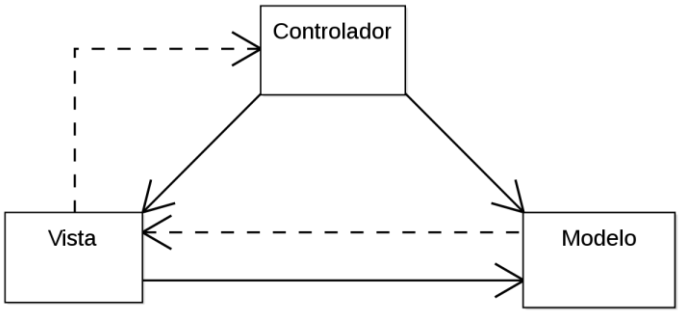


Figura 2-14 Modelo MVC

Modelos: es la capa donde trabajaremos con los datos, en ella se implementan todas las responsabilidades de interacción con bases de datos y definición e instancias de clases que servirán de apoyo para las demás tareas. Lo normal en las tecnologías web actuales es usar mapeadores objeto-relacional o ORMs, se trata de frameworks que de una forma transparente al usuario definan y mapeen la base de datos a partir del mantenimiento de las clases en memoria o viceversa. Adicionalmente, la capa de modelo cuenta con una subcapa a la que llamaremos capa de servicio

o capa de negocio, esta será la encargada de realizar las transformaciones sobre los datos y llevar sobre ellos los procesos que mande la interfaz de usuario gobernada por el controlador.

Controladores: es una capa de comunicación entre el modelo y las vistas en la cual se define la interacción mediante métodos HTML (GET, POST; PUT, etc.) dando como resultado un modelo de vista de usuario. Esta capa no debe definir lógica de transformación de datos alguna y debe ser meramente una pasarela entre el modelo y la vista obedeciendo la navegación del usuario.

Vistas: La vista define la interfaz de usuario, a partir de los modelos devueltos por los controladores, se usan tecnologías de tipo web como HTML, CSS, JavaScript, etc.. para desarrollar esta capa. Además, interactuará con los controladores mediante HTTP. En ella se deben definir los eventos que el usuario usará para interactuar como los resultados producidos por un clic de botón o el arrastre de un elemento.

2.7 Entorno de desarrollo

Vamos a hablar un poco de los entornos de desarrollo usado para la aplicación, principalmente dos: Visual Studio 2019 en el que hemos desarrollado todo el código de la aplicación , además de ser el entorno de compilación, test y depuración, y SQL Server Management Studio, el gestor de base de datos mediante el cual administramos la capa de datos.

2.7.1 Visual Studio

Visual Studio es un IDE desarrollado por Microsoft y especialmente pensado para proyecto y soluciones de plataforma .NET aunque es compatible con casi todos los lenguajes de programación extendidos como C++, C#, F#, Java, Python, CLisp, etc. Desde su primera versión en 1998 hasta la actualidad ha crecido y perfeccionado a lo largo de 12 versiones en las cuales se han incluido multitud de herramientas y plugin que lo hace compatible con casi todo lo que existe en el mercado.

Me he decantado por tecnología .NET desde un principio. La combinación de C# en backend con ASP.NET de parte de frontend, hace que sea muy fácil implementar un modelo MVC. Visual Studio ofrece plantillas donde al crear un proyecto de 0 te construye una estructura de solución básica y jerarquizada siguiendo este patrón. En la Figura 2-15 podemos ver como se crea un proyecto nuevo en Visual Studio. En la Figura 2-16 podemos ver como la creación del proyecto anterior da lugar a una estructura de proyecto con distintos ficheros y directorios.

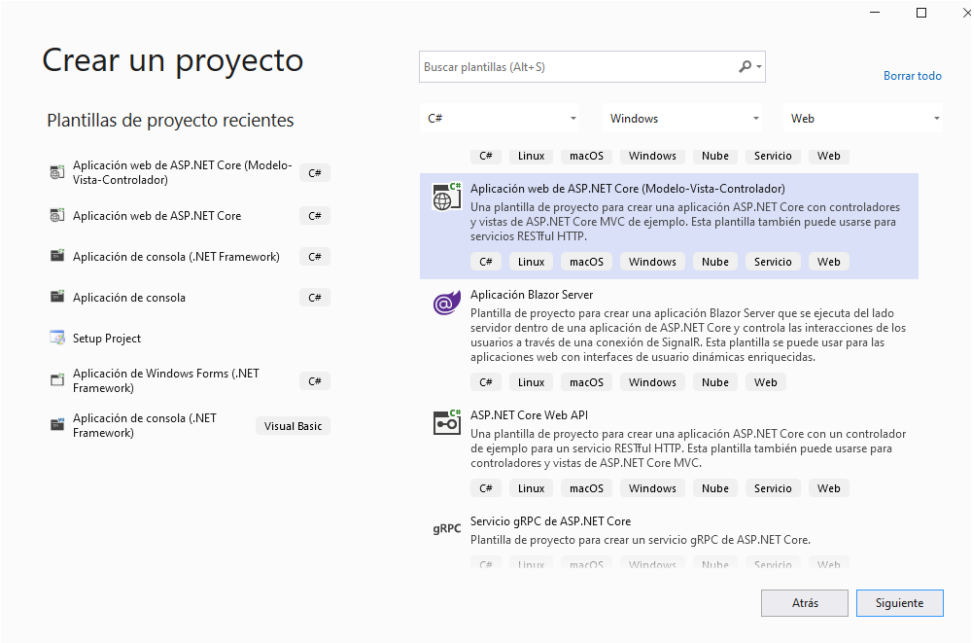


Figura 2-15 Visual Studio -Creación de un proyecto MVC nuevo

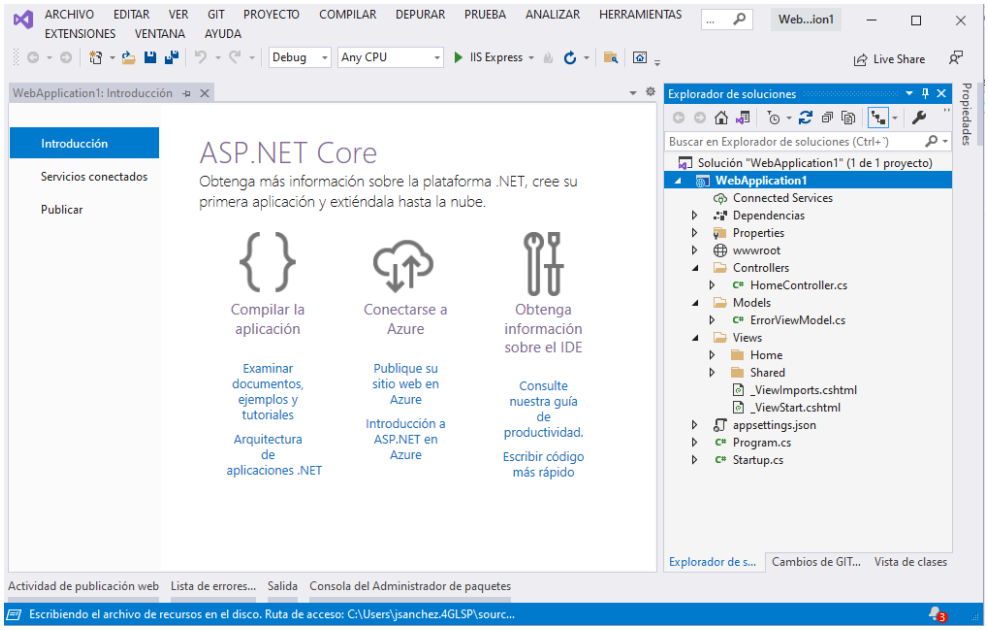


Figura 2-16 Visual Studio - Estructura de un nuevo proyecto MVC

2.7.2 SQL Server Management Studio

SQL Server es la tecnología de bases de datos nativa de .NET y por tanto con mayor compatibilidad con proyectos de Microsoft. No obstante Visual Studio y en general el framework .NET puede trabajar con cualquier origen de datos.

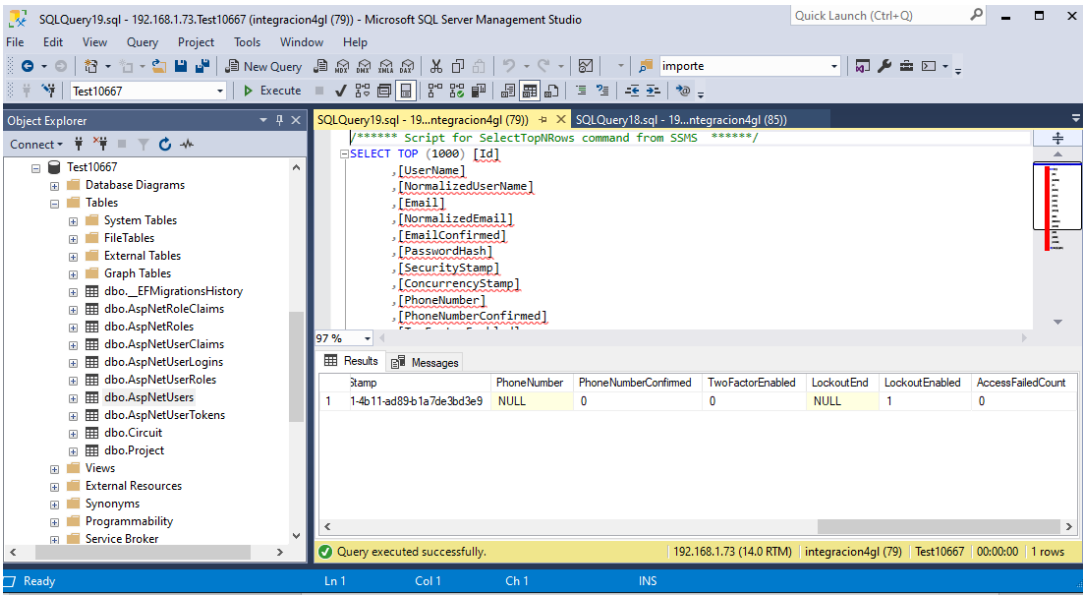


Figura 2-17 SQL Server Management Studio

Management Studio, Figura 2-17, es la herramienta gráfica de consulta para conectar con el motor de base de datos. Es un entorno sencillo en el cual se puede visualizar la jerarquía de elementos de catálogo (tablas, consultas, funciones, etc.) a la izquierda y un editor de consultas a la derecha

2.8 Lenguajes de programación

Vamos a usar dos grupos de lenguajes de programación. Por un lado lenguajes de servidor para implementar lógica de negocio, lógica de acceso a datos, entidades, etc. Por otro lado usaremos lenguajes del lado de cliente para crear vistas de representación al usuario y enriquecer la interfaz gráfica.

2.8.1 C# y Asp.NET

La plataforma .Net cuenta con dos lenguajes principales de desarrollo tradicionales, estos son C# y Visual Basic. Aunque ambos son traducidos en su fase de compilación al mismo bytecode, en sintaxis C# está más estandarizado en el sentido de que se parece mucho más a otros

lenguajes de uso común como Java o C, por lo que su comprensión y manejo es más sencillo, además de que tiene un soporte mayoritario en la comunidad de programadores. Hemos optado por C# para la codificación de toda la lógica contenida en backend.

A lo largo de la primera década del actual siglo empezaron a emerger todos los lenguajes de desarrollo web dinámicos, con el auge de internet se hacía necesario una solución a las anticuadas páginas estáticas en HTML, es aquí cuando surgieron PHP, JSP y ASP.NET, ASP.NET o Active Server Pages es el lenguaje nativo de la plataforma .NET para desarrollo de páginas dinámicas, tiene una sintaxis que permite mezclar código HTML, JavaScript, CSS y es capaz de reutilizar componentes creados para facilitar la factorización de código.

2.8.2 HTML, JavaScript y JQuery

Nuestra aplicación es una solución de tipo web, y como tal es obligado recurrir a las tecnologías típicas, con HTML podemos dar una estructura a las páginas creando un diseño basado en contenedores y tablas, añadiendo recursos de tipo textual e imágenes.

JavaScript es un lenguaje ligero e interpretado que permite definir eventos y acciones dinámicas sobre el código HTML estático, esto es necesario en nuestra aplicación ya que van a ser continuas las acciones de crear objetos en tiempo de ejecución, arrastrarlos a un contendor, cambiar sus propiedades e interactuar con ellos. Nos ayudaremos de la librería JQuery que es un framework sobre JavaScript para simplificar la escritura de código, además del medio para usar mxGraph.

2.9 Conclusiones

El problema de realizar una aplicación de modelado es complejo y necesita tener en cuenta muchos elementos. El comportamiento de los elementos, la interacción que tienen entre sí, los algoritmos de traducción al lenguaje resultante, las tecnologías implicadas y la arquitectura de la aplicación. Por ello ha sido necesario realizar un estudio de la evolución de la simulación en tiempo continuo, en el cual hemos visto los distintos hitos que han provocado la creación de distintos lenguajes y distintas herramientas.

La propuesta desarrollada se ha centrado en la estructuración del código Modelica, dando lugar a bloques concretos donde se define en un primer lugar la definición de los elementos, después el comportamiento interno de cada uno de ellos y por último la interrelación entre ellos.

Hemos escogido Modelica como lenguaje de referencia para trabajar con modelos físicos por varias razones. En primer lugar, es orientado a objetos lo que permite la reutilización de componentes y la organización de modelos complejos en elementos más simples. En segundo lugar, permite describir sistemas dinámicos sin especificar el orden de ecuaciones o eventos, a lo que llamamos descripción no causal. La interoperabilidad y la integración con otros sistemas, además de la amplia comunidad hacen de Modelica la mejor opción para trabajar.

En este capítulo hemos hecho una parada en los aspectos más relevantes del lenguaje y en los entornos de desarrollo más extendidos y usados por nuestra aplicación.

La exploración de distintas tecnologías hace posible usar una arquitectura tecnología robusta, usando el patrón MVC como *ley motiv*. La plataforma .NET, es un amplio conjunto de bibliotecas de desarrollo que pueden ser utilizadas con el objetivo principal de acelerar el desarrollo de software, en este caso nos valemos de C#, ASP.NET y SQL Server para dar forma a toda la solución.

3 Análisis y planificación del proyecto

3.1 Introducción

Este capítulo describirá el análisis previo al diseño y desarrollo de la aplicación donde nos centraremos en el catálogo de requisitos de la aplicación así como los diagramas de interacción de la aplicación con el usuario y la planificación del proyecto.

3.2 Catálogo de requisitos

Un análisis de requisitos es el estudio de los requerimientos de negocio necesarios para satisfacer las necesidades del usuario que la aplicación debe cubrir. Este análisis se plasma en un documento en forma de lista que luego servirá a la empresa o desarrollador para establecer una plantilla por la cual se valide el resultado final de la aplicación.

Una de las ventajas de tener un buen análisis de requisitos es que de cara al cliente o futuros usuarios, se formalice de forma contractual que debe hacer la aplicación encargada, de tal manera que se establezcan claramente los límites de los funcionales a desarrollar. Se trata de un proceso de investigación y documentación que permite definir y entender los requisitos del sistema que se va a desarrollar. El objetivo del análisis de requisitos es asegurar que el sistema cumpla con los objetivos y necesidades del usuario final.

Al trabajar mediante una metodología ágil un catálogo de requisitos se puede plasmar en un documento o Product Backlog el cual desglosa información útil al equipo desarrollador como prioridades o estimaciones de tiempo en cada funcional que a la vez se puede desglosar en subtareas.

Requisitos funcionales		Descripción
RF1 – Gestión de usuarios		Se debe crear una gestión completa de usuarios.
	RF1-1 Registro de usuarios	Los usuarios deben poder registrarse en la aplicación usando un email y un password.
	RF1-2 Login de usuarios	Los usuarios deben poder ingresar en la página web usando su autenticación con email y password, accediendo a sus trabajos.
	RF1-3 Recuperación de contraseña	Un usuario que ha extraviado la contraseña debe poder recuperarla.
RF2 – Gestión de trabajos		Los usuarios deben tener acceso a sus trabajos siendo capaces de crear, eliminar y editar modelos dentro de estos.
	RF2-1 _Gestión de proyectos	Los usuarios pueden gestionar un proyecto completo que servirá de contenedor a los modelos implementados.
	RF2-2 Gestión de circuitos	Los usuarios deben poder recuperar y editar los circuitos, donde editarán directamente los modelos.
RF3 – Gestión de modelos		Debe ser posible crear modelos desde cero o trabajar sobre modelos ya guardados.
	RF3-1 Guardado de modelos	Los modelos se podrán guardar en base de datos con el fin de persistir el trabajo de cada usuario.
	RF3-2 edición de modelos	Los modelos deben poder recuperarse de la base de datos para poder editarlos
RF4 – Construir un editor		Debe ser posible la edición de modelos mediante un editor gráfico.
	RF4-1 Crear paleta de componentes	Debe existir una paleta de componentes eléctricos que permita seleccionar componentes para añadirlos al modelo.
	RF4-2 Propiedades de componentes	Se deben poder editar las propiedades de cada componente ajustando numéricamente los valores tanto numéricos como literales.
	RF4-3 Conexiones entre componentes	Los componentes deben ser capaces de conectarse entre sí mediante nodos.
	RF4-4 Barra de herramientas del modelo	EL modelo debe tener la posibilidad de rehacer o deshacer cambios, guardar o cargar modelos y eliminar elementos del modelo
RF5 – Comprobaciones		
	RF5-1 Comprobaciones de conexiones	Se debe comprobar que todos los nodos tengan conexiones correctas
	RF5-2 Comprobaciones de fuentes	No puede haber dos generadores de tensión conectados en paralelo. No puede haber dos generadores de corriente conectados en serie
	RF5-3 Comprobaciones de nodos	Todos los circuitos deben tener al menos un nodo desde el cual se realicen los cálculos de corrientes y voltajes. No puede asignarse el mismo nombre a 2 nodos o componentes.
	RF5-4 Comprobaciones nodo tierra	Todos los circuitos deben tener un único nodo tierra. Los componentes solo pueden conectarse a un nodo o a tierra.
RF6 -Generación de código		
	RF6-1 Código en formato atómico	Se debe poder generar el código Modelica en formato atómico
	RF6-2 Código en formato compuesto	Debe ser posible generar el código Modelica en formato compuesto instanciando las clases de la librería estándard de Modelica
	RF6-3 Posicionamiento de elementos	En el formato compuesto debe ser posible calcular el posicionamiento de los elementos sobre el lienzo para posibilitar la visualización gráfica en los entornos de Modelica

Tabla 3-1 Requisitos funcionales

Además de los requisitos funcionales no debemos menospreciar los requisitos no funcionales, estos describirán las características internas tales como rendimiento, disponibilidad, accesibilidad o seguridad.

Requisitos No funcionales	Descripción
RNF-1 Disponibilidad	La aplicación debe estar disponible los 365 días del año, el servidor web donde se aloje debe tener al menos una disponibilidad en su SLA de al menos 99.9%
RNF-2 Rendimiento	Toda funcionalidad del sistema y transacción debe responder al usuario en menos de 5 segundos
RNF-3 Capacidad	La aplicación debe tener una respuesta óptima para albergar al menos a 1000 usuarios trabajando paralelamente.
RNF-4 Recursos	La aplicación debe consumir un porcentaje inferior al 50% de la RAM del servidor donde esté alojado, además de garantizar que no haya carga de CPU superior al 60%
RNF-5 Seguridad	Todas las contraseñas de usuarios deben ser encriptadas, de tal manera que no sea posible ningún ataque externo.
RNF-6 Accesibilidad	La página debe ser accesible para cualquier usuario, teniendo en cuenta los recursos textuales, multimedia y el acceso de usuarios con capacidades visuales o auditivas reducidas

Tabla 3-2 Requisitos no funcionales

3.3 Análisis y especificaciones

En el caso de la gestión de usuarios usaremos “Identity”, esto es el framework de .NET indicado para la gestión de usuarios dentro de una aplicación. La propia instalación del framework da como resultado un modelo de tablas standard que albergará todo lo necesario para la persistencia de usuarios. Este modelo de tablas se basa en el diagrama de la Figura 3-3, de todas las tablas creadas hay dos que son las más importantes y de mayor uso, AspNetUsers donde se guardan las cunetas de usuario y AspNetLogins donde se guardan los inicios de sesión .

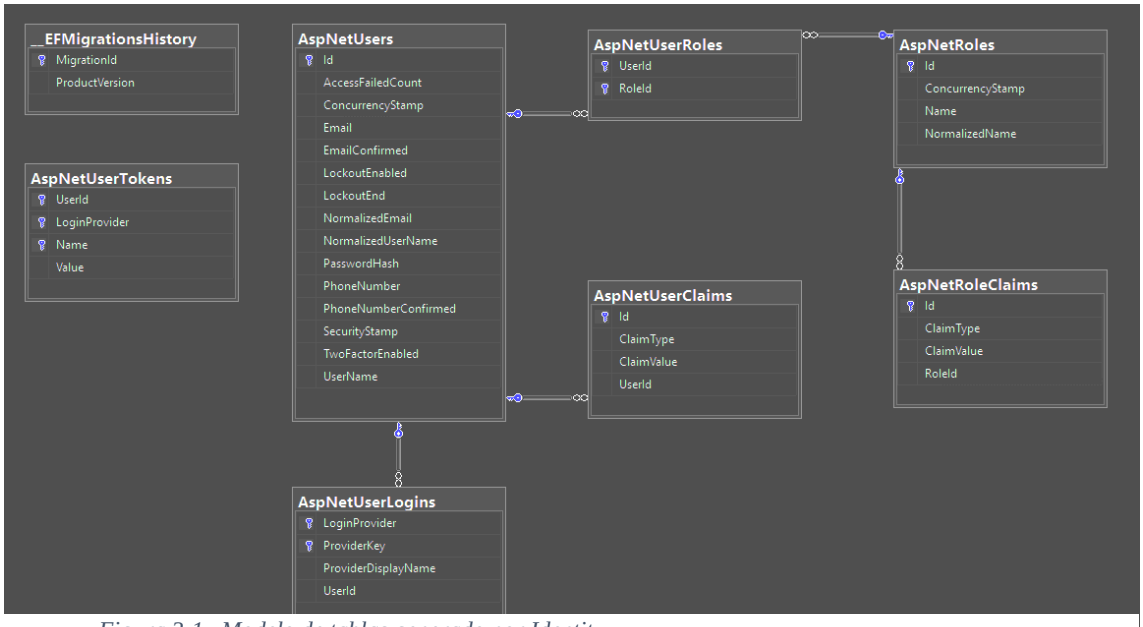


Figura 3-1 Modelo de tablas generado por Identity

Este modelo, se debe complementar con un modelo conceptual de tablas interno para el guardado de proyectos y circuitos.

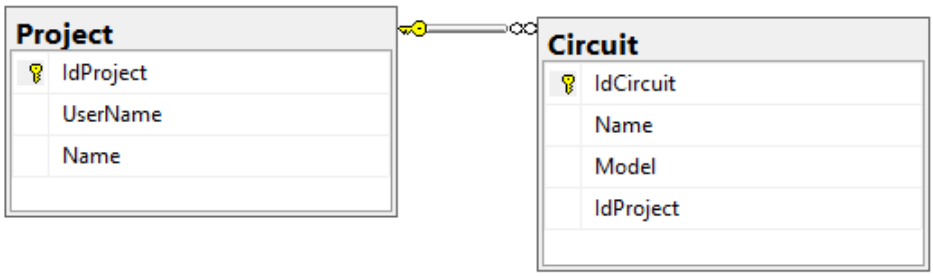


Figura 3-2 Diagrama conceptual de proyectos y circuitos

Donde Project será la tabla de guardado de proyectos:

Project

- IdProject – identificador interno
- UserName – Nombre de usuario que ha creado el proyecto
- Name – Nombre del proyecto

Y la tabla de Circuitos

Circuit

- IdCircuit – identificador del circuito
- IdProject – Proyecto vinculado
- Name – Nombre del circuito
- Model – Código de mxGraph que representa el modelo

Los casos de uso aplicación-usuario son muy sencillos ya que el usuario se va a limitar a usar la aplicación de la siguiente manera en un escenario normal:

- El usuario se registrará en la aplicación
- El usuario una serie de proyectos que albergarán los modelos
- El usuario creará un modelo o circuito nuevo usando la interfaz gráfica, para ello deberá generar el código en Modelica. Existirán otros escenarios alternativos donde se podrán eliminar modelos y proyectos

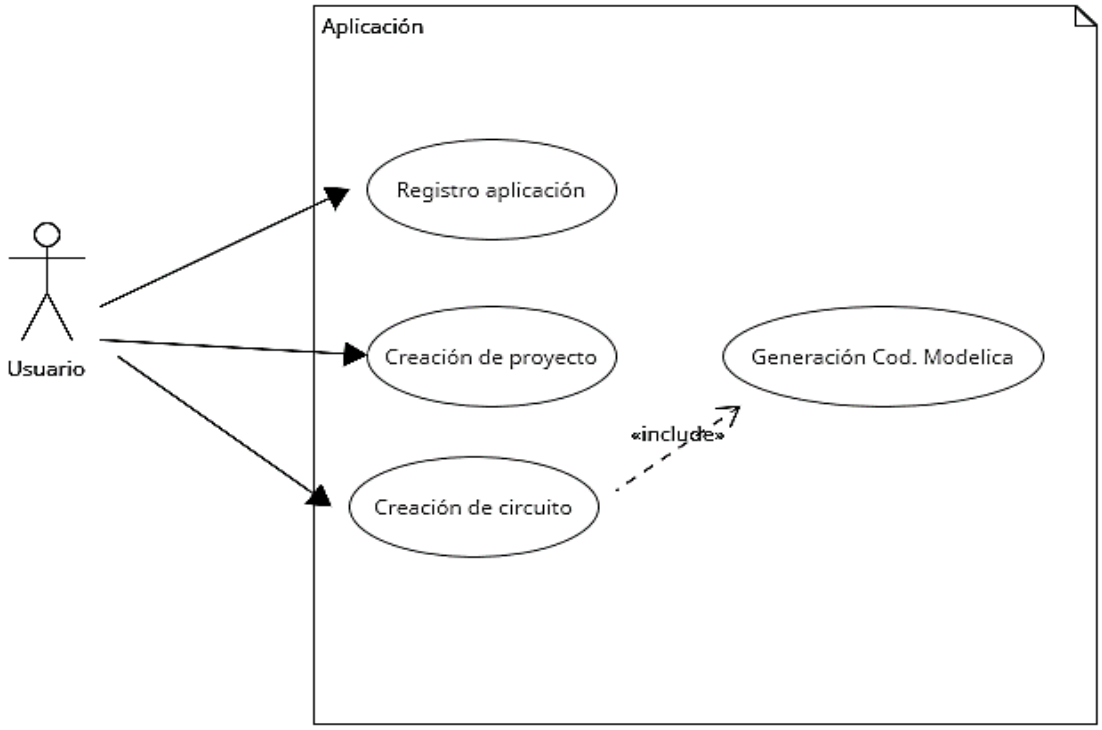


Figura 3-3 Casos de uso de la aplicación

3.4 Planificación

Para desarrollar este proyecto partiremos de una metodología ágil basada en iteraciones, en cada iteración se realizarán todas las fases clásicas del desarrollo software: un análisis de los funcionales a realizar, diseño, construcción y pruebas.

Las iteraciones se planifican en función de los requisitos funcionales definidos en la Sección 3.1 y desglosando en tareas concretas los trabajos a realizar. De esta manera, la Tabla 3.6 describe este desglose de tareas.

Iteración	Tareas	Requisito relacionado
1 – Primera prueba de concepto. En esta primera iteración se pretende crear un editor simple donde se pueda generar un modelo gráfico con elementos conectables	1-1 Crear editor con elementos conectables	RF4
	1-2 Crear paleta de elementos	RF4-1
	1-3 Gestionar creación y eliminado de los elementos	RF4-1, RF4-3
	1-4 Gestionar modelo subyacente a los elementos conectados	RF4
	1-5 Crear barra de herramientas del modelo	RF4-4
	1-6 Gestionar propiedades de los componentes	RF4-2
	1-7 Guardar código del modelo mxGrpah	RF4
2 – Generación de código en formato atómico	2-1 interpretar modelo generado en mxGraph y traducirlo a clases de C#	RF6-1
	2-2 Definir a nivel de negocio las ecuaciones de cada componente y vincularlo a su instancia de clase	RF6-1
	2-3 Parsear modelo y generar variables y parámetros	RF6-1
	2-4 Realizar algoritmo que de cómo resulta las ecuaciones en los nodos y las ecuaciones constitutivas en Modelica	RF6-1
3 – Generación de código en formato compuesto	3-1 Reutilizar modelo de clases en C# para crear modelo compuesto	RF6-2
	3-2 Traducir cada elemento a una clase de la MSL	RF6-2
	3-3 Calcular posicionamiento de cada elemento	RF6-3
4 – Implementar comprobaciones . En esta iteración el objetivo es que se quede un editor consistente donde el código generado tenga sentido	4-1 Crear reglas de comprobación en la capa de negocio sobre el modelo	RF5
5 - Gestión de usuarios y trabajos	5-1 Creación de modelo en Identity	RF1
	5-2 Instanciado de a BD y creación de las clases proyectos y circuitos	RF2
	5-3 Creación de páginas para implementar toda la lógica de gestión de usuarios	RF1
	5-4 Creación de páginas para incorporar lógica de creación de proyectos y circuitos	RF2
	5-5 Guardado y recuperación de modelos desde la base de datos	RF3

Tabla 3-3 Desglose de iteraciones y tareas del proyecto

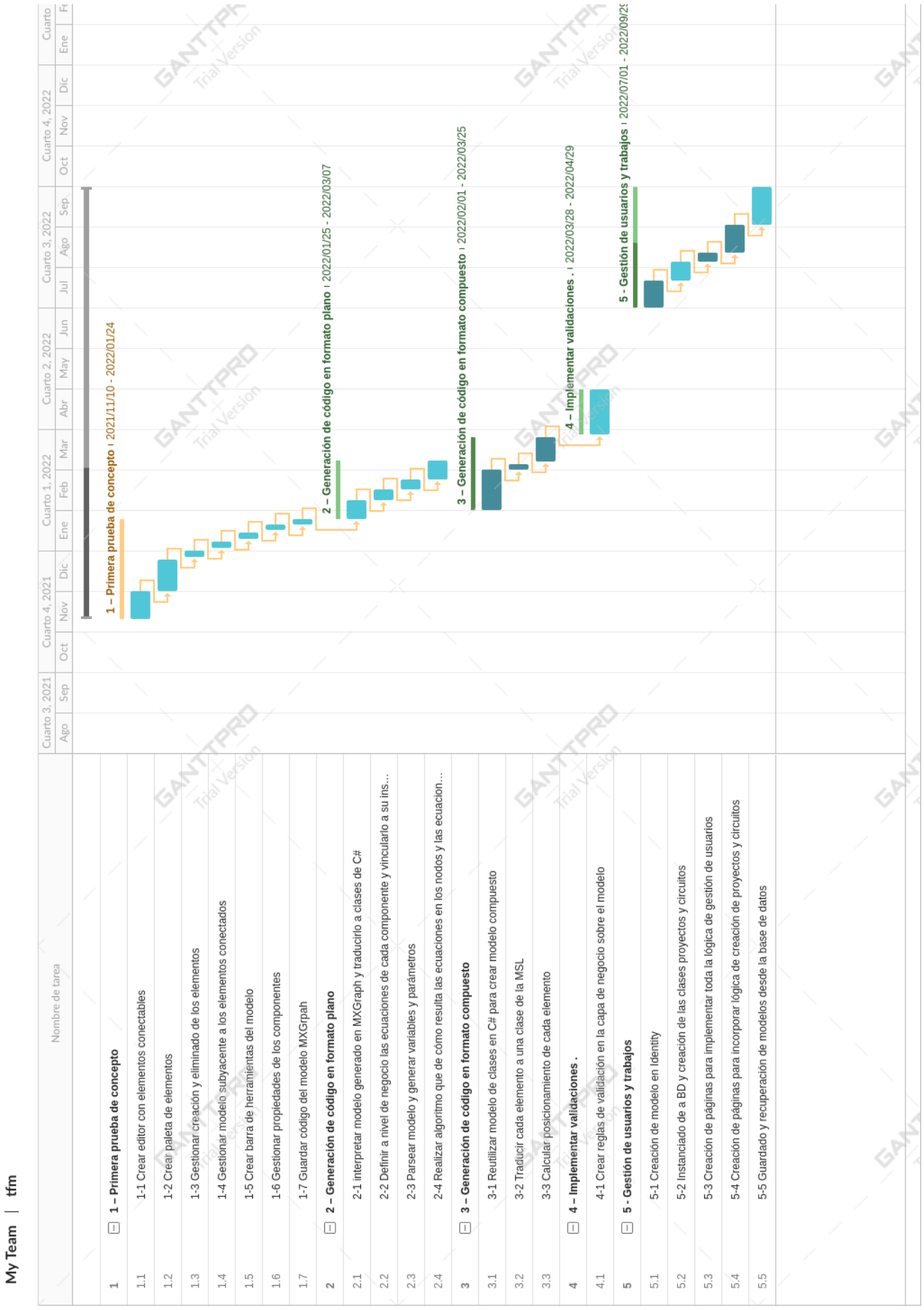


Tabla 3-4 Diagrama de Gantt con planificación temporal

3.5 Conclusiones

En este capítulo hemos profundizado en el análisis inicial de la aplicación, desglosando todos los requisitos funcionales y no funcionales, definiendo el modelo conceptual de clases, explicando los casos de uso y la metodología a seguir basada en iteraciones.

La planificación de proyectos software es clave para administrar eficientemente los recursos de un proyecto, en mi caso, jugar con el recurso tiempo ha sido clave para garantizar que los funcionales mínimos hayan sido desarrollados y con un estándar de calidad mínimo aceptable.

4 Definición y traducción de los modelos

4.1 Introducción

Una de las mayores complejidades de este proyecto es el trabajo con diferentes modelos. Aun partiendo en esencia del mismo circuito, se expresará de distinta manera. Este capítulo tratará de explicar todos los modelos que vamos a usar desde que hacemos un esquema mediante un diagrama en la interfaz gráfica hasta que ese modelo permanece en memoria para ser evaluado y hasta que se vuelve a traducir a código Modelica.

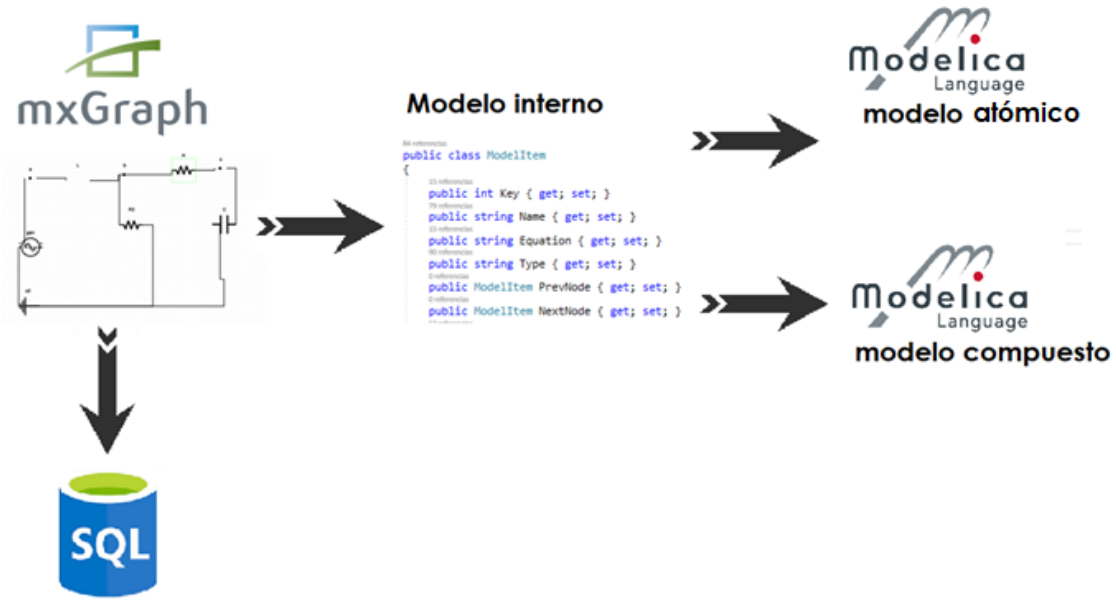
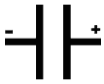
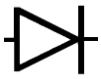


Figura 4-1 Esquema de modelos y transformaciones entre ellos usados en la aplicación

Como podemos ver en la Figura 4.1, vamos a partir de un modelo gráfico definido en XML que hace que la interfaz de mxGraph sea capaz de interpretar para funcionar. Este modelo es cerrado y exclusivo de este framework de trabajo. Para poder hacer un análisis de lo que el usuario va a dibujar en la interfaz gráfica, es necesario convertirlo a clases de programación en C#, es sobre estas clases donde haremos la comprobaciones pertinentes. Por último, se realizará la traducción de código a Modelica dando la posibilidad de tener un modelo atómico o modelo compuesto.

Componente		Relación constitutiva	Magnitudes
	Condensador	$C.\frac{du}{dt} = i$	C = Capacidad (F)
	Diodo	$i = I_s\,.(\exp\left(\frac{u}{V_t}\right) - 1)$	I_s = Corriente de saturación (A) V_t = Tensión térmica
	Inductor	$u = L\,\frac{di}{dt}$	L = Inductancia (H)
	Generador de tensión sinusoidal	$u = U_0\,.\sin(\omega.t + \varphi)$	U_0 = Amplitud (V) ω = Frecuencia angular (rad/s) φ = Desfase (rad)

	Generador de tensión de onda cuadrada	`when sample(0,T) then`
		`up = not pre(up);`
		`when;`
		`u = if up then` U_1 `else 0;`
			U_1 = Amplitud (V)
			T = semiperiodo (s)

	Generador de corriente de onda cuadrada	`when sample(0,T) then`
		`up = not pre(up);`
		`when;`
		`i = if up then` I_1 `else 0;`
			I_1 =Corriente (A)
			T = semiperiodo (s)

	Resistencia	$u = i\,.R$	R = Resistencia (ohm)
	Nodo tierra	$u = 0$	

	Interruptor	`Real s(final unit="1")`
		`constant SI.Voltage unitVoltage=1;`
		`constant SI.Current unitCurrent=1;`
		`when sample(0,T) then`
		`off = not pre(off);`
		`end when;`
		`u = (s*unitCurrent)*(if off then 1`
		`else Ron)`
		`i = (s*unitVoltage)*(if off then Go`
		`else 1)`
			T = semiperiodo (s)
			Ron = Resistencia = 1e-5 (ohm)
			Goff = Conductancia = 1e-5 (S)

Tabla 4-1 Componentes eléctricos

En la Tabla 4.1 podemos ver los componentes eléctricos que intervienen en nuestra aplicación, estos componentes tendrán una clase o traducción en cada uno de los modelos descritos. En la interfaz gráfica de usuario, usaremos estos componentes dentro de una paleta en la cual se podrán seleccionar para arrastrar al modelo y conectarlos entre sí.

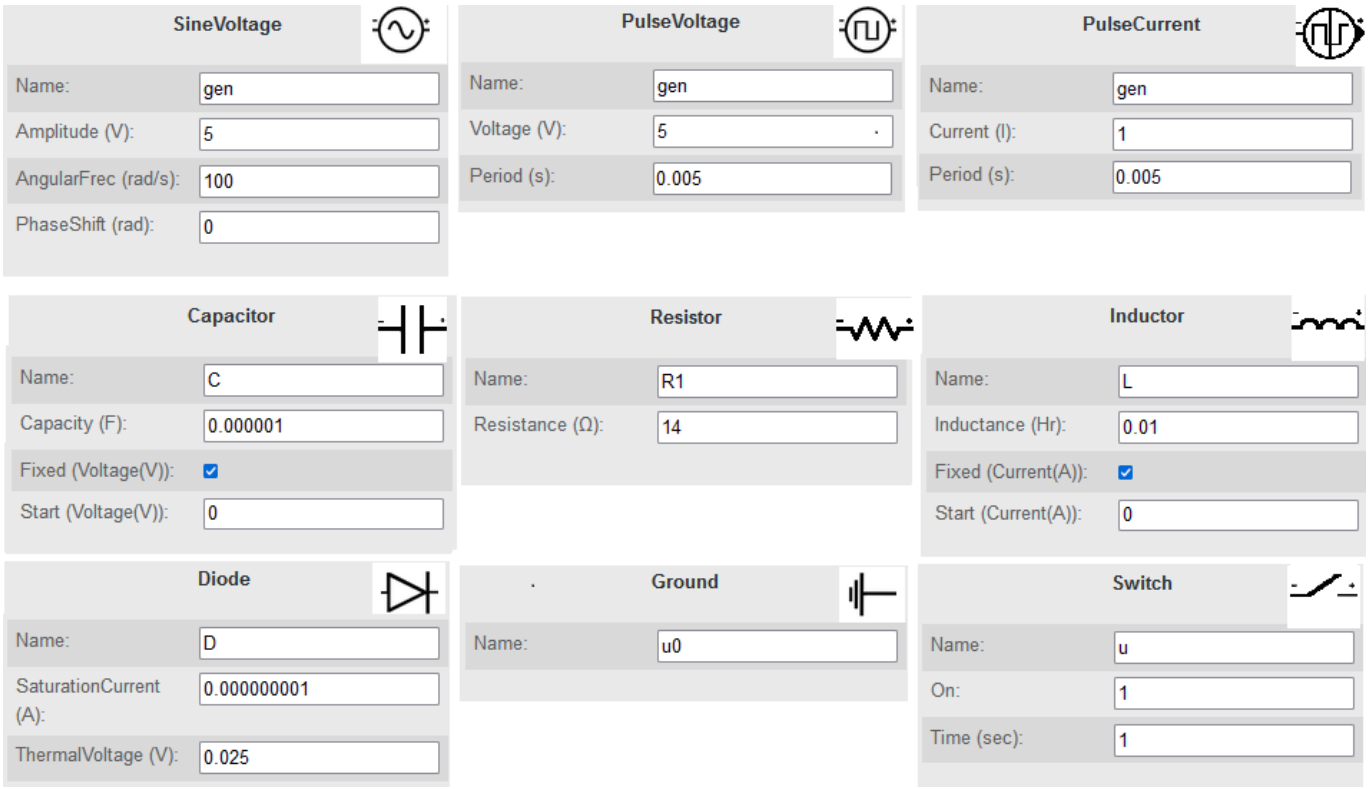


Figura 4-2 Propiedades con valores iniciales de cada componente

4.2 Modelo XML mxGraph

Hemos introducido la tecnología mxGraph en el Apartado 2.5.2, pero vamos a profundizar más sobre como lo hemos utilizado en la aplicación.

En primer lugar instanciamos la librería para usarla de esta manera

```
<script type="text/javascript" src="~/js/mxGraph/mxClient.js"></script>
```

Al cargar la página usaremos un contenedor HTML para usarla de lienzo y cargar el framework, llamaremos a este contenedor “graphContainer”

```
<body onload="main(document.getElementById('graphContainer'))">
```

La inicialización del contenedor la hace el siguiente código, primero cargando la barra de herramientas y luego el lienzo de dibujado del grafo.

```
            // Creates the div for the toolbar
            var tbContainer = document.getElementById('toolbarContainer');

            // Creates new toolbar without event processing
            var toolbar = new mxToolbar(tbContainer);
            toolbar.enabled = false

            // Workaround for Internet Explorer ignoring certain styles
            if (mxClient.IS_QUIRKS) {
                document.body.style.overflow = 'hidden';
                new mxDivResizer(tbContainer);
                new mxDivResizer(container);
                new mxDivResizer(outline);
            }
            var graph = new mxGraph(container);
            graph.view.scale = 1;
            graph.setPanning(true);
            graph.setConnectable(true);
            graph.setDisconnectOnMove(false);
            graph.foldingEnabled = false;
```

Más adelante, lo que vamos a hacer es instanciar clases concretas para cada componente eléctrico, asignando una imagen que representará en el grafo y una serie de propiedades, en el siguiente código podemos ver como se definen las clases de una resistencia o un inductor.

```
            function createResistor() {
                var doc = mxUtils.createXmlDocument();
                var values = doc.createElement('Resistor');
                values.setAttribute('name', 'R');
                values.setAttribute('resistance', '100');
                addVertex('../../Images/electric/resistor.png', 60, 60, values,
'shape=image;verticalLabelPosition=top;verticalAlign=bottom;image=../../Images/electric/r
esistor.png;');
            }


            function createInductor() {
                var doc = mxUtils.createXmlDocument();
                var values = doc.createElement('Inductor');
                values.setAttribute('name', 'L');
                values.setAttribute('inductance', '0.01');
                values.setAttribute('fixed', '1');
                addVertex('../../Images/electric/inductor.png', 60, 60, values,
'shape=image;verticalLabelPosition=top;verticalAlign=bottom;image=../../Images/electric/i
nductor.png;');
                    }
```

Esta definición de clases y la conexión de instancias entre clases da lugar a un modelo en XML similar al del código a continuación.

```
<mxGraphModel>
 <root>
    <mxCell id="0" />
    <mxCell id="1" parent="0" />
    <Resistor name="R" resistance="100" id="2">
      <mxCell style="image=../Images/resistor.png;" vertex="1" parent="1">
        <mxGeometry x="156.10" y="136.46" width="60" height="60" />
      </mxCell>
    </Resistor>
    <Inductor name="L" inductance="0.01" fixed="1" id="3">
      <mxCell style="image=../Images/inductor.png;" vertex="1" parent="1">
        <mxGeometry x="317.76" y="139.99" width="60" height="60" />
      </mxCell>
    </Inductor>
    <mxCell id="4" edge="1" parent="1" source="2" target="3">
      <mxGeometry relative="1" as="geometry" />
    </mxCell>
    <Ground name="u0" id="5">
      <mxCell style=" image=../Images/ground.png;" vertex="1" parent="1">
        <mxGeometry x="74.76" y="233.46" width="60" height="60" />
      </mxCell>
    </Ground>
    <mxCell id="6" edge="1" parent="1" source="5" target="2">
      <mxGeometry relative="1" as="geometry" />
    </mxCell>
  </root>
</mxGraphModel>
```

A simple vista parece complicado y una estructura con mucha información pero es bastante sencillo de interpretar. Cada elemento del grafo tiene un nodo con su nombre, por ejemplo, podemos observar cómo existen nodos de tipo “Ground”, “Resistor”, “Node” o “Inductor”, dentro de cada uno de los nodos podemos observar como un atributo “style” marca la imagen desde la que se renderiza y los subnodos “mxGeomentry” marcan las coordenadas en el lienzo donde se representa. Además podemos observar cómo en cada nodo tenemos las propiedades de cada elemento “resistance”, “inductance”, etc. El código explicado mostraría el modelo representado en la Figura 4-2.

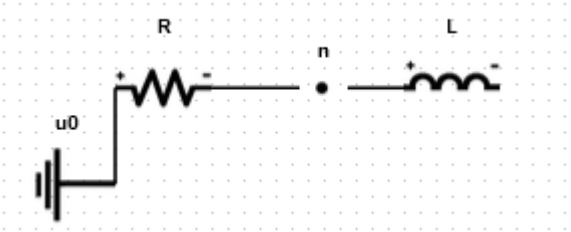


Figura 4-3 Modelo simple conectando 3 elementos

A continuación, de los elementos del grafo podemos ver las conexiones representadas con los nodos “mxCell”, los atributos “source” y “target” indican de que nodo x a que nodo y va conectado. Este modelo en XML es el que es guardado y persistido en la tabla “Circuit” en la base de datos en el campo “model”.

4.3 Modelo de clases interno

La necesidad de procesar el modelo para efectuar comprobaciones hace necesario traducir el modelo anterior a otro más homogéneo y completo. El parseo del modelo en mxGraph nos da información sobre las conexiones y propiedades de cada componente pero no tenemos el comportamiento completo. Nos hace falta incluir información adicional al objeto, además de tener una estructura más accesible en la que podamos enlazar los flujos de entrada y salida con los componentes conexos. Entre esta información adicional, necesitamos registrar en cada componente las reglas o ecuaciones que rigen su comportamiento, estas ecuaciones serán un campo más, que formarán parte de la instancia de cada objeto.

De esta manera tenemos el modelo de clases presentado en la Figura 4-3.

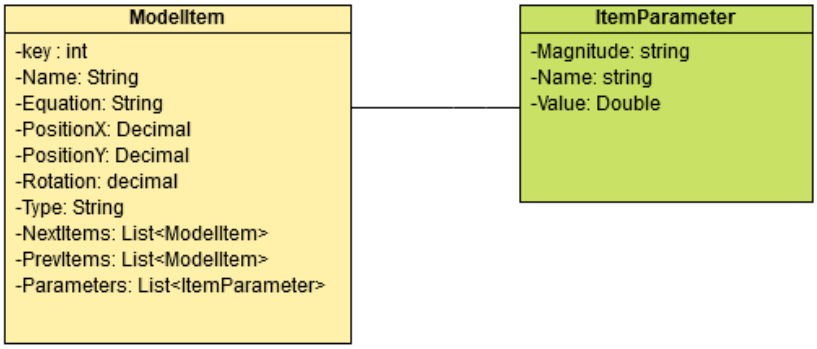


Figura 4-4 Modelo de clases interno

Suponiendo una instancia de un componente, un generador de tensión de onda sinusoidal por ejemplo podemos ver como esta estructura se rellenaría de la siguiente manera:

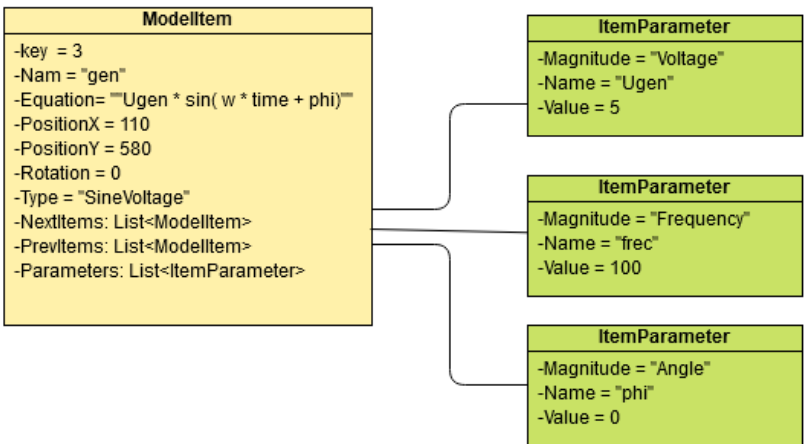


Figura 4-5 Instancia de la clase ModelItem simulando un generador sinusoidal

Obsérvese en la clase ModelItem que contamos con la propiedad “Type” donde nos indicará de que elemento estamos hablando, en función de este campo, las ecuaciones y parámetros de cada elemento serán distintas, pudiendo definir objetos nuevos a partir de la variación del campo Type y una serie de ecuaciones y parámetros asociados a cada tipo.

El campo “Equation”, es usado en la traducción al modelo atómico, cruzándolo con los distintos parámetros, de esta manera, si la ecuación del generador indica que existe un parámetro “phi”, este debe tener su debida inicialización en la lista de parámetros asociados a la clase ModelItem. En el Anexo B, apartado B-1 podemos ver el código completo del fichero ModelicaController.cs, encargado de realizar esta tarea.

4.4 Traducción a modelo atómico en Modelica

Un modelo atómico es aquel que usa un lenguaje matemático para describir otro modelo. Se define en función de ecuaciones, parámetros, algoritmos y eventos, este modelo es el que los distintos entornos de Modelica usan para realizar la compilación del ejecutable final. Aun definiendo un modelo como compuesto, OpenModelica o Dymola traducirán este a un modelo plano, con el fin de tener una relación de ecuaciones ordenadas y causalizadas. Este modelo plano, en nuestro caso cumple las reglas sintácticas de Modelica por lo que podemos decir que es una descripción atómica del modelo.

En este punto convertiremos el modelo descrito mediante instancias de componentes y sus conexiones, a un modelo equivalente consistente en declaraciones de parámetros y variables, y ecuaciones, esto es el modelo atómico. La entrada con la que contaremos para hacer esta traducción será una lista de objetos de tipo ModelItem (ver apartado anterior).

El algoritmo de traducción consta de 4 partes:

- Declarar los voltajes y corrientes
- Declarar los parámetros
- Describir las ecuaciones en los nodos
- Describir las relaciones constitutivas entre los elementos

En el Anexo B, Apartado B-2 podemos ver el código del fichero FlatService.cs, encargado de realizar la descripción del modelo atómico.

4.4.1 Declarar los voltajes y corrientes

El algoritmo sería el siguiente:

Para un componente de tipo “Node”, estos son los elementos que sirven de conexión entre dos o más elementos y los puntos del sistema eléctrico donde nos interesa realizar una medición, definiremos un nuevo voltaje. Los voltajes se nombrarán con el nombre asignado a los nodos.

```
SI.Voltage a; // “Voltaje a”
```

Para un componente distinto al anterior, y además distinto a una toma de tierra definiremos un parámetro de tipo corriente, los nombraremos como “i_” seguido del nombre asignado al componente.

```
SI.Current i_R;// “Corriente R”
```

Nótese, que usamos el “alias” SI para referenciar a la librería Modelica.Units.SI. Esta librería proporciona los tipos predefinidos basados en el estándar internacional de unidades, por lo que al inicio del código debemos definir la siguiente sentencia:

```
import SI = Modelica.Units.SI;
```

Además, es en esta parte del código donde asignamos los atributos *start* y *fixed* a los parámetros. Estos atributos asignan el valor inicial a la variable, cuando *fixed* es igual a true, el valor inicial se inicializa con el valor de *start*, cuando *fixed* es igual a false, se toma como valor inicial del proceso iterativo para la solución del modelo en el instante inicial de la simulación el valor de *start* . En la Figura 4-6 podemos ver como nuestra interfaz permite configurar el atributo fixed para algunos componentes como condensadores, diodos o inductores.

```
SI.Current i_L(start=0, fixed=true);// “Corriente L”
```

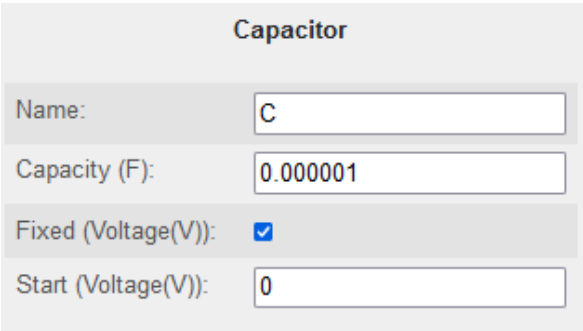


Figura 4-6 Configuración del atributo fixed

4.4.2 Declarar los parámetros

La declaración de parámetros consiste en la enumeración y asignación de valores de los distintos parámetros que intervienen en las ecuaciones asociadas a cada uno de los elementos

definidos en la lista de ModelItem. Esto es, para la instancia del ejemplo de la Figura 4-3 debemos definir qué es y qué valor tienen “phi”, “w” , “U” y “frec”. Al declarar las variables, estás se nombran usando el nombre del componente al que van asociadas.

```
// "Parámetros del generador gen"
parameter SI.Voltage Ugen = 5;
parameter SI.Frequency frec_gen = 100;
parameter SI.AngularFrequency w_gen = 2 * Modelica.Constants.pi * frec_gen;
parameter SI.Angle phi_gen = 0;
```

Cada tipo de componente declarará una serie de parámetros distintos, todo ello va en relación con las ecuaciones que gobiernan el componente. Es en este punto donde se usa la lista de parámetros asociados a la entidad ModelItem. En la Figura 4-6, podemos ver un ejemplo de declaración de parámetros del condensador, donde además del nombre de componente y el atributo fixed, especificamos la capacidad en faradios, siempre especificaremos el símbolo o la abreviación de la magnitud al lado de cada parámetro. En el caso de los componentes resistencia, condensador e inducción, el nombre del parámetro en el modelo atómico coincide con el nombre que el usuario ha dado al componente al definir el diagrama del circuito

Ponemos el ejemplo de una resistencia, un inductor y un condensador.

```
// "Parámetros de la Resistencia R"
parameter SI.Resistance R = 15;
// "Parámetros del inductor L"
parameter SI.Inductance L = 0.1;
// "Parámetros del condensador C"
parameter SI.Capacitance C = 1e-06;
```

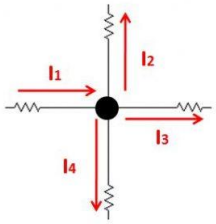
4.4.3 Describir las ecuaciones en los nodos

La conservación de la energía es un principio fundamental en la física y se aplica a todos los sistemas, incluyendo los sistemas eléctricos. En un sistema eléctrico, la energía se puede transferir de una forma a otra, pero la cantidad total de energía presente en el sistema se mantiene constante. Por ejemplo, en un circuito eléctrico, la energía eléctrica se puede transformar en calor, luz o movimiento mecánico. Sin embargo, la cantidad total de energía presente en el circuito no cambia. Si se mide la cantidad de energía que entra al circuito a través de la fuente de alimentación y se compara con la cantidad de energía que sale del circuito en forma de calor, luz o movimiento mecánico, se encontrará que son iguales. A la hora de analizar este concepto en el dominio eléctrico es obligatorio mencionar las leyes de Kirchoff.

Las leyes de Kirchhoff son un conjunto de principios fundamentales utilizados en la electrónica y la electricidad para analizar y diseñar circuitos eléctricos. Estas leyes se basan en la

conservación de la energía y la carga en un circuito eléctrico y se utilizan para calcular las corrientes y tensiones en diferentes puntos del circuito.

Hay dos leyes de Kirchhoff, la ley de corriente y la ley de voltaje. La ley de corriente de Kirchhoff establece que la suma de las corrientes que entran en un nodo (un punto en el circuito donde se encuentran varias ramas) es igual a la suma de las corrientes que salen del nodo. Esta ley se aplica a los nodos y se conoce como la "primera ley de Kirchhoff".



$$I_1 = I_2 + I_3 + I_4$$

Figura 4-7 Ejemplo de la primera Ley de Kirchhoff

La ley de voltaje de Kirchhoff establece que la suma de los voltajes en un circuito cerrado es igual a cero. Esta ley se aplica a los bucles o caminos en el circuito y se conoce como la "segunda ley de Kirchhoff".

Vamos a suponer un circuito eléctrico en el que tenemos una serie de componentes, resistencias, condensadores, diodos, etc. Cada componente se conecta con otro mediante pines de conexión, a su vez, tenemos un componente que denominamos "nodo", el cual será receptor de varias conexiones pero no tendrá ningún comportamiento tal que modifique alguna magnitud del circuito, a este componente lo identificamos como *Type* "node" y se visualiza como un punto al cual se le puede asignar una etiqueta y conexiones.

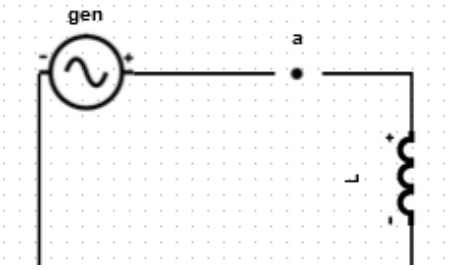


Figura 4-8 Nodo con nombre "a" conectando un generador con un inductor

Este nodo tiene una corriente entrante y una saliente por lo que debemos establecer esta igualdad en forma de ecuación.

El algoritmo desarrollado comprobará las corrientes entrantes y salientes en el nodo. En este caso, poniendo como ejemplo la Figura 4-8, tenemos que la corriente en el generador debe ser igual que la corriente medida en el inductor.

```
// Ecuaciones en los nodos
i_L = i_gen;
```

Si tenemos más de un componente conectado, se considera siempre que la suma de las corrientes de entrada en los distintos componentes debe ser igual a la suma de las corrientes de salida. En el caso de la Figura 4-9 tenemos una resistencia (R1) a la izquierda y otros 3 componentes a la derecha, de esta manera la ecuación resultante debe ser:

```
i_R2 + i_C + i_L = i_R1;
```

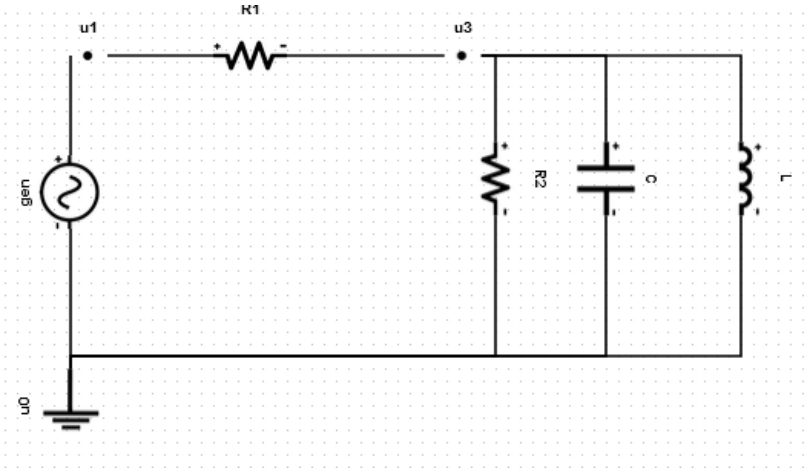


Figura 4-9 Circuito modelado con varios elementos en paralelo

Este principio hace que Modelica tenga que hacer una distinción entre variables de tipo *across* y tipo *through*. Las variables tipo across o también denominadas variables de esfuerzo son aquellas que valen lo mismo en el punto de conexión, en este caso, el voltaje se considera de este tipo. Las variables de tipo through o variables de flujo son aquellas cuya suma vale 0 en el punto de conexión, el ejemplo que hemos puesto en la Figura 4-9 con la corriente en el nodo “u3” es un ejemplo de variable through.

Otro factor para tener en cuenta, nuestro algoritmo usa los nodos del circuito para establecer las relaciones entre los componentes conectados a ese nodo. Es por ello por lo que se implementó una regla por la cual siempre debe existir un nodo entre cada componente. Aunque en el mundo real si es posible conectar componentes entre sí, nuestra herramienta mantiene esta regla por cuestiones de facilitar los cálculos de corriente y voltajes e interpretar correctamente el grafo y sobre todo cada nudo, donde varias conexiones se pueden hacer entre pines diferentes. Esta restricción se contempla por ejemplo en SPICE, una herramienta de simulación de circuitos integrados donde su algoritmo también se centra en análisis en los nodos.

4.4.4 Describir las relaciones constitutivas

La última parte del cálculo del modelo atómico implica usar las ecuaciones que definen cada elemento. En el punto 4.3 hablamos de los campos que almacenábamos en las clases ModelItem y pusimos como ejemplo una fuente de corriente sinusoidal, vimos como almacenábamos su ecuación en la instancia del elemento, pues bien, es en este proceso donde debemos sustituirlas en los voltajes que calculamos en cada elemento.

Por ejemplo, en un modelo de un circuito eléctrico, las relaciones constitutivas pueden incluir la ley de Ohm, que establece que la corriente eléctrica que fluye a través de un conductor es directamente proporcional a la diferencia de potencial eléctrico (voltaje) aplicada y es inversamente proporcional a la resistencia del conductor. Esta ley constituye una relación matemática entre la corriente, el voltaje y la resistencia, y se puede utilizar para predecir cómo se comportará el circuito en función de estos parámetros.

En el apartado anterior hemos hablado sobre las variables de tipo across, en este caso, el voltaje debemos definirla de esta manera, por lo que, la diferencia de voltajes en un lado y en otro de cualquier componente debe ser el voltaje resultante en ese componente. Volvamos a la Figura 4-9, si vemos la resistencia R1, el voltaje en ese punto debe ser u1 – u3, y esto lo expresaremos de esta forma en Modelica.

```
// Relaciones constitutivas
u1 - u3 = i_R1 * R1;
```

Hemos sustituido el voltaje en la resistencia por su ecuación (U = I . R), de esta manera implicamos las variables que hemos definido inicialmente de resistencia y de corriente de cada elemento.

Si juntamos estos 4 procesos descritos para obtener el modelo atómico, obtendríamos el siguiente código para el circuito definido en la Figura 4-9.

```
model Circuit8Flat
import SI = Modelica.Units.SI;
  SI.Current i_gen;// " Corriente gen"
  SI.Current i_R1;// " Corriente R1"
  SI.Current i_R2;// " Corriente R2"
  SI.Current i_L(start=0, fixed=true);// " Corriente L"
  SI.Current i_C;
  SI.Voltage u_C(start=0, fixed=true);// "Voltaje C"
  SI.Voltage u1;// "Voltaje u1"
  SI.Voltage u3;// "Voltaje u3"
// "Parámetros del generador gen"
  parameter SI.Voltage Ugen = 5;
  parameter SI.Frequency frec_gen = 100;
  parameter SI.AngularFrequency w_gen = 2 * Modelica.Constants.pi * frec_gen;
  parameter SI.Angle phi_gen = 0;
// "Parámetros de la Resistencia R1"
  parameter SI.Resistance R1 = 100;
// "Parámetros de la Resistencia R2"
  parameter SI.Resistance R2 = 100;
// "Parámetros del inductor L"
  parameter SI.Inductance L = 0.01;
// "Parámetros del condensador C"
  parameter SI.Capacitance C = 1e-06;

equation
// Ecuaciones en los nodos
  i_R1 = i_gen;
  i_C + i_L + i_R2 = i_R1;
// Relaciones constitutivas
  u1 = Ugen * sin( w_gen * time + phi_gen);
  u1 - u3 = i_R1 * R1;
  u3 = i_R2 * R2;
  u3 = L * der(i_L);
  i_C = C * der( u_C );
  u3 = u_C;
end Circuit8Flat;
```

4.5 Generación del modelo compuesto en Modelica

Un modelo compuesto es un modelo basado en clases siguiendo el paradigma de la programación orientada a objetos. Cada componente estará definido en un "contenedor" en el cual estarán declaradas una serie de propiedades y comportamientos. A su vez, cada clase tiene una interfaz accesible para poder declarar sus parámetros.

Por poner un ejemplo, en la librería standard de Modelica podemos ver como se define una resistencia con el siguiente código, observamos como implícitamente existen las variables que declaran la resistencia, la temperatura o el coeficiente lineal de temperatura.

```
within Modelica.Electrical.Analog.Basic;
model Resistor “Ideal linear electrical resistor”
  parameter SI.Resistance R(start=1)
    “Resistance at temperature T_ref”;
  parameter SI.Temperature T_ref=300.15 “Reference temperature”;
  parameter SI.LinearTemperatureCoefficient 65lpha=0
    “Temperature coefficient of resistance (R_actual = R*(1 + 65lpha*(T_heatPort -T_ref))”;

  extends Modelica.Electrical.Analog.Interfaces.OnePort;
  extends Modelica.Electrical.Analog.Interfaces.ConditionalHeatPort(T=T_ref);
  SI.Resistance R_actual
    “Actual resistance = R*(1 + 65lpha*(T_heatPort – T_ref))”;

equation
  assert((1 + 65lpha*(T_heatPort – T_ref)) >= Modelica.Constants.eps,
    “Temperature outside scope of model!”);
  R_actual = R*(1 + 65lpha*(T_heatPort – T_ref));
  v = R_actual*i;
  LossPower = v*i;

end Resistor;
```

El comportamiento del objeto se define en la sección “equation” donde podemos ver como usa las variables declaradas y calcula el voltaje como intensidad por resistencia. Este modelo hace que podamos reutilizarlo en un sistema más complejo, de la siguiente forma:

```
Analog.Basic.Resistor R(R = 15)
annotation(Placement(visible = true, transformation(
      origin = { 433, 627},
      extent = { { -20, -20}, { 20, 20} },
      rotation = 0)));
```

Nótese que las cláusulas “annotation” definen su situación en un plano bidimensional para establecer una representación gráfica.

Además de la definición del componente debemos integrarlo o conectarlo con otros componentes, para ello se extiende la interfaz “OnePort”, la cual dota l componente de dos pines, uno positivo y otro negativo. Usaremos estos pines para realizar la conexión con otros componentes con la sentencia “connect”.

```
Connect(b, R.p) annotation (Line(points ={{354,685},{433,627}}, color ={ 0,0,255}));
connect(R.n, c) annotation (Line(points ={{433,627},{555,659}}, color ={ 0,0,255}));
```

Una vez sabemos cómo funcionan los modelos predefinidos de las librerías de Modelica, podemos traducir nuestro modelo gráfico en nuestra aplicación a un modelo compuesto en Modelica. En el Anexo B, Apartado B-3 mostramos el código del fichero CompositeService.cs, encargado de realizar todo el proceso descrito en este apartado.

El siguiente ejemplo sería una muestra de código basado en la Figura 4-8

```
model Circuit8Composite
import Analog = Modelica.Electrical.Analog;
Analog.Sources.SineVoltage gen(V = 5, offset = 0 , f = 100)
 annotation(Placement(visible = true, transformation(
        origin = { 104, 644},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -90)));

Analog.Basic.Resistor R1(R = 100)
 annotation(Placement(visible = true, transformation(
        origin = { 250, 755},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Basic.Resistor R2(R = 100)
 annotation(Placement(visible = true, transformation(
        origin = { 428, 655},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -270)));

Analog.Basic.Inductor L(L = 0.01)
 annotation(Placement(visible = true, transformation(
        origin = { 650, 655},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 90)));

Analog.Basic.Capacitor C(C = 1E-06)
 annotation(Placement(visible = true, transformation(
        origin = { 540, 655},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 90)));

Analog.Basic.Ground u0
 annotation(Placement(visible = true, transformation(
        origin = { 84, 470},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -90)));

Analog.Interfaces.NegativePin u1
 annotation(Placement(transformation(extent ={ { 94,745},{ 114,765} })));

Analog.Interfaces.NegativePin u3
```

```
43  annotation(Placement(transformation(extent ={ { 413,745},{ 433,765} })));
44
45
46 equation
47 connect(u0.p, gen.n) annotation (Line(points ={{104,470},{104,644}}, color
   ={ 0,0,255}));
   connect(gen.p, u1) annotation (Line(points
48 ={{104,644},{119,755},{119,755}}, color ={ 0,0,255}));
   connect(u1, R1.p) annotation (Line(points ={{119,755}}, color ={
   0,0,255}));
   connect(R1.n, u3) annotation (Line(points ={{250,755},{423,755}}, color ={
49 0,0,255}));
   connect(u3, R2.p) annotation (Line(points ={{423,755},{450,655}}, color ={
   0,0,255}));
   connect(R2.n, u0.p) annotation (Line(points
50 ={{450,655},{450,470},{104,470}}, color ={ 0,0,255}));
   connect(u3, C.p) annotation (Line(points ={{423,755},{540,655}}, color ={
51 0,0,255}));
   connect(C.n, u0.p) annotation (Line(points
   ={{540,655},{540,470},{104,470}}, color ={ 0,0,255}));
52 connect(u3, L.p) annotation (Line(points ={{423,755},{650,655}}, color ={
   0,0,255}));
   connect(L.n, u0.p) annotation (Line(points
53 ={{650,655},{650,470},{104,470}}, color ={ 0,0,255}));


54 annotation(
55         uses(Modelica(version = "4.0.0")),
56         Diagram(coordinateSystem(extent = { { 0, 0}, { 1400, 1400} })),
57         Icon(coordinateSystem(extent = {{0, 0}, {1400, 1400}})),
58         version = "");
59
60 end Circuit8Composite;
```

La traducción a un modelo compuesto resulta mucho más fácil cuando los objetos con los que trabajas están bien estructurados y tienen un acceso sencillo a los nodos conexos, es por ello por lo que el uso del modelo de clases interno ha sido una buena decisión a la hora de realizar los algoritmos de traducción.

Hemos comentado, que guardamos en las anotaciones información sobre las coordenadas y tamaño de los objetos, pues bien, esta información es de gran ayuda en el código ya que nos permite ver gráficamente el mismo modelo que estamos viendo nuestra aplicación en un entorno de simulación como OpenModelica.

Figura 4-10 Iconos de tipos de representación en OpenModelica

En la Figura 4-10 tenemos los iconos de OpenModelica donde se puede cambiar el tipo de representación del modelo a la vista por defecto que sería en código a una vista basada en diagramas gráficos. En el caso del modelo que estamos estudiando la vista en modo diagrama quedaría como se puede ver en la Figura 4-11

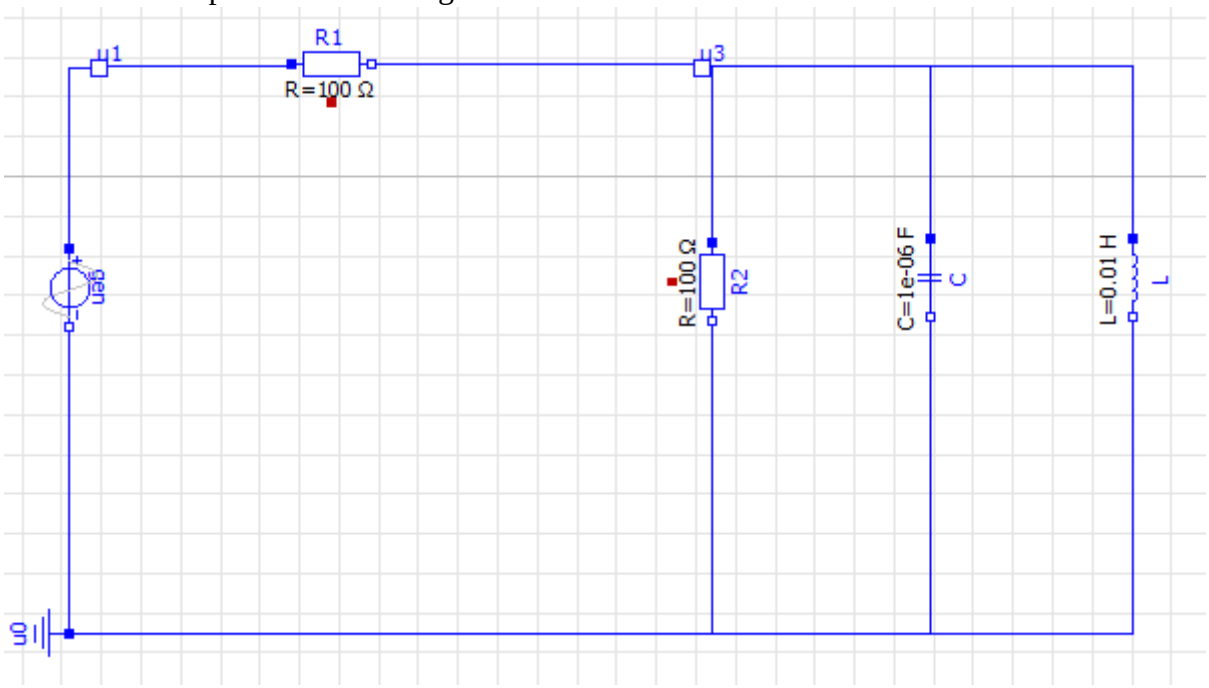


Figura 4-11 Representación gráfica de un modelo en OpenModelica

4.6 Comprobaciones sobre el modelo

Para comprobar que el modelo generado sea correcto y su traducción no ocasione ningún problema de sintaxis ni de compilación a la hora de realizar la simulaciones es necesario establecer unas reglas previas. Enumeraremos los mensajes de comprobación que pueden mostrarse en la aplicación y los escenarios que llevan a provocar estos mensajes.

Código	Mensaje
01	A ground connection is necessary for the model to be valid
02	Only one ground node can be declared
03	A component can only be connected to one node or to ground
04	At least one node is necessary for the model to be valid
05	There cannot be two consecutive nodes
06	There cannot be two elements with the same name
07	All elements must be connected for the model to be valid
08	Cannot connect current sources in series
09	Cannot connect voltage sources in parallel
10	A node cannot be connected to the ground node

Tabla 4-2 Mensajes de comprobación

01- Todos los modelos deben tener al menos una conexión a tierra

En teoría de circuitos, el voltaje se refiere a la diferencia de potencial eléctrico entre dos puntos. Por lo tanto, se necesita un punto de referencia para medir el voltaje en un circuito. Este punto de referencia se conoce como el punto de masa o punto de tierra, y se utiliza como el punto de referencia para todas las mediciones de voltaje en el circuito.

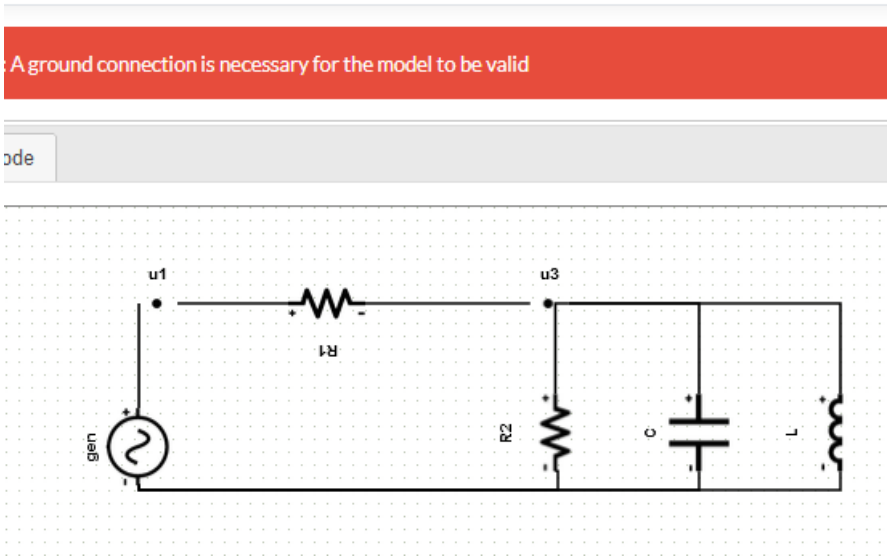


Figura 4-12 Comprobación 01

02- Solo puede existir un único nodo tierra

Por simplicidad de cálculos de conexiones y resolución de ecuaciones hemos añadido esta condición. El nodo tierra siempre marca un voltaje igual a cero, de cara a simplificar y reducir problemas en las simulaciones se decide añadir esta comprobación.

03- Los componentes solo pueden conectarse a un nodo o a tierra

Los componentes solo deben conectarse a dos nodos, o a un nodo y a tierra, esta restricción es necesaria para reducir el número de variaciones que pueden hacer que el grafo resultante del diseño en la interfaz de usuario derive en casos que habría que tratar individualmente.

Por ejemplo, imaginemos que en vez de conectar el nodo tierra a dos componentes, conectamos el nodo tierra al pin negativo de uno de ellos, y a ese mismo pin conectamos el otro componente. Habría que identificar este segundo componente como derivado a tierra.

Otra casuística podría ser en la confluencia de dos componentes en un nodo, pero en este caso, uno de los componentes A se conecta al pin del otro componente B que va conectado al nodo. El componente A habría que identificarlo como conectado al nodo para calcular la corriente en los nodos

04- Al menos necesitamos un nodo en el modelo

El algoritmo de traducción al modelo atómico hace uso de estos nodos para establecer las relaciones entre los demás componentes, es necesaria esta regla para poder desarrollar las ecuaciones en los nodos dentro del modelo.

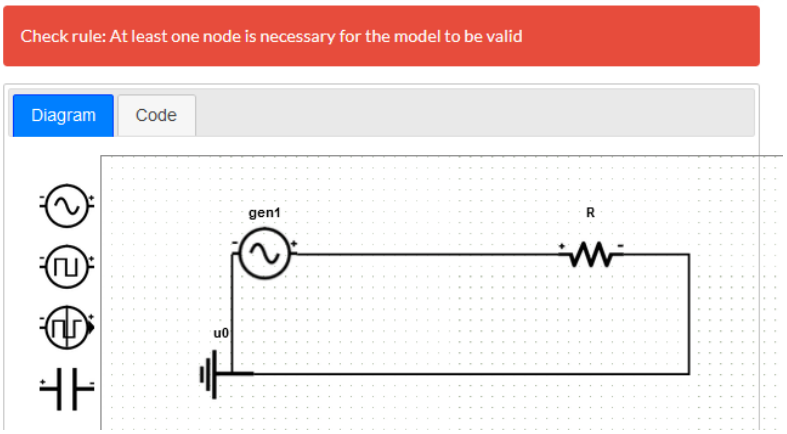


Figura 4-13 Comprobación 04

Un modelo atómico se puede representar como un conjunto de nodos y ramas, donde cada nodo representa una variable independiente y cada rama representa una ecuación que relaciona dos o más variables. Los nodos se utilizan para establecer las relaciones entre los demás

componentes del sistema y para desarrollar las ecuaciones que describen el comportamiento del sistema.

05- No pueden existir dos nodos consecutivos

Al igual que en la segunda comprobación, por cuestiones de simplicidad de los modelos y por reducción de problemas, optamos por poner esta regla.

06- No pueden existir dos componentes con el mismo nombre

Es importante asegurarse de que no existan dos componentes con el mismo nombre en un modelo. Esto se debe a que cada componente debe tener un nombre único para poder identificarlo de manera clara y precisa en el modelo. Si dos componentes tienen el mismo nombre, puede haber confusiones y errores en la representación y en la simulación del sistema.

Cuando hablamos de componentes, no nos referimos a los componentes eléctricos únicamente, también se incluyen los nodos y la tierra.

Además, es posible que algunos programas de simulación no permitan la existencia de dos componentes con el mismo nombre en un modelo. En este caso, el programa podría generar un error o una advertencia cuando se cargue el modelo. En el caso de la Figura 4-14, podemos ver como al nombrar dos componentes con el mismo nombre, ya nos recalca un error en el que nos avisa que no es un escenario posible y por lo tanto no se puede compilar el modelo.

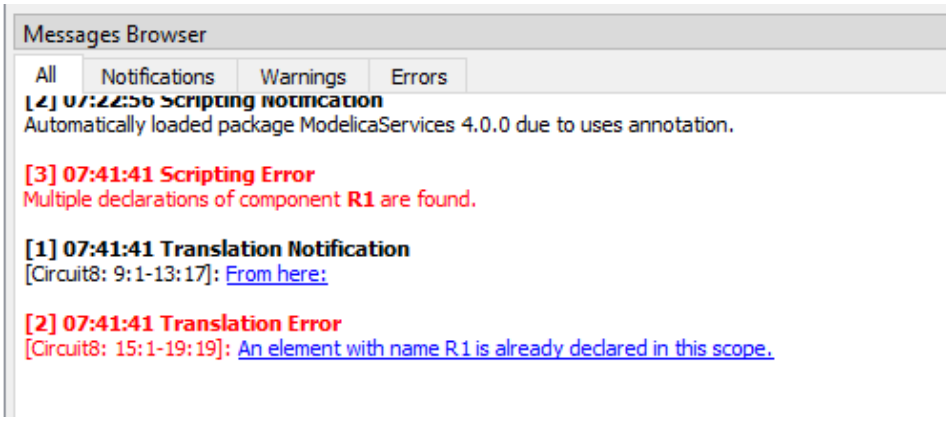


Figura 4-14 Mensajes de comprobación de OpenModelica al escribir dos resistencias con el mismo nombre

Para evitar estos problemas, es importante asegurarse de que cada componente del sistema tenga un nombre único y distintivo. Esto puede implicar asignar nombres específicos a cada componente o utilizar códigos de identificación únicos. De esta manera, se puede garantizar que cada componente del sistema se represente de manera precisa y se pueda identificar de manera clara y sencilla en el modelo. En la Figura 4-15 podemos ver cómo se puede editar el nombre de un componente mediante una de sus propiedades

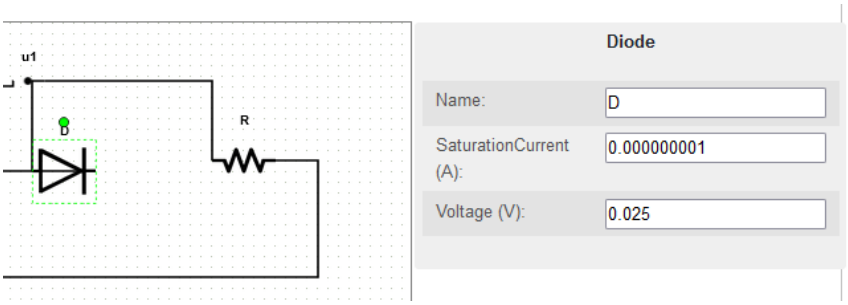


Figura 4-15 Panel de propiedades donde se asigna el nombre a cada elemento

07- Todos los elementos deben estar conectados

En un modelo de sistema eléctrico, es importante asegurarse de que todos los elementos estén correctamente conectados. Añadimos esta regla para ayudar al usuario a evitar que deje ningún pin sin conectar debido a algún posible despiste. Cuando en OpenModelica o en Dymola dejamos un pin sin conectar se añade una ecuación por defecto al modelo en la que la corriente en ese punto es cero. En algunas ocasiones está ecuación hace que el modelo sea correcto ya que equipara el número de ecuaciones al de incógnitas pero en otros, el modelo se convierte en singular, es decir, distinto número de ecuaciones y de incógnitas.

Si hay componentes del sistema que no están conectados de manera adecuada, aunque el modelo compile correctamente, la simulación no será correcta. Por lo tanto, es importante asegurarse de que todos los componentes del sistema estén correctamente conectados y relacionados en el modelo.

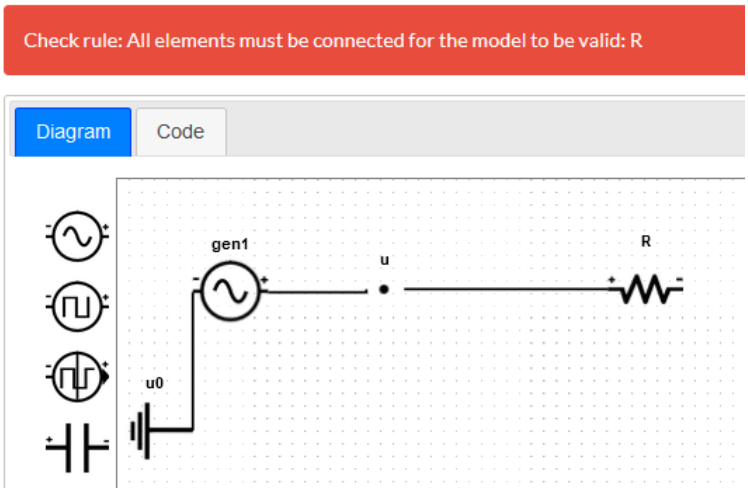


Figura 4-16 Comprobación 06, donde vemos como falta una conexión en la resistencia R

08- No se pueden conectar fuentes de corriente en serie

Conectar dos fuentes de corriente en serie, supondría que aportaría cada una de ellas una ecuación para el cálculo de la corriente en la misma rama. Es importante tener en cuenta que las corrientes deben medirse en la misma dirección, es decir, deben ser coherentes, cada componente de esa rama debe tener definida una única corriente.

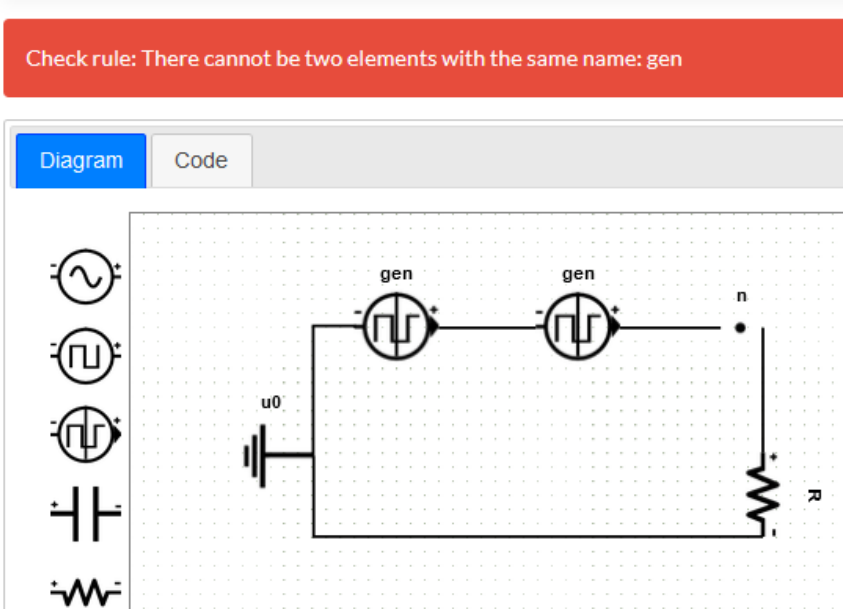


Figura 4-17 Comprobación 08, fuentes de corriente en serie

09- No se pueden conectar dos fuentes de tensión en paralelo

Si enunciamos la segunda ley de Kirchhoff, podemos decir que “En un circuito cerrado, la suma de todas las caídas de tensión es igual a la tensión total suministrada. De forma equivalente, la suma algebraica de las diferencias de potencial eléctrico en un circuito es igual a cero.”, lo que viene representado por la siguiente fórmula:

$$\sum_{k=1}^{n} V_k = V_1 + V_2 + V_3 \cdots + V_n = 0$$

Para explicarlo mejor, partamos del siguiente modelo en la Figura 4-18, donde tenemos dos fuentes de voltaje en paralelo. Si calculamos el voltaje total tendríamos que:

$$V_1 + V_2 = 0$$

Fuentes de voltaje distintas hacen que esta igualdad no se cumpla dando lugar a un sistema singular.

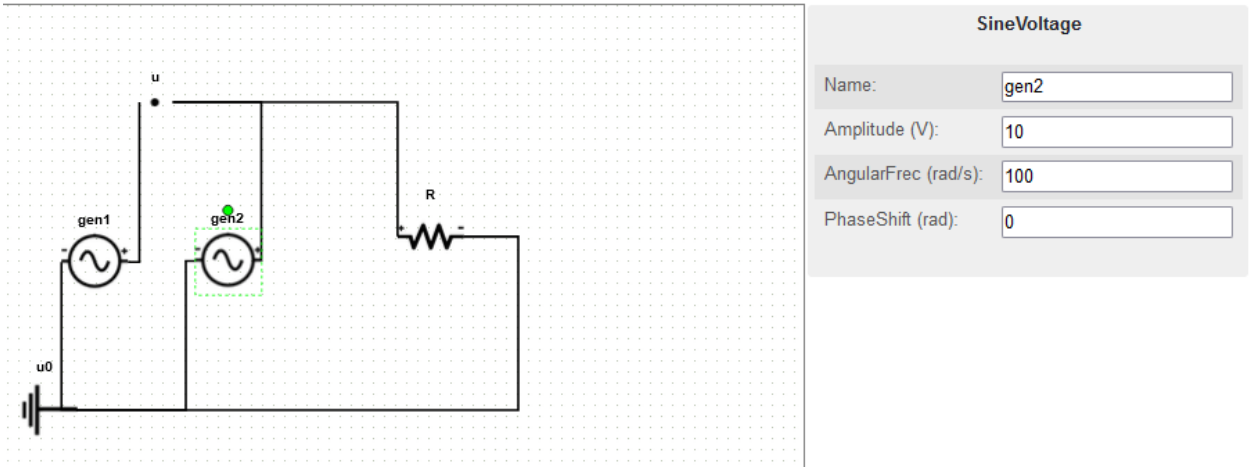


Figura 4-18 Fuentes de tensión en paralelo

10 – No se puede conectar un nodo a tierra

Esta restricción hace que se simplifique el modelo, evitando que la ecuación en los nodos conectados a tierra sea redundante. Tampoco tiene sentido ya que el voltaje en ese nodo siempre sería igual a 0.

4.7 Conclusiones

En este capítulo hemos hecho un recorrido para ver cómo se tratan los modelos desde que se dibujan en la interfaz gráfica hasta como se traducen en código Modelica.

- El modelo de mxGraph nos da una estructura XML, es un modelo cerrado del propio framework y nos permite interactuar con los objetos gráficamente.
- El modelo interno es una traducción a clases de C# desde el modelo mxGraph para dar mayor claridad a los algoritmos de traducción y poder desarrollar con un código limpio las comprobaciones del sistema.
- La traducción a un modelo atómico, la basamos en las ecuaciones y parámetros de cada componente, teniendo en cuenta las conexiones y los parámetros de voltaje y corriente que se calculan en cada nodo.
- La traducción al modelo compuesto tiene en cuenta una estructura de clases y objetos que se reutilizan para conectarse entre sí y dar lugar al sistema final que queremos simular.

5 Construcción del frontend

5.1 Introducción

Una interfaz gráfica de usuario (GUI, por sus siglas en inglés) es un tipo de interfaz que permite a los usuarios interactuar con un aplicativo mediante elementos gráficos y visuales en lugar de comandos de texto. Una GUI se compone de diversos elementos gráficos, como botones, menús, iconos y ventanas, que permiten a los usuarios ejecutar tareas y acceder a información de manera más intuitiva y sencilla.

En nuestro caso, la interfaz gráfica nos va a proporcionar una serie de herramientas para trabajar con la aplicación y poder definir los modelos correctamente, en este capítulo vamos a definir como es la interacción del usuario con la aplicación y como se va a relacionar con la interfaz para conseguir el objetivo principal, que es generar código Modelica. Nos centraremos en el editor de modelos, para ver el funcionamiento integral de la aplicación se puede consultar el Anexo A.

5.2 Lienzo y propiedades del entorno

La pantalla que compone el editor principal se estructura de la siguiente manera:

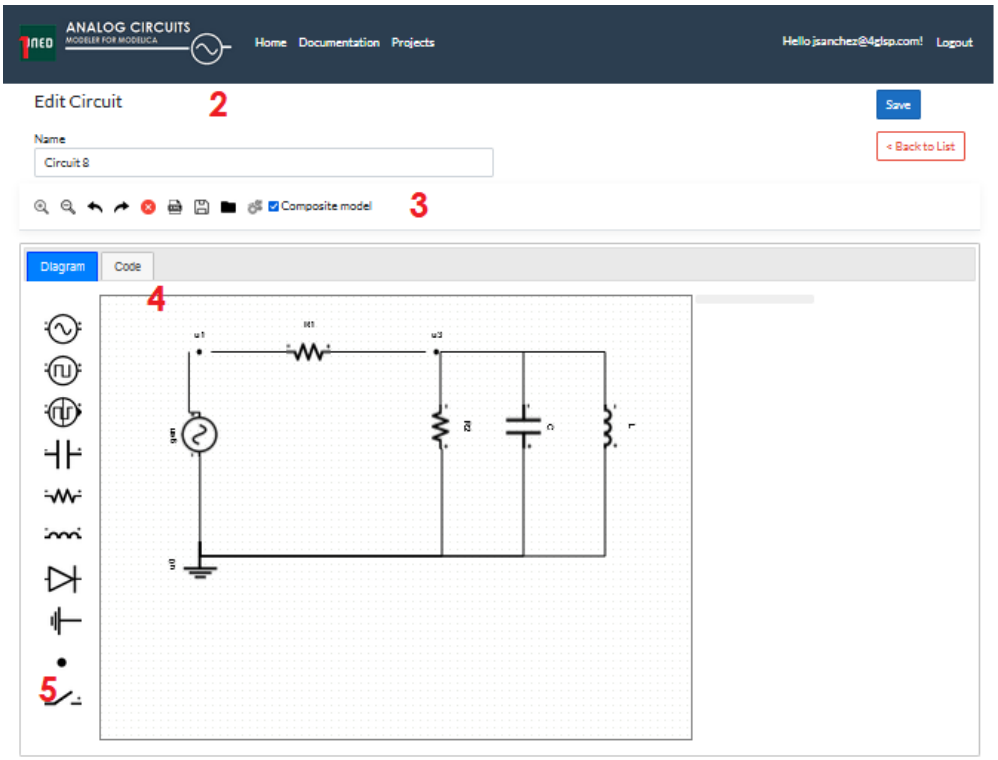


Figura 5-1 Aspecto general del editor dentro de la aplicación.

1) Una barra de encabezado superior donde podemos navegar a otras opciones de la aplicación, tales como cerrar la sesión del login actual, la pantalla de bienvenida de la aplicación o la jerarquía de proyectos del usuario.
2) Un cuadro de texto donde podemos editar el nombre del modelo, con la posibilidad de guardar los cambios o volver a la página de proyectos.
3) Inmediatamente debajo tenemos una barra de herramientas para trabajar con los elementos del modelo. En esta barra podremos ver el XML de mxGraph que forma el modelo, cargar un modelo externo, hacer o deshacer cambios, etc.
4) Tenemos un lienzo donde visualizaremos el modelo, con una segunda pestaña donde podremos visualizar el código generado.
5) Una paleta de elementos que se podrán arrastrar al lienzo, los componentes eléctricos con los que vamos a trabajar.

5.3 Definición de paleta de herramientas

La paleta de herramientas se define en JavaScript con imágenes en formato .png y está compuesta por los elementos del dominio eléctrico que usaremos para realizar nuestros modelos.

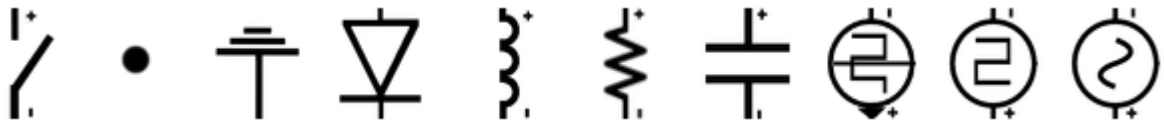


Figura 5-2 Paleta de componentes del editor

De izquierda a derecha son los siguiente: interruptor, nodo, toma de tierra, diodo, inductor, resistencia, condensador, fuente de corriente, fuente de tensión de onda cuadrada y fuente de tensión sinusoidal.

Estos elementos de la paleta se podrán seleccionar y arrastrar al lienzo para formar el modelo deseado.

5.4 Propiedades de los elementos

Cada vez que seleccionamos un componente dentro de nuestro modelo, se habilitará una ventana de propiedades a la derecha del lienzo, esta ventana permitirá definir los valores de los

parámetros que gobiernan cada elemento. Como por ejemplo, en la Figura 5-3, vemos que al seleccionar un diodo, se permite configurar la corriente de saturación, el voltaje y el atributo fixed.

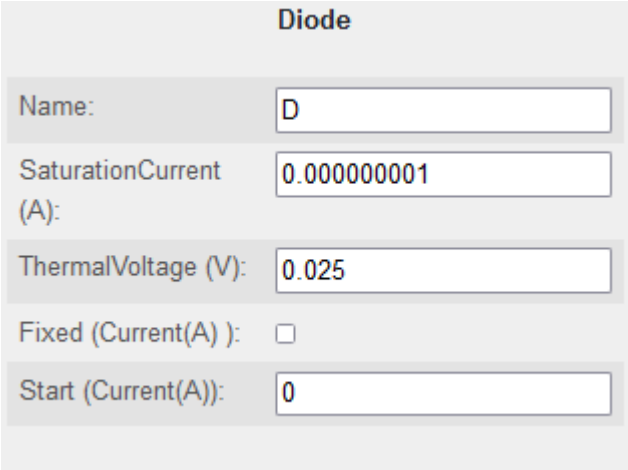


Figura 5-3 Ventana de propiedades de componente

Esta ventana de propiedades será distinta para cada componente según sus parámetros pero hay un elemento común en todos los elementos. La propiedad "Name" debe ser única para cada componente para que el modelo sea correcto, de hecho es la comprobación implementada con código 05 ya comentada en el capítulo anterior.

Para cada parámetro que da valor a una magnitud concreta se define entre paréntesis la magnitud de la que hablamos, por ejemplo si estamos definiendo un voltaje veremos "V" entre paréntesis, si estamos definiendo una resistencia "Ω".

En nuestro código JavaScript definiremos el siguiente bloque para definir la tabla de propiedades, al seleccionar un componente, se recorrerán todos sus atributos creando un campo de formulario para definir sus valores.

```
// Creates the form from the attributes of the user object
var form = new mxForm();
var attrs = cell.value.attributes;
for (var i = 0; i < attrs.length; i++) {
    createTextField(graph, form, cell, attrs[i]);
}
```

La función que crea cada entrada en el formulario explora de que tipo de magnitud es, asignando la unidad correspondiente y creando el elemento HTML input dinámicamente para que el usuario pueda introducir el valor.

```
function createTextField(graph, form, cell, attribute) {
                var magnitude = "";
                if (attribute.nodeName == "current") { magnitude = " (I)"; }
                if (attribute.nodeName == "voltage") { magnitude = " (V)"; }
                if (attribute.nodeName == "frequency") { magnitude = " (Hz)"; }
                if (attribute.nodeName == "angularFrec") { magnitude = " (rad/s)"; }
                if (attribute.nodeName == "phaseShift") { magnitude = " (rad)"; }
                if (attribute.nodeName == "time") { magnitude = " (sec)"; }
                if (attribute.nodeName == "resistance") { magnitude = " (Ω)"; }
                if (attribute.nodeName == "inductance") { magnitude = " (Hr)"; }
                if (attribute.nodeName == "capacity") { magnitude = " (F)"; }
                if (attribute.nodeName == "amplitude") { magnitude = " (V)"; }
                if (attribute.nodeName == "saturationCurrent") { magnitude = " (A)"; }

                var input;
                if (name == "Fixed") {
                    var isTrueSet = parseInt(attribute.nodeValue);

                    input = form.addCheckbox(name + magnitude + ':', isTrueSet);
                } else {
                    input = form.addText(name + magnitude + ':', attribute.nodeValue);
                       }
```

5.5 Barra de herramientas

En el contenedor superior al lienzo hay una barra de herramientas con funcionalidades varias. De izquierda a derecha explicamos que función realiza cada icono.

Figura 5-4 Barra de funciones

1- Aumentar zoom: aumenta el rango de visión dentro del lienzo.
2- Reducir zoom: aleja la perspectiva sobre el lienzo.
3- Deshacer: deshace las últimas modificaciones.
4- Rehacer: rehace sobre el historial de modificaciones.
5- Eliminar: elimina el elemento seleccionado.
6- Visualizar modelo: visualiza el modelo mxGraph en formato XML en un cuadro de dialogo modal.
7- Descarga modelo: descarga el modelo mxGraph en formato XML
8- Carga modelo: Carga un fichero xml que contiene un modelo en mxGraph, esta opción es muy útil para compartir modelos entre distintos usuarios.
9- Procesa el modelo: realiza las traducciones a código Modelica. Según si está marcado el check "Composite model" realizará la traducción al modelo compuesto, si no, realizará la traducción al modelo atómico.

5.6 Editor de código

En la pestaña “Code” definimos un editor de código, con un plugin de terceros llamado “edit_Area” licenciado bajo Apache License 2.0. Este será el componente que muestre el código Modelica, se ha elegido este componente porque permite el visualizado de código interpretando la sintaxis de distintos lenguajes.

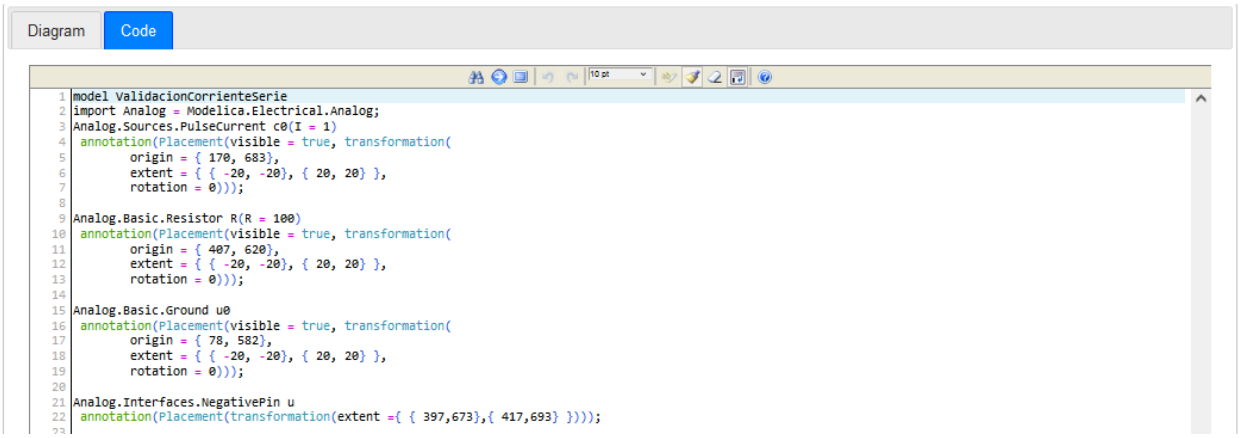


Figura 5-5 Vista del editor de código

El componente se carga en un contenedor HTML especificando una serie de propiedades entre ellas, a que sintaxis vamos a hacer referencia.

```
            $("#tabCode").html("<textarea id='code_1' style='height:100%; width: 100%;'
name='test_1'></textarea>");
            editAreaLoader.init({
                id: "code_1" // id of the textarea to transform
                , start_highlight: true    // if start with highlight
                , allow_resize: "both"
                , allow_toggle: true
                , font_size: 10
                , word_wrap: true
                , language: "en"
                , syntax: "modelica"
            });
```

La herramienta Edit_Area lleva la sintaxis de los lenguajes más comunes incorporada de serie (C, Java, HTML, etc.), pero desgraciadamente Modelica no, no obstante, podemos crear una sintaxis nueva, creando un fichero con las palabras claves de este lenguaje. En este fichero, definimos el estilo de los comentarios, los operadores, las palabras claves para funciones o reservadas, etc. El editor usará este fichero para dar formato al código generado y diferenciar visualmente sentencias, variables etc.

```
editAreaLoader.load_syntax["modelica"] = {
	'DISPLAY_NAME' : 'Modelica'
	,'COMMENT_SINGLE' : {1 : '//', 2 : '#'}
	,'COMMENT_MULTI' : {'/*' : '*/'}
	,'QUOTEMARKS' : {1: "'", 2: '"'}
	,'KEYWORD_CASE_SENSITIVE' : false
	,'KEYWORDS' : {
		'statements' : [
			'include', 'require', 'model', 'constant', 'parameter', 'equation',
			'end', 'annotation', 'connect'
		]
		,'reserved' : [
			'false', '&lt;?php', '?&gt;', '&lt;?', '&lt;script language',
			'&lt;/script&gt;', 'true', 'var', 'default', 'function', 'class', 'new',
			'&amp;new', 'this', 'Placement', 'transformation', 'Line'
		]
		,'functions' : [
			'echo', 'print', 'global', 'static', 'exit', 'array', 'empty', 'eval', -
		       'isset', 'unset', 'die'
		]
	}
	,'OPERATORS' :[
		'+', '-', '/', '*', '=', '<', '>', '%', '!', '&&', '||'
	]
	,'DELIMITERS' :[
		'(', ')', '[', ']', '{', '}'

		]
```

5.7 Accesibilidad y usabilidad

La accesibilidad se refiere a la facilidad de uso de un producto o servicio por parte de todas las personas, independientemente de sus habilidades o limitaciones físicas, sensoriales o cognitivas. La usabilidad, por otro lado, se refiere a la facilidad de uso de un producto o servicio para un usuario específico en un contexto determinado. Ambas son importantes para garantizar que un producto o servicio sea accesible y fácil de usar para todos los usuarios.

Para garantizar la accesibilidad y la usabilidad de un producto o servicio, es importante tener en cuenta a todos los posibles usuarios y sus necesidades durante el diseño y desarrollo de este. Esto incluye considerar a personas con discapacidad y a aquellas que puedan tener dificultades para usar el producto o servicio por razones culturales o de idioma. También es importante probar y evaluar el producto o servicio con una amplia variedad de usuarios para asegurar que es fácil de usar para todos ellos.

Hemos seguido las Pautas de accesibilidad al contenido web (WCAG) 2.0, son un conjunto de pautas desarrolladas por el World Wide Web Consortium (W3C) para hacer que el contenido web sea más accesible para las personas con discapacidades. Las pautas cubren una amplia gama de recomendaciones para hacer que el contenido web sea más accesible, incluidas recomendaciones sobre cómo diseñar páginas web, escribir texto, usar colores y crear contenido

que sea accesible para personas con discapacidades. WCAG 2.0 se basa en cuatro principios: perceptible, operable, comprensible y sólido. El cumplimiento de estas pautas ayuda a que la web sea más accesible para las personas con discapacidades y también facilita el uso de la web para todos.

Hemos usado analizadores WCAG como Wave para comprobar estos criterios, y hemos ido depurando e incorporando las recomendaciones que indica la herramienta de evaluación.

5.8 Conclusiones

Un editor de modelos es una herramienta de software que se utiliza para crear y modificar modelos de datos, procesos o sistemas. Los modelos pueden ser utilizados para representar la estructura y el funcionamiento de un sistema en una forma gráfica o visual.

En este capítulo hemos comentado todas la partes que forman el editor de modelos, así como sus opciones, herramientas y algunas pinceladas de cómo se han implementado. Esta es la pantalla principal de interacción del usuario y la que desarrolla el 95% del trabajo de la aplicación.

6 Fase de pruebas en OpenModelica

6.1 Introducción

Para la fase de pruebas vamos a utilizar varios modelos de referencia y los vamos a comparar tanto en su versión plana como compuesta. A continuación veremos las mediciones de tensión y corriente para algunos de los puntos del circuito y la coincidencia entre ambos modelos, además, vamos a comprobar como sentaría al modelo cambios en algunos de sus elementos.

6.2 Modelo de Test 1

Para el primer test he elegido un circuito en serie donde vamos a tener dos generadores sinusoidales, tres resistencia y un condensador.

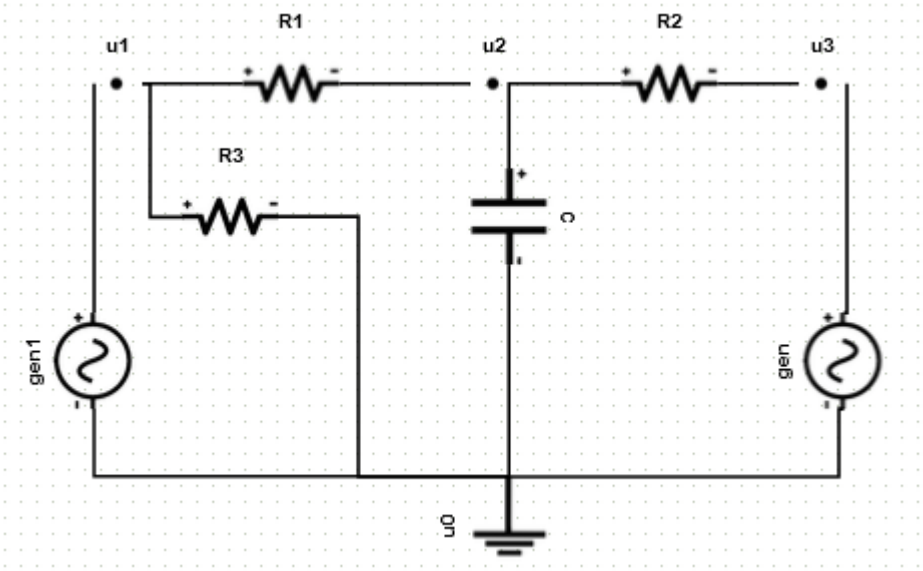


Figura 6-1 Circuito de Test 1

Hemos pasado los dos modelos (atómico y compuesto) a OpenModelica realizando una simulación subiendo a 5000 el número de intervalos. En este caso nos centramos en el voltaje en el condensador viendo que tanto los valores del modelo atómico como el compuesto coinciden. Podemos comprobar como el voltaje oscila entre los 5 y -5 voltios. Por poner otro ejemplo, en la Figura 6-3 mostramos la corriente en la resistencia R3.

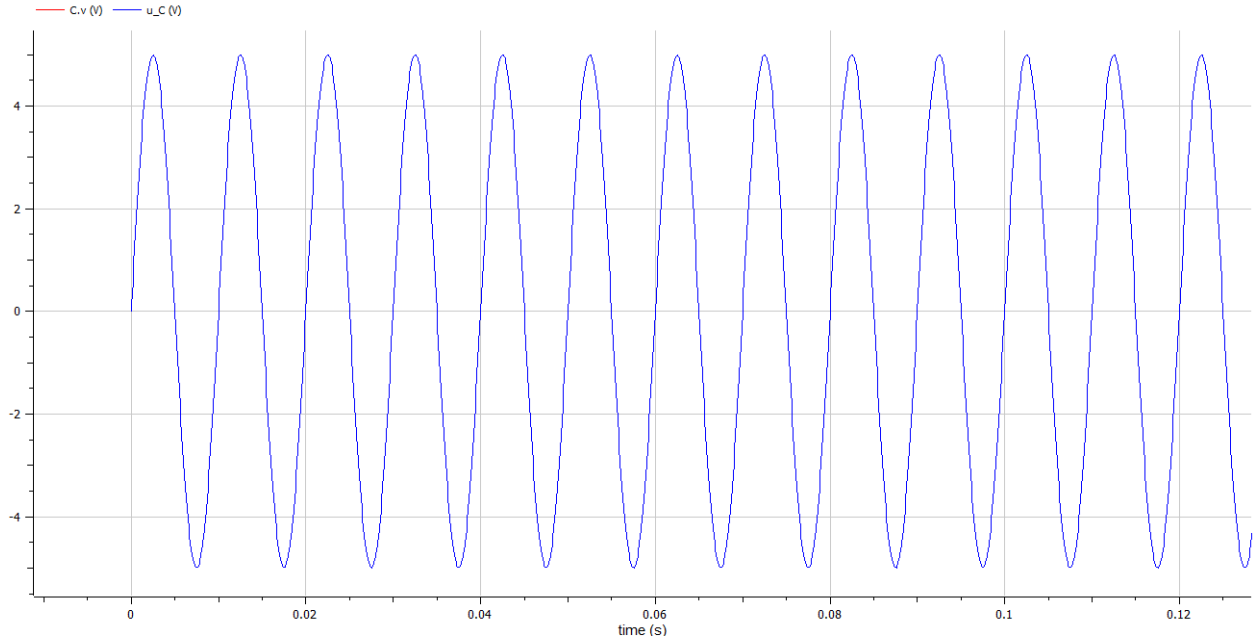


Figura 6-3 Voltaje en el condensador de los modelos atómico y compuesto del Test1

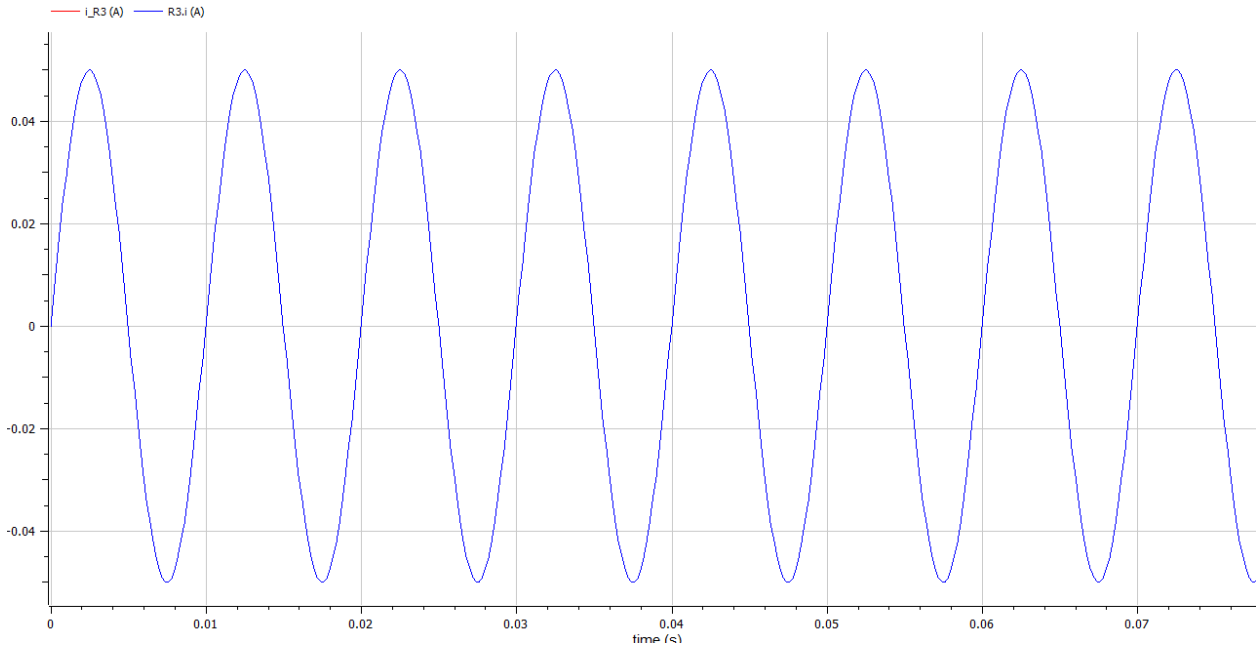


Figura 6-2 Corriente en la resistencia R3

El código del modelo atómico será el siguiente:

```
model Test1Flat
import SI = Modelica.Units.SI;
  SI.Current i_gen1;// "Corriente gen1"
  SI.Current i_gen2;// "Corriente gen2"
  SI.Current i_R1;// "Corriente R1"
  SI.Current i_R2;// "Corriente R2"
  SI.Current i_R3;// "Corriente R3"
  SI.Current i_C;// "Corriente C"
  SI.Voltage u_C(start=0, fixed=true);// "Voltaje C"
  SI.Voltage u1;// "Voltaje u1"
  SI.Voltage u2;// "Voltaje u2"
  SI.Voltage u3;// "Voltaje u3"
// "Parámetros del generador gen1"
  parameter SI.Voltage Ugen1 = 5;
  parameter SI.Frequency frec_gen1 = 100;
  parameter SI.AngularFrequency w_gen1 = 2 * Modelica.Constants.pi *
frec_gen1;
  parameter SI.Angle phi_gen1 = 0;
// "Parámetros del generador gen2"
  parameter SI.Voltage Ugen2 = 5;
  parameter SI.Frequency frec_gen2 = 100;
  parameter SI.AngularFrequency w_gen2 = 2 * Modelica.Constants.pi *
frec_gen2;
  parameter SI.Angle phi_gen2 = 0;
// "Parámetros de la Resistencia R1"
  parameter SI.Resistance R1 = 100;
// "Parámetros de la Resistencia R2"
  parameter SI.Resistance R2 = 100;
// "Parámetros de la Resistencia R3"
  parameter SI.Resistance R3 = 100;
// "Parámetros del condensador C"
  parameter SI.Capacitance C = 1e-06;

equation
// Ecuaciones en los nodos
  i_R1 + i_R3 = i_gen1;
  i_C + i_R2 = i_R1;
  0 = i_gen2 + i_R2;
// Relaciones constitutivas
  u1 = Ugen1 * sin( w_gen1 * time + phi_gen1);
  u3 = Ugen2 * sin( w_gen2 * time + phi_gen2);
  u1 - u2 = i_R1 * R1;
  u2 - u3 = i_R2 * R2;
  u1 = i_R3 * R3;
  i_C = C * der( u_C );
  u2 = u_C;
end Test1Flat;
```

El código del modelo compuesto será el siguiente:

```
model Test1Composite
import Analog = Modelica.Electrical.Analog;
Analog.Sources.SineVoltage gen1(V = 5, offset = 0 , f = 100)
 annotation(Placement(visible = true, transformation(
        origin = { 45, 577},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -90)));

Analog.Sources.SineVoltage gen2(V = 5, offset = 0 , f = 100)
 annotation(Placement(visible = true, transformation(
        origin = { 520, 577},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -90)));

Analog.Basic.Resistor R1(R = 100)
 annotation(Placement(visible = true, transformation(
        origin = { 168, 750},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Basic.Resistor R2(R = 100)
 annotation(Placement(visible = true, transformation(
        origin = { 404, 750},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Basic.Resistor R3(R = 100)
 annotation(Placement(visible = true, transformation(
        origin = { 130, 667},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Basic.Capacitor C(C = 1E-06)
 annotation(Placement(visible = true, transformation(
        origin = { 304, 667},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -270)));

Analog.Basic.Ground u0
 annotation(Placement(visible = true, transformation(
        origin = { 284, 475},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -90)));

Analog.Interfaces.NegativePin u1
 annotation(Placement(transformation(extent ={ { 50,740},{ 70,760} })));

Analog.Interfaces.NegativePin u2
```

```
 annotation(Placement(transformation(extent ={ { 285,740},{ 305,760} })));

Analog.Interfaces.NegativePin u3
 annotation(Placement(transformation(extent ={ { 490,740},{ 510,760} })));


equation
connect(u0.p, gen1.n) annotation (Line(points
  ={{304,475},{45,577},{45,577}}, color ={ 0,0,255}));
connect(gen1.p, u1) annotation (Line(points ={{45,577},{60,750},{60,750}},
  color ={ 0,0,255}));
connect(u1, R1.p) annotation (Line(points ={{60,750},{168,750}}, color ={
  0,0,255}));
connect(R1.n, u2) annotation (Line(points ={{168,750},{295,750}}, color ={
  0,0,255}));
connect(u2, R2.p) annotation (Line(points ={{295,750},{404,750}}, color ={
  0,0,255}));
connect(R2.n, u3) annotation (Line(points ={{404,750},{500,750}}, color ={
  0,0,255}));
connect(u3, gen2.p) annotation (Line(points
  ={{500,750},{520,577},{520,577}}, color ={ 0,0,255}));
connect(u2, C.p) annotation (Line(points ={{295,750},{304,667}}, color ={
  0,0,255}));
connect(C.n, u0.p) annotation (Line(points ={{304,667},{304,475}}, color
  ={ 0,0,255}));
connect(u0.p, gen2.n) annotation (Line(points
  ={{304,475},{520,475},{520,577}}, color ={ 0,0,255}));
connect(u1, R3.p) annotation (Line(points ={{60,750},{130,750},{130,667}},
  color ={ 0,0,255}));
connect(R3.n, u0.p) annotation (Line(points
  ={{130,667},{240,637},{240,475},{304,475}}, color ={ 0,0,255}));

annotation(
        uses(Modelica(version = "4.0.0")),
        Diagram(coordinateSystem(extent = { { 0, 0}, { 1400, 1400} })),
        Icon(coordinateSystem(extent = {{0, 0}, {1400, 1400}})),
        version = "");

end Test1Composite;
```

Sobre el mismo modelo vamos a realizar una variación, cambiando los componentes C y gen2 invirtiendo sus conexiones, en este caso los polos positivos van directamente a tierra.

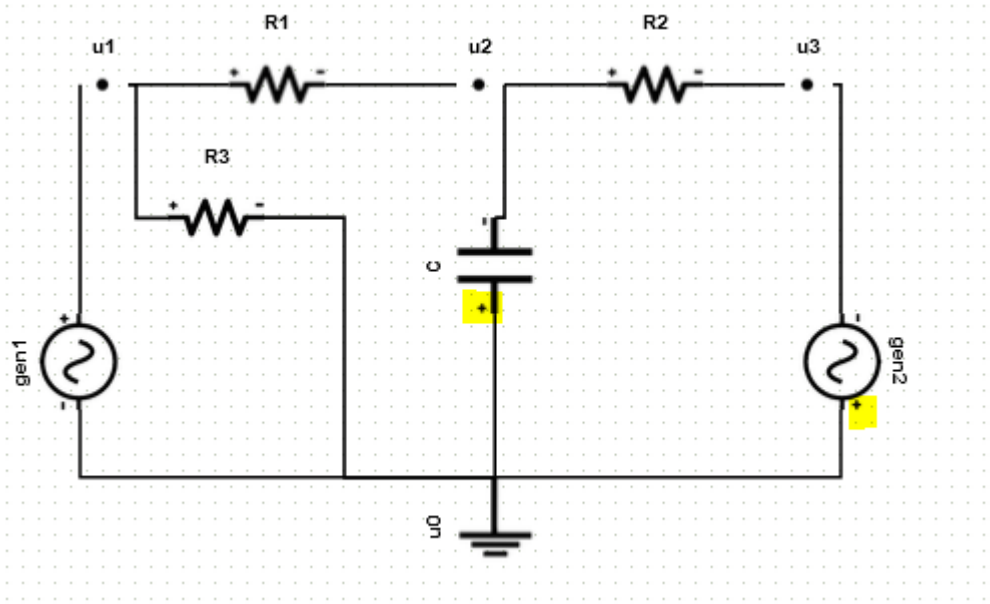


Figura 6-4 Variación del modelo de test1

Si nos fijamos ahora en el modelo atómico, observamos como las ecuaciones cambian. En este caso la tensión en gen2 es igual a -u3 y en el condensador es -u2.

En las ecuaciones en los nodos tenemos que en el nodo u3, `i_gen2 = i_R2` y en el nodo u2, `i_R2 = i_C + i_R1`, ya que la corriente de entrada en u2 del condensador es negativa.

```
equation
// Ecuaciones en los nodos
  i_R1 + i_R3 = i_gen1;
  i_R2 = i_C + i_R1;
  i_gen2 = i_R2;
// Relaciones constitutivas
  u1 = Ugen1 * sin( w_gen1 * time + phi_gen1);
   - u3 = Ugen2 * sin( w_gen2 * time + phi_gen2);
  u1 - u2 = i_R1 * R1;
  u2 - u3 = i_R2 * R2;
  u1 = i_R3 * R3;
  i_C = C * der( u_C );
   - u2 = u_C;
end Test9Flat;
```

6.3 Modelo de Test 2

Para el Test 2 he elegido un modelo en el cual tendremos resistencias en paralelo con el fin de ver como se distribuye la corriente y la tensión en el circuito, vamos a comprobar en este experimento como la suma de los voltajes debe ser igual a 0 y como las intensidades de corrientes se distribuyen por igual en las resistencias en paralelo (todas las resistencias se asumen de 100 ohmios).

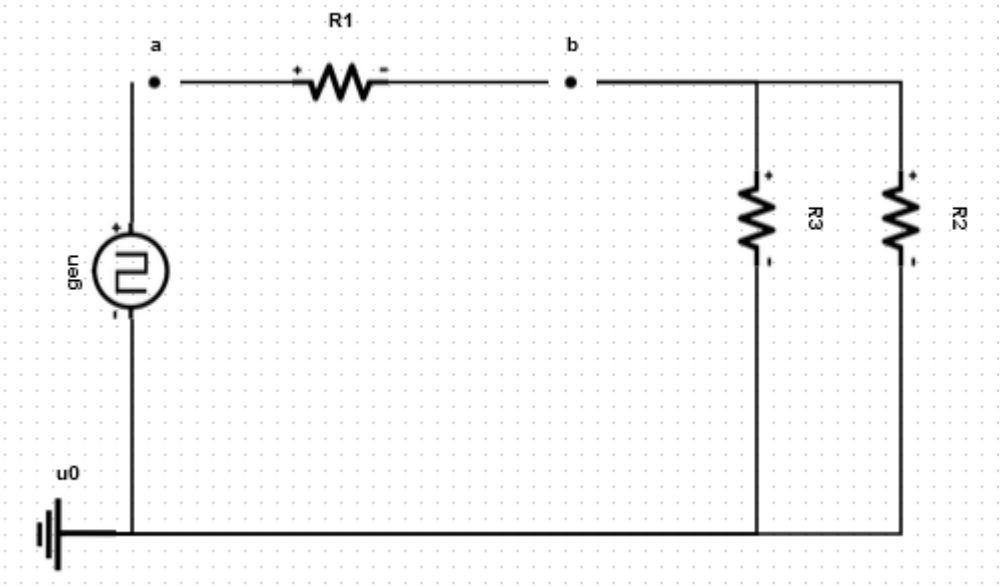


Figura 6-5 Modelo de Test 2

A continuación se muestra la traducción al modelo atómico y modelo compuesto, nótese como en las ecuaciones en los nodos la corriente de R1 es igual a la suma de las corrientes de R2 y R3. El modelo atómico y compuesto quedaría con el código que mostramos a continuación:

```
model Test2Flat
import SI = Modelica.Units.SI;
  SI.Current i_gen;
  SI.Voltage u_gen;
  Boolean up_gen;
  SI.Current i_R1;// " Corriente R1"
  SI.Current i_R2;// " Corriente R2"
  SI.Current i_R3;// " Corriente R3"
  SI.Voltage a;// "Voltaje a"
  SI.Voltage b;// "Voltaje b"
// "Parámetros del generador gen"
  parameter SI.Voltage Ugen = 5;
  parameter SI.Time Tgen = 0.0025;
// "Parámetros de la Resistencia R1"
  parameter SI.Resistance R1 = 100;
// "Parámetros de la Resistencia R2"
  parameter SI.Resistance R2 = 100;
// "Parámetros de la Resistencia R3"
  parameter SI.Resistance R3 = 100;

equation
// Ecuaciones en los nodos
  i_R1 = i_gen;
  i_R2 + i_R3 = i_R1;
// Relaciones constitutivas
  when sample(0,Tgen) then
    up_gen = not pre(up_gen);
  end when;
  u_gen = if up_gen then Ugen else 0;
  a = u_gen;
  a - b = i_R1 * R1;
  b = i_R2 * R2;
  b = i_R3 * R3;
end Test2Flat;
```

```
model Test2Composite
import Analog = Modelica.Electrical.Analog;
Analog.Sources.PulseVoltage gen(V = 5, offset = 0, period = 0.005 )
 annotation(Placement(visible = true, transformation(
        origin = { 30, 650},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Basic.Resistor R1(R = 100)
 annotation(Placement(visible = true, transformation(
        origin = { 190, 762},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Basic.Resistor R2(R = 100)
 annotation(Placement(visible = true, transformation(
        origin = { 538, 667},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Basic.Resistor R3(R = 100)
 annotation(Placement(visible = true, transformation(
        origin = { 403, 677},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Basic.Ground u0
 annotation(Placement(visible = true, transformation(
        origin = { 20, 480},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Interfaces.NegativePin a
 annotation(Placement(transformation(extent ={ { 65,752},{ 85,772} })));

Analog.Interfaces.NegativePin b
 annotation(Placement(transformation(extent ={ { 325,752},{ 345,772} })));

equation
connect(u0.p, gen.n) annotation (Line(points
={{20,480},{30,650},{30,650}}, color ={ 0,0,255}));
```

```
41 connect(gen.p, a) annotation (Line(points ={{30,650},{75,762},{75,762}},
   color ={ 0,0,255}));
42 connect(a, R1.p) annotation (Line(points ={{75,762},{190,762}}, color ={
   0,0,255}));
43 connect(R1.n, b) annotation (Line(points ={{190,762},{335,762}}, color ={
   0,0,255}));
44 connect(b, R2.p) annotation (Line(points ={{335,762},{538,667},{538,667}},
   color ={ 0,0,255}));
   connect(R2.n, u0.p) annotation (Line(points
   ={{538,667},{538,480},{20,480}}, color ={ 0,0,255}));
45 connect(b, R3.p) annotation (Line(points ={{335,762},{403,677},{403,677}},
   color ={ 0,0,255}));
46 connect(R3.n, u0.p) annotation (Line(points
   ={{403,677},{403,480},{20,480}}, color ={ 0,0,255}));
47
48 annotation(
49         uses(Modelica(version = "4.0.0")),
50         Diagram(coordinateSystem(extent = { { 0, 0}, { 1400, 1400} })),
51         Icon(coordinateSystem(extent = {{0, 0}, {1400, 1400}})),
52         version = "");
53
54 end Test2Composite;
```

Cuando empezamos la simulación podemos ver como la suma de la intensidades i_R2 e i_R3 son iguales a i_R1.

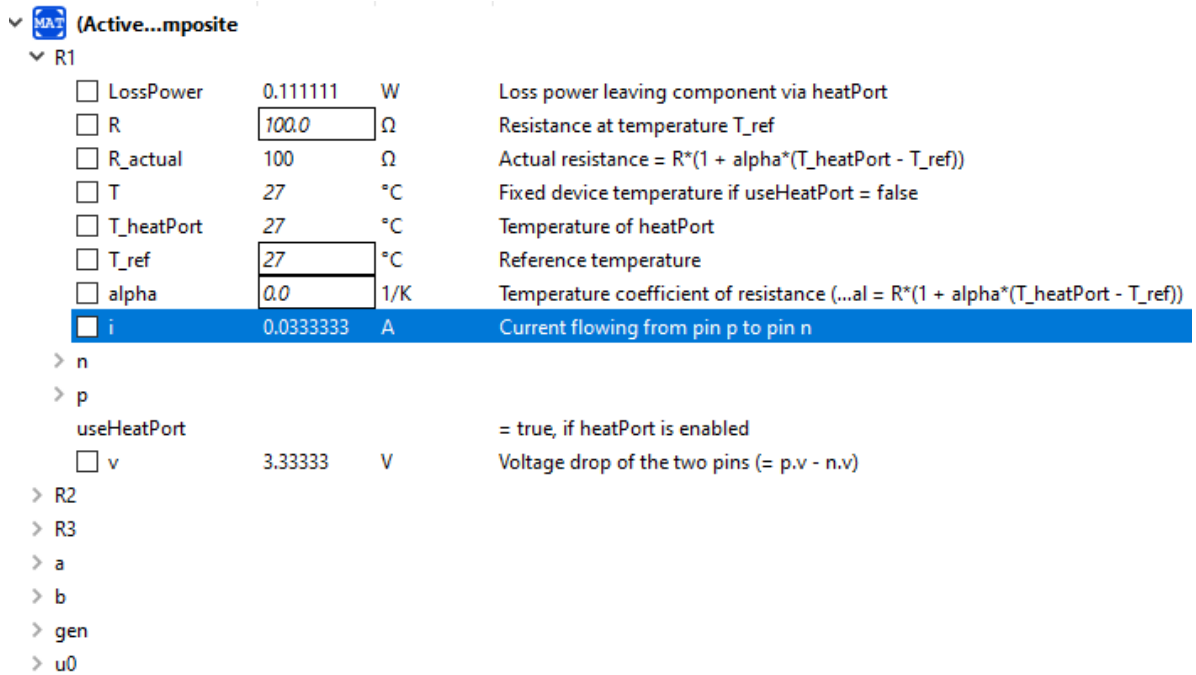


Figura 6-6 Valores de la simulación sobre el voltaje en R1 en Test 2

Si nos fijamos ahora en los voltajes, observamos como la suma del voltaje en el generador más la suma de los voltajes de R1 y R2 con R3 en paralelo son iguales a 0, R2 y R3 tienen el mismo voltaje -0,0166667, pero al estar en paralelo sumarían solo como uno. En la Figura 6-3 podemos ver los valores y en la gráfica de la Figura 6-6 vemos como se complementan.

Variables	Value	Display Unit	Description
R1			
R2			
LossPower	0.0277778	W	Loss power leaving component via heatPort
R	100.0	Ω	Resistance at temperature T_ref
R_actual	100	Ω	Actual resistance = R*(1 + alpha*(T_heatPort - T_ref))
T	27	°C	Fixed device temperature if useHeatPort = false
T_heatPort	27	°C	Temperature of heatPort
T_ref	27	°C	Reference temperature
alpha	0.0	1/K	Temperature coefficient of resistance (...al = R*(1 + alpha*(T_heatPort - T_ref))
i	0.0166667	A	Current flowing from pin p to pin n
n			
p			
useHeatPort			= true, if heatPort is enabled
v	1.66667	V	Voltage drop of the two pins (= p.v - n.v)
R3			
LossPower	0.0277778	W	Loss power leaving component via heatPort
R	100.0	Ω	Resistance at temperature T_ref
R_actual	100	Ω	Actual resistance = R*(1 + alpha*(T_heatPort - T_ref))
T	27	°C	Fixed device temperature if useHeatPort = false
T_heatPort	27	°C	Temperature of heatPort
T_ref	27	°C	Reference temperature
alpha	0.0	1/K	Temperature coefficient of resistance (...al = R*(1 + alpha*(T_heatPort - T_ref))
i	0.0166667	A	Current flowing from pin p to pin n

Figura 6-7 Valores sobre la simulación en el voltaje de R2 y R3 del Test 2

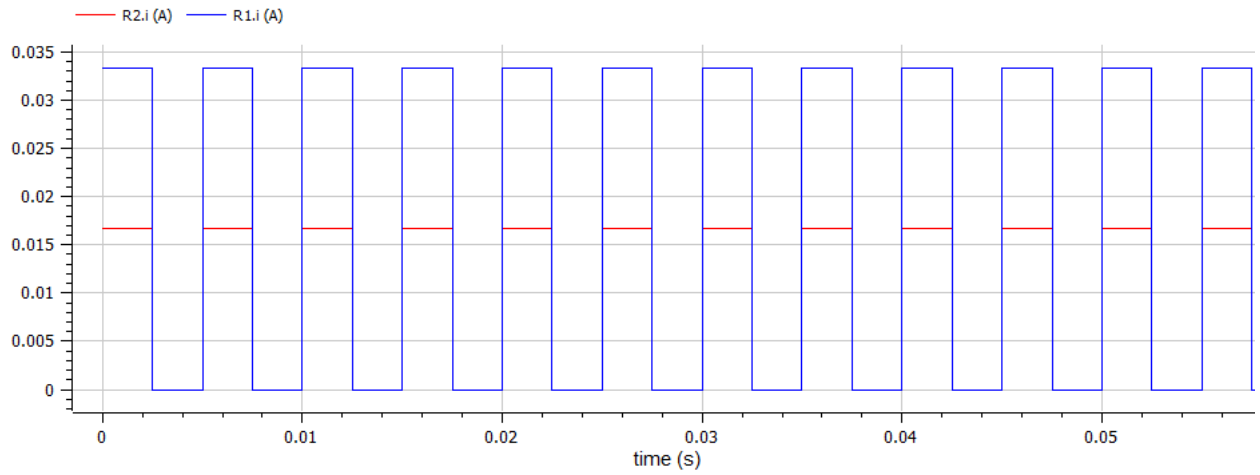


Figura 6-8 Valores de corriente en R1 y R2 sobre la simulación del Test 2

6.4 Modelo de Test 3

Vamos a realizar un experimento en el tercer test intentando aplicar el teorema de Thévenin. El teorema de Thévenin es una herramienta útil en el análisis de circuitos eléctricos que permite reducir un circuito complejo a una fuente de voltaje en serie con una resistencia. Este teorema establece que cualquier circuito eléctrico puede ser representado por una fuente de voltaje ideal en serie con una resistencia de Thévenin, independientemente de la configuración y la carga del circuito.

Para aplicar el teorema de Thévenin, primero debes identificar la fuente de voltaje y la resistencia de Thévenin. La fuente de voltaje de Thévenin es el voltaje que se observaría en el punto de medida (denominado "terminal de medición") si se cortara la conexión entre el punto de medida y cualquier otro elemento del circuito y se pusiera en su lugar una resistencia de Thévenin. La resistencia de Thévenin es la resistencia que se observaría en el punto de medida si se cortara la conexión entre el punto de medida y cualquier otro elemento del circuito y se pusiera en su lugar una fuente de voltaje ideal.

Una vez que se han determinado la fuente de voltaje y la resistencia de Thévenin, el circuito original puede ser representado por una fuente de voltaje en serie con una resistencia de Thévenin. Esto permite simplificar el análisis del circuito y hacer cálculos más precisos y fáciles.

Vamos a usar un circuito de partida con una fuente de 100 voltios y 3 resistencias (R1, R2 y R3), intentaremos medir la corriente que pasa por una resistencia de carga (RL) aislada entre dos nodos a y b. La corriente en esta resistencia debe ser igual a sustituir el resto del circuito por su equivalente Thévenin reducido.

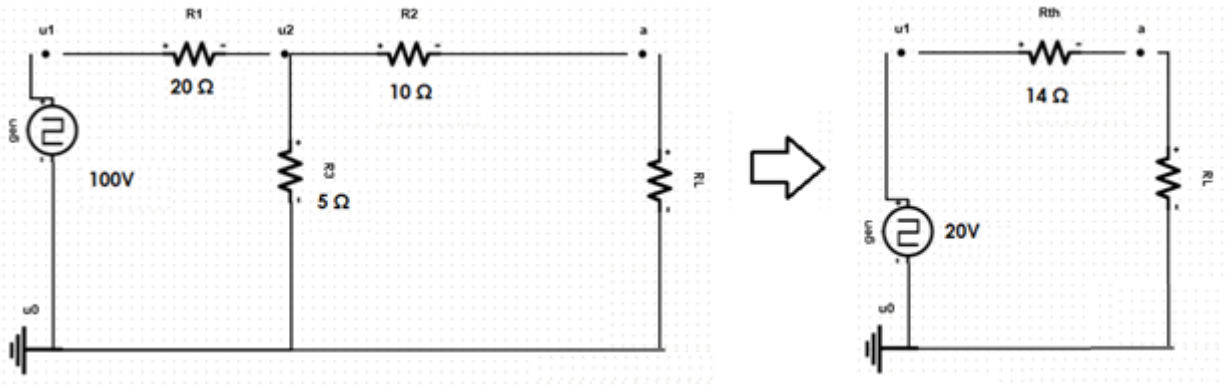


Figura 6-9 Modelo de Test 3 y su equivalente Thévenin reducido.

En primer lugar debemos calcular la tensión de Thévenin entre los puntos a y b (en la Figura 6-9 el punto b debería ser el nodo tierra), para ello dividimos el voltaje entre la suma de las

resistencias en serie y la multiplicamos por la resistencia en paralelo. Obtendremos un voltaje de Thévenin de 20 V.

$$V_{TH} = \frac{100V}{20\Omega + 5\Omega} \cdot 5\Omega \ = 20 \ V$$

A continuación, podemos calcular la resistencia total entre los puntos a y b, puesto que R1 y R3 se encuentran en el mismo nudo, calculamos su equivalente y luego sumamos la resistencia R2 que está en lazo abierto con los terminales a y b, es decir, solo, está conectada en un pin al circuito en caso de que quitemos la resistencia de carga.

$$R_{TH} = \frac{20 \cdot 5}{20 + 5} + 10 \ \Omega = 14 \ \Omega$$

Una vez generado un circuito equivalente, podemos exportar los dos modelos a OpenModelica, y compararlos. En este caso generamos el modelo compuesto para ambos (los llamaremos test3Composite y test3CompositeTh).

Comprobamos en las propiedades de la simulación que la corriente en la resistencia de carga es la misma para ambos modelos, el valor de 0.0416667 Amperios.

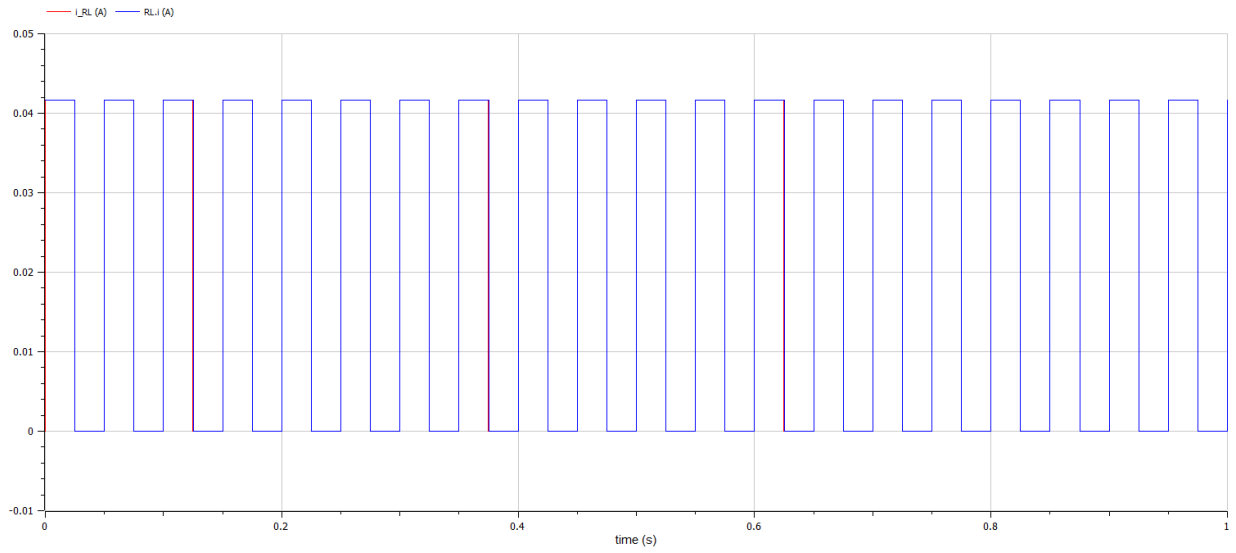


Figura 6-10 Propiedades de la simulación de los modelos para Test 3

El código del modelo atómico y compuesto se muestra a continuación:

```
model Test3Flat
import SI = Modelica.Units.SI;
  SI.Current i_gen;
  SI.Voltage u_gen;
  Boolean up_gen;
  SI.Current i_R1;// " Corriente R1"
  SI.Current i_R3;// " Corriente R3"
  SI.Current i_R2;// " Corriente R2"
  SI.Current i_RL;// " Corriente RL"
  SI.Voltage a;// "Voltaje a"
  SI.Voltage b;// "Voltaje b"
  SI.Voltage u2;// "Voltaje u2"
  SI.Voltage u1;// "Voltaje u1"
// "Parámetros del generador gen"
  parameter SI.Voltage Ugen = 5;
  parameter SI.Time Tgen = 0.0025;
// "Parámetros de la Resistencia R1"
  parameter SI.Resistance R1 = 20;
// "Parámetros de la Resistencia R3"
  parameter SI.Resistance R3 = 5;
// "Parámetros de la Resistencia R2"
  parameter SI.Resistance R2 = 10;
// "Parámetros de la Resistencia RL"
  parameter SI.Resistance RL = 10;

equation
// Ecuaciones en los nodos
  i_RL = i_R2;
  b = 0;
  i_R2 + i_R3 = i_R1;
  i_R1 = i_gen;
// Relaciones constitutivas
  when sample(0,Tgen) then
    up_gen = not pre(up_gen);
  end when;
  u_gen = if up_gen then Ugen else 0;
  u1 = u_gen;
  u1 - u2 = i_R1 * R1;
  u2 = i_R3 * R3;
  u2 - a = i_R2 * R2;
  a - b = i_RL * RL;
end Test3Flat;
```

```
model Test3Composite
import Analog = Modelica.Electrical.Analog;
Analog.Sources.PulseVoltage gen(V = 5, offset = 0 , period = 0.005)
 annotation(Placement(visible = true, transformation(
        origin = { 27, 706},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -90)));

Analog.Basic.Resistor R1(R = 20)
 annotation(Placement(visible = true, transformation(
        origin = { 159, 778},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Basic.Resistor R3(R = 5)
 annotation(Placement(visible = true, transformation(
        origin = { 249, 668},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -270)));

Analog.Basic.Resistor R2(R = 10)
 annotation(Placement(visible = true, transformation(
        origin = { 360, 778},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Basic.Resistor RL(R = 10)
 annotation(Placement(visible = true, transformation(
        origin = { 595, 659},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -270)));

Analog.Basic.Ground u0
 annotation(Placement(visible = true, transformation(
        origin = { 6, 500},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Interfaces.NegativePin a
 annotation(Placement(transformation(extent ={ { 570,768},{ 590,788}
}})));

Analog.Interfaces.NegativePin b
 annotation(Placement(transformation(extent ={ { 585,505},{ 605,525}
}})));

Analog.Interfaces.NegativePin u2
 annotation(Placement(transformation(extent ={ { 235,768},{ 255,788}
}})));

```

```
48 Analog.Interfaces.NegativePin u1
49  annotation(Placement(transformation(extent ={ { 11,768},{ 31,788} })));
50
51
52 equation
53 connect(R2.n, a) annotation (Line(points ={{360,778},{580,778}}, color ={
   0,0,255}));
54 connect(R3.n, u0.p) annotation (Line(points
   ={{249,668},{249,500},{6,500}}, color ={ 0,0,255}));
55 connect(a, RL.p) annotation (Line(points ={{580,778},{595,659}}, color ={
   0,0,255}));
56 connect(b, u0.p) annotation (Line(points ={{595,515},{595,500},{6,500}},
   color ={ 0,0,255}));
57 connect(u0.p, gen.n) annotation (Line(points
   ={{6,500},{27,706},{27,706}}, color ={ 0,0,255}));
58 connect(R1.n, u2) annotation (Line(points ={{159,778},{245,778}}, color
   ={ 0,0,255}));
59 connect(u2, R2.p) annotation (Line(points ={{245,778},{360,778}}, color
   ={ 0,0,255}));
60 connect(u2, R3.p) annotation (Line(points ={{245,778},{249,668}}, color
   ={ 0,0,255}));
61 connect(RL.n, b) annotation (Line(points ={{595,659}}, color ={
   0,0,255}));
62 connect(gen.p, u1) annotation (Line(points ={{27,706},{21,778},{21,778}},
   color ={ 0,0,255}));
63 connect(u1, R1.p) annotation (Line(points ={{21,778},{159,778}}, color ={
   0,0,255}));
64
65 annotation(
66         uses(Modelica(version = "4.0.0")),
67         Diagram(coordinateSystem(extent = { { 0, 0}, { 1400, 1400} })),
68         Icon(coordinateSystem(extent = {{0, 0}, {1400, 1400}})),
69         version = "");
70
71 end Test3Composite;
```

6.5 Modelo de Test 4

Para el siguiente test vamos a crear un modelo con un interruptor que se accione y se desconecte cada segundo, por lo que en ese segundo habrá dos cambios de estado. De tal manera que podamos verificar cómo se comporta un elemento, un condensador que dependa de que el circuito quede cerrado con el interruptor.

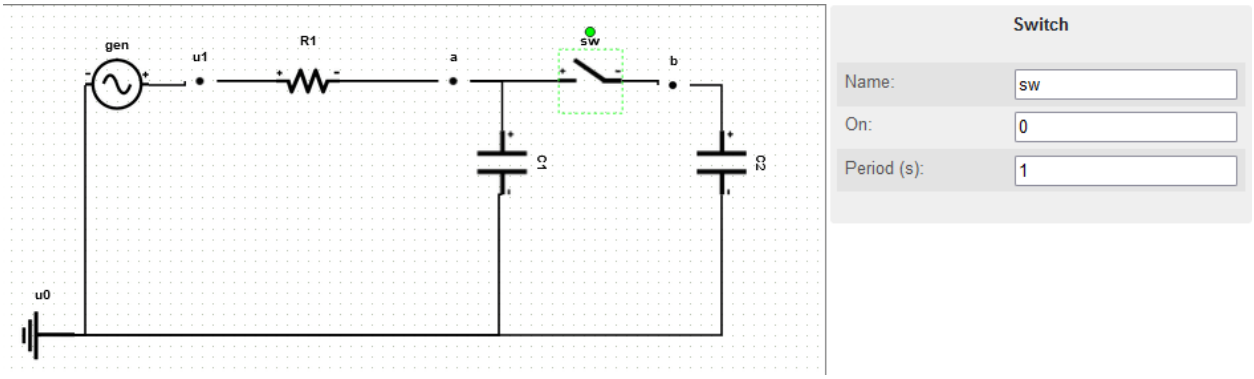


Figura 6-11 Modelo del Test 4

El interruptor se ha implementado como el tipo “pulso booleano” en Modelica, este objeto tendrá la propiedad “period” que indicará el intervalo de tiempo en el que se repetirá tanto el cierre en el circuito como la apertura, por lo que cada 0,5 segundos se producirá un cambio de estado.

```
Modelica.Blocks.Sources.BooleanPulse booleanPulse(period= 1);
Modelica.Electrical.Analog.Ideal.IdealOpeningSwitch u
 annotation(Placement(visible = true, transformation(
        origin = {469, 761},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));
```

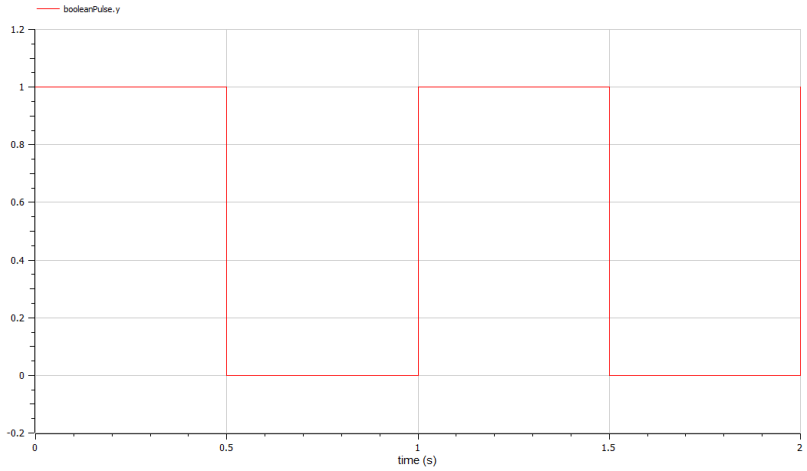


Figura 6-12 Cambios de estado en el interruptor

Si medimos la tensión en el condensador C2, el condensador que queda aislado del circuito, vemos como se produce claramente un intervalo en el que la tensión es nula y otro intervalo en la cual oscila según la fuente aplicada en este caso de onda sinusoidal.

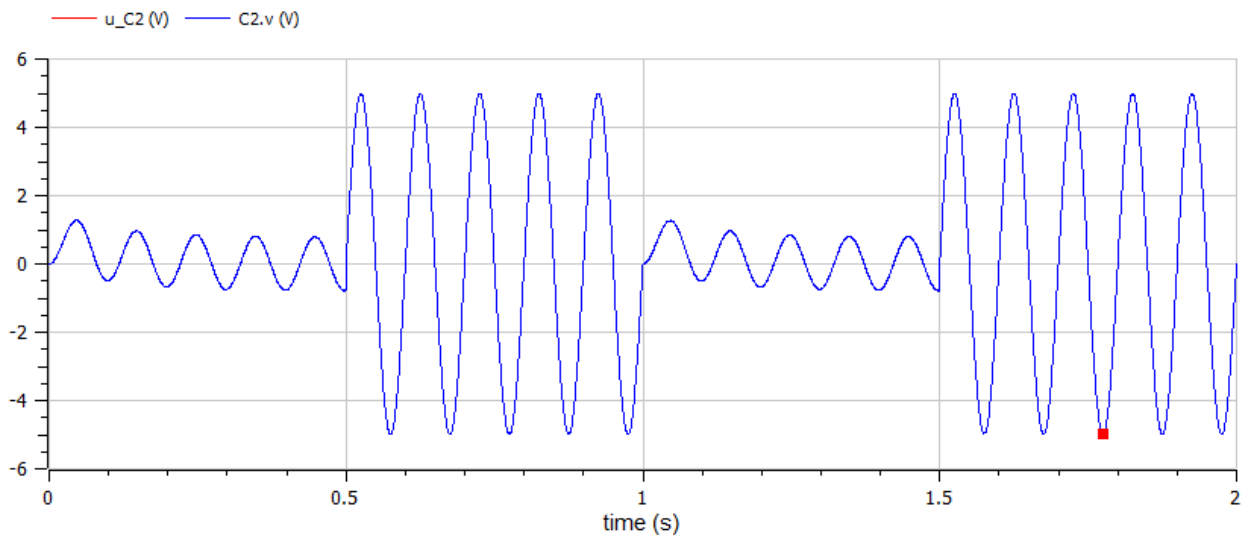


Figura 6-13 Valor del voltaje en el condensador C2

El condensador C1, por el contrario permanece en todo momento dentro del circuito, por lo cual no se distinguen los intervalos de apertura y cierre del interruptor. En la Figura 6-12 se muestra el voltaje del condensador C1, la simulación se ha realizado configurando a 5000 el número de intervalos. Podemos ver además, como se generaría el diagrama en OpenModelica usando las anotaciones de posición generadas en nuestro modelo compuesto, en la Figura 6-13.

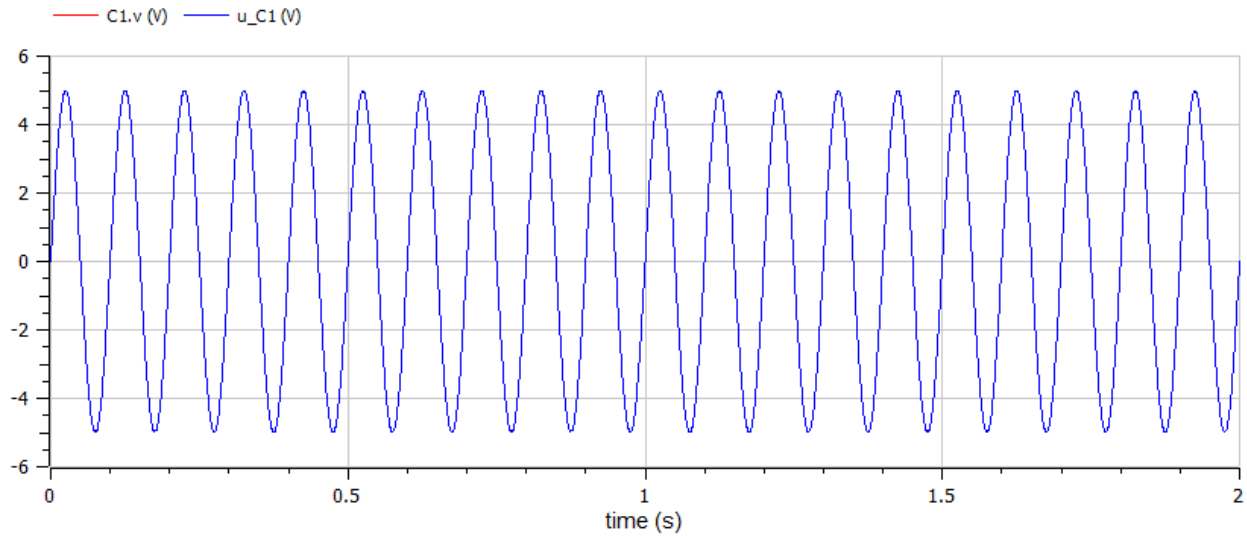


Figura 6-14 Valor del voltaje en el condensador C1

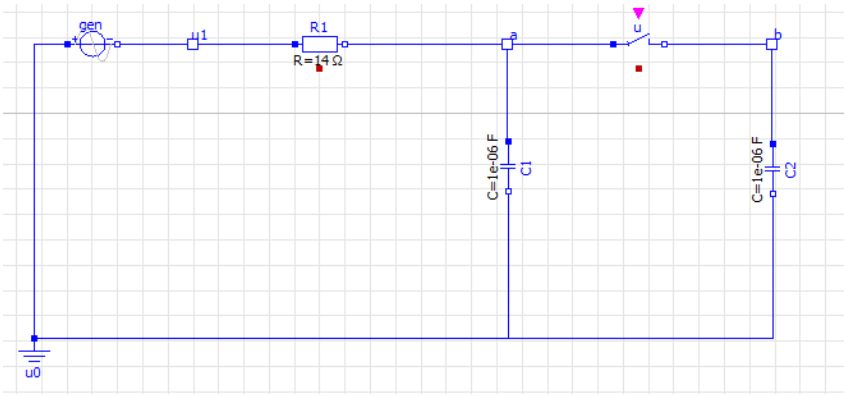


Figura 6-15 Diagrama del modelo en OpenModelica

A continuación mostramos el código para los modelos atómico y compuesto generados para el Test 4.

```
model Test4Flat
import SI = Modelica.Units.SI;
  SI.Current i_gen;// "Voltaje gen"
  SI.Current i_sw;// "Corriente sw"
  SI.Voltage v_sw;// "Voltaje sw"
  Boolean off_sw(start=false, fixed=true);
  SI.Current i_R1;// "Voltaje R1"
  SI.Current i_C1;
  SI.Voltage u_C1(start=0, fixed=true);// "Voltaje C1"
  SI.Current i_C2;
  SI.Voltage u_C2(start=0, fixed=true);// "Voltaje C2"
  SI.Voltage a;// "Voltaje a"
  SI.Voltage b;// "Voltaje b"
  SI.Voltage u1;// "Voltaje u1"
// "Parámetros del generador gen"
  parameter SI.Voltage Ugen = 5;
  parameter SI.Frequency frec_gen = 10;
  parameter SI.AngularFrequency w_gen = 2 * Modelica.Constants.pi *
frec_gen;
  parameter SI.Angle phi_gen = 0;
// "Parámetros del interruptor sw"
  parameter SI.Time Tsw = 0.5;
// "Closed switch resistancer sw"
  parameter SI.Resistance Ron(final min=0) = 1e-5;
  parameter SI.Conductance Goff(final min=0) = 1e-5;
  Real s_sw(final unit="1") "Auxiliary variable";
  constant SI.Current unitCurrent=1 annotation (HideResult=true);
  constant SI.Voltage unitVoltage=1 annotation (HideResult=true);
// "Parámetros de la Resistencia R1"
  parameter SI.Resistance R1 = 14;
// "Parámetros del condensador C1"
  parameter SI.Capacitance C1 = 1e-06;
// "Parámetros del condensador C2"
```

```
33   parameter SI.Capacitance C2 = 1e-06;
34
35 equation
36 // Ecuaciones en los nodos
37   i_C1 + i_sw = i_R1;
38   i_C2 = i_sw;
39   i_R1 = i_gen;
40 // Relaciones constitutivas
41   u1 = Ugen * sin( w_gen * time + phi_gen);
42   u1 - a = i_R1 * R1;
43   i_C1 = C1 * der( u_C1 );
44   a = u_C1;
45   i_C2 = C2 * der( u_C2 );
46   b = u_C2;
47   v_sw = a - b;
48   when sample(0,Tsw) then
49     off_sw = not pre(off_sw);
50   end when;
51   i_sw = (s_sw *unitVoltage)*(if off_sw then Goff else 1);
52   v_sw = (s_sw *unitCurrent)*(if off_sw then 1 else Ron);
53 end Test4Flat;
```

```
 1 model Test4Composite
 2 import Analog = Modelica.Electrical.Analog;
 3 Analog.Sources.SineVoltage gen(V = 5, offset = 0 , f = 10)
 4  annotation(Placement(visible = true, transformation(
 5         origin = { 100, 755},
 6         extent = { { -20, -20}, { 20, 20} },
 7         rotation = 0)));
 8
 9 Modelica.Blocks.Sources.BooleanPulse booleanPulse(period= 1);
10 Analog.Ideal.IdealOpeningSwitch u
11  annotation(Placement(visible = true, transformation(
12         origin = { 530, 759},
13         extent = { { -20, -20}, { 20, 20} },
14         rotation = 0)));
15
16 Analog.Basic.Resistor R1(R = 14)
17  annotation(Placement(visible = true, transformation(
18         origin = { 280, 759},
19         extent = { { -20, -20}, { 20, 20} },
20         rotation = 0)));
21
22 Analog.Basic.Capacitor C1(C = 1E-06)
23  annotation(Placement(visible = true, transformation(
24         origin = { 463, 682},
25         extent = { { -20, -20}, { 20, 20} },
26         rotation = 90)));
27
```

```
28 Analog.Basic.Capacitor C2(C = 1E-06)
29  annotation(Placement(visible = true, transformation(
30         origin = { 670, 682},
31         extent = { { -20, -20}, { 20, 20} },
32         rotation = -270)));
33
34 Analog.Basic.Ground u0
35  annotation(Placement(visible = true, transformation(
36         origin = { 30, 520},
37         extent = { { -20, -20}, { 20, 20} },
38         rotation = 0)));
39
40 Analog.Interfaces.NegativePin a
41  annotation(Placement(transformation(extent ={ { 408,749},{ 428,769} })));
42
43 Analog.Interfaces.NegativePin b
44  annotation(Placement(transformation(extent ={ { 615,745},{ 635,765} })));
45
46 Analog.Interfaces.NegativePin u1
47  annotation(Placement(transformation(extent ={ { 169,749},{ 189,769} })));
48
49
50 equation
51 connect(booleanPulse.y, u.control);
52 connect(R1.n, a) annotation (Line(points ={{280,759},{418,759}}, color ={
   0,0,255}));
53 connect(a, u.p) annotation (Line(points ={{418,759},{530,759}}, color ={
   0,0,255}));
54 connect(a, C1.p) annotation (Line(points ={{418,759},{463,682},{463,682}},
   color ={ 0,0,255}));
55 connect(u.n, b) annotation (Line(points ={{530,759},{625,755},{625,755}},
   color ={ 0,0,255}));
56 connect(b, C2.p) annotation (Line(points ={{625,755},{670,682}}, color ={
   0,0,255}));
57 connect(C1.n, u0.p) annotation (Line(points
   ={{463,682},{463,520},{30,520}}, color ={ 0,0,255}));
58 connect(C2.n, u0.p) annotation (Line(points
   ={{670,682},{670,520},{30,520}}, color ={ 0,0,255}));
59 connect(u0.p, gen.n) annotation (Line(points
   ={{30,520},{100,755},{100,755}}, color ={ 0,0,255}));
60 connect(gen.p, u1) annotation (Line(points
   ={{100,755},{179,759},{179,759}}, color ={ 0,0,255}));
61 connect(u1, R1.p) annotation (Line(points ={{179,759},{280,759}}, color ={
   0,0,255}));
62 annotation(
63         uses(Modelica(version = "4.0.0")),
64         Diagram(coordinateSystem(extent = { { 0, 0}, { 1400, 1400} })),
65         Icon(coordinateSystem(extent = {{0, 0}, {1400, 1400}})),
66         version = "");
67 end Test4Composite;
```

6.6 Modelo de Test 5

Para el modelo del Test 5 hemos elegido el ejercicio planteado en septiembre 2019 como prueba de evaluación continua en la asignatura de Métodos de Simulación y Modelado.

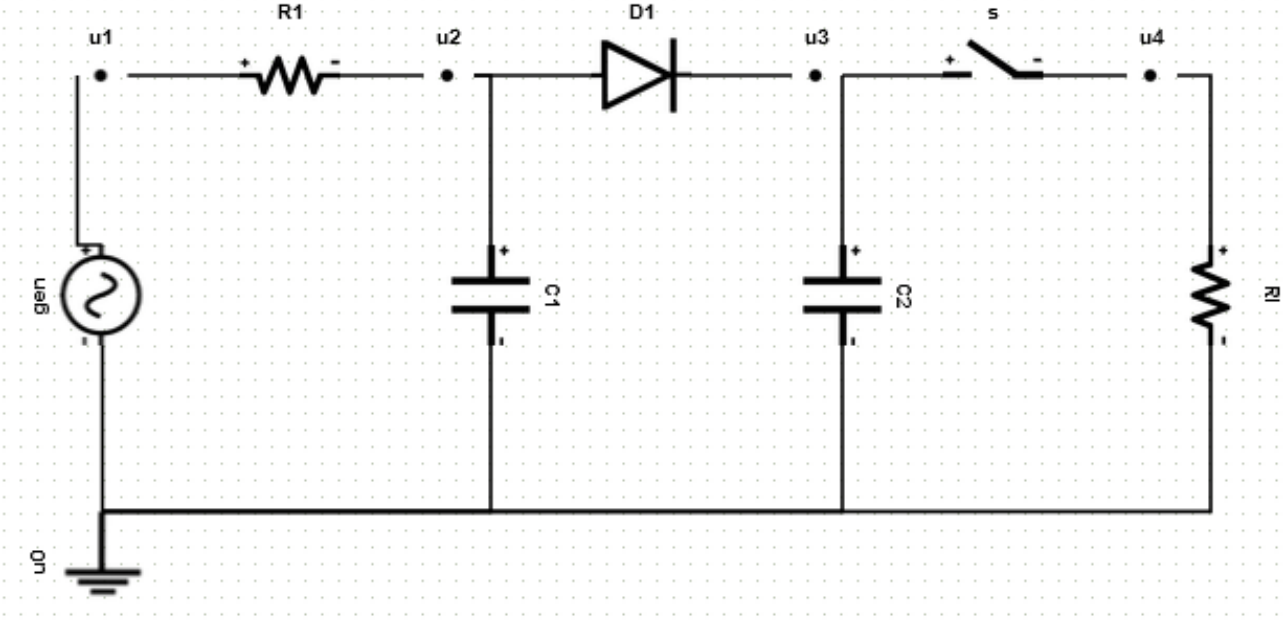


Figura 6-16 Model de Test 5

Este modelo contiene una malla compuesta por un generador sinusoidal de tensión, un diodo, dos condensadores, dos resistencias y un interruptor.

Los valores de las resistencias y capacidades son conocidos y constantes: R1=20 ohm, RL=1000 ohm, C1=10−5 F, C2=0.01 F. El generador sinusoidal de tensión tiene una amplitud de 12 V y una frecuencia de 60 Hz. El diodo es descrito empleando el modelo de Shockley y asignando a sus parámetros los valores siguientes: corriente de saturación, Is=10−9 A; tensión térmica, Vt=0.025 V. El interruptor eléctrico es ideal. Puede encontrarse en dos fases: abierto y cerrado. Mientras está abierto no permite el paso de corriente. Mientras está cerrado la caída de tensión entre sus bornes es cero. El interruptor cambia de fase cada T segundos, siendo T = 0.05 s un parámetro del modelo. A continuación mostramos el código para el modelo atómico

```
model Test5Flat
import SI = Modelica.Units.SI;
  SI.Current i_gen;// " Corriente gen"
  SI.Current i_sw;// "Corriente sw"
  SI.Voltage v_sw;// "Voltaje sw"
  Boolean off_sw(start=false, fixed=true);
  SI.Current i_R1;// " Corriente R1"
  SI.Current i_Rl;// " Corriente Rl"
  SI.Current i_D1(start=0, fixed=false);// " Corriente D1"
  SI.Current i_C1;
  SI.Voltage u_C1(start=0, fixed=true);// "Voltaje C1"
  SI.Current i_C2;
  SI.Voltage u_C2(start=0, fixed=true);// "Voltaje C2"
  SI.Voltage u1;// "Voltaje u1"
  SI.Voltage u2;// "Voltaje u2"
  SI.Voltage u3;// "Voltaje u3"
  SI.Voltage u4;// "Voltaje u4"
// "Parámetros del generador gen"
  parameter SI.Voltage Ugen = 12;
  parameter SI.Frequency frec_gen = 60;
  parameter SI.AngularFrequency w_gen = 2 * Modelica.Constants.pi *
frec_gen;
  parameter SI.Angle phi_gen = 0;
// "Parámetros del interruptor sw"
  parameter SI.Time Tsw = 0.5;
// "Closed switch resistancer sw"
  parameter SI.Resistance Ron(final min=0) = 1e-5;
  parameter SI.Conductance Goff(final min=0) = 1e-5;
  Real s_sw(final unit="1") "Auxiliary variable";
  constant SI.Current unitCurrent=1 annotation (HideResult=true);
  constant SI.Voltage unitVoltage=1 annotation (HideResult=true);
// "Parámetros de la Resistencia R1"
  parameter SI.Resistance R1 = 20;
// "Parámetros de la Resistencia Rl"
  parameter SI.Resistance Rl = 100;
// "Parámetros del diodo D1"
  parameter SI.Current IsD1 = 1e-09;
  parameter SI.Voltage VtD1 = 0.025;
// "Parámetros del condensador C1"
  parameter SI.Capacitance C1 = 1e-06;
// "Parámetros del condensador C2"
  parameter SI.Capacitance C2 = 0.01;

equation
// Ecuaciones en los nodos
  i_R1 = i_gen;
  i_D1 + i_C1 = i_R1;
  i_C2 + i_sw = i_D1;
  i_Rl = i_sw;
// Relaciones constitutivas
```

```
50   u1 = Ugen * sin( w_gen * time + phi_gen);
51   u1 - u2 = i_R1 * R1;
52   u4 = i_Rl * Rl;
53   i_D1  = IsD1 * ( exp((u2-u3) / VtD1) -1);
54   i_C1 = C1 * der( u_C1 );
55   u2 = u_C1;
56   i_C2 = C2 * der( u_C2 );
57   u3 = u_C2;
58   v_sw = u3 - u4;
59   when sample(0,Tsw) then
60     off_sw = not pre(off_sw);
61   end when;
62   i_sw = (s_sw*unitVoltage)*(if off_sw then Goff else 1);
63   v_sw = (s_sw*unitCurrent)*(if off_sw then 1 else Ron);
64 end Test5Flat;
```

Tras ejecutar la simulación del modelo podemos observar cómo se comporta la tensión en el nodo u1, en la Figura 6-15 mostramos su gráfica.

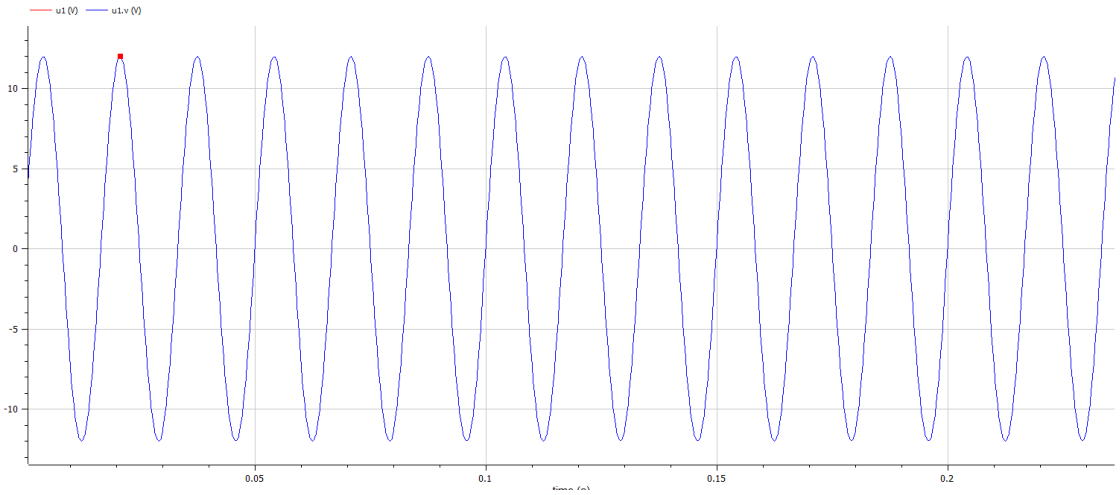


Figura 6-17 Voltaje en el nodo u1

Vemos en la Figura 6-16 la tensión en los nodos u3 y u4, llama la atención como aumenta para u3 de una forma casi lineal y como es escalonada para u4, esta gráfica coincide con la resolución numérica del ejercicio y muestra visualmente el comportamiento del interruptor.

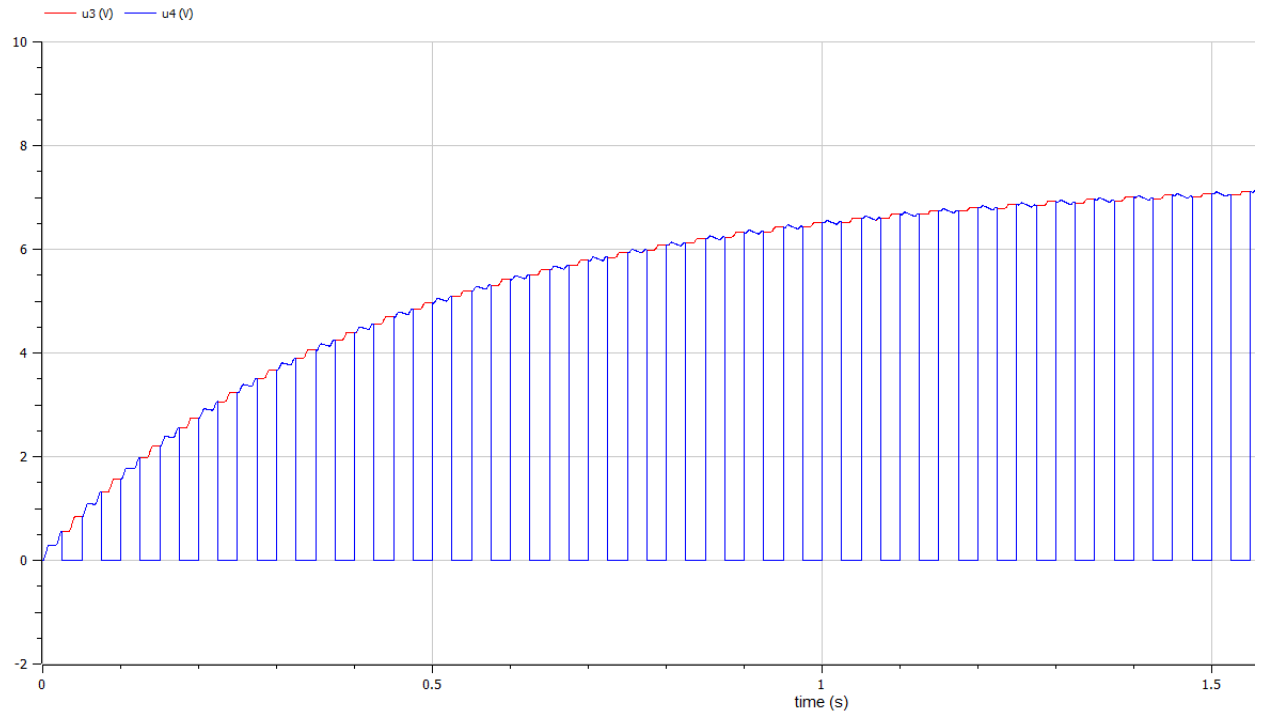


Figura 6-18 Voltaje en los nodos u3 y u4

```
model Test5Composite
import Analog = Modelica.Electrical.Analog;
Analog.Sources.SineVoltage gen(V = 12, offset = 0 , f = 60)
 annotation(Placement(visible = true, transformation(
        origin = { 58, 650},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -90)));

Modelica.Blocks.Sources.BooleanPulse booleanPulse(period= 0,05);
Analog.Ideal.IdealOpeningSwitch s
 annotation(Placement(visible = true, transformation(
        origin = { 590, 781},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Basic.Resistor R1(R = 20)
 annotation(Placement(visible = true, transformation(
        origin = { 170, 781},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Basic.Resistor Rl(R = 100)
```

```
 annotation(Placement(visible = true, transformation(
        origin = { 720, 650},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -270)));

Analog.Semiconductors.Diode  D1(Ids = 1E-09, Vt = 0.025)
 annotation(Placement(visible = true, transformation(
        origin = { 380, 781},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Basic.Capacitor C1(C = 1E-06)
 annotation(Placement(visible = true, transformation(
        origin = { 290, 650},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 90)));

Analog.Basic.Capacitor C2(C = 0.01)
 annotation(Placement(visible = true, transformation(
        origin = { 500, 650},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -270)));

Analog.Basic.Ground u0
 annotation(Placement(visible = true, transformation(
        origin = { 38, 491},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -90)));

Analog.Interfaces.NegativePin u1
 annotation(Placement(transformation(extent ={ { 48,771},{ 68,791} })));

Analog.Interfaces.NegativePin u2
 annotation(Placement(transformation(extent ={ { 255,771},{ 275,791} })));

Analog.Interfaces.NegativePin u3
 annotation(Placement(transformation(extent ={ { 475,771},{ 495,791} })));

Analog.Interfaces.NegativePin u4
 annotation(Placement(transformation(extent ={ { 675,771},{ 695,791} })));


equation
connect(booleanPulse.y, s.control);
connect(u0.p, gen.n) annotation (Line(points ={{58,491},{58,650}}, color
={ 0,0,255}));
connect(gen.p, u1) annotation (Line(points ={{58,650},{58,781}}, color ={
0,0,255}));
connect(u1, R1.p) annotation (Line(points ={{58,781},{170,781}}, color ={
0,0,255}));
```

```
70 connect(R1.n, u2) annotation (Line(points ={{170,781},{265,781}}, color ={
   0,0,255}));
71 connect(u2, D1.p) annotation (Line(points ={{265,781},{380,781}}, color ={
   0,0,255}));
72 connect(D1.n, u3) annotation (Line(points ={{380,781},{485,781}}, color ={
   0,0,255}));
73 connect(u3, s.p) annotation (Line(points ={{485,781},{590,781}}, color ={
   0,0,255}));
74 connect(s.n, u4) annotation (Line(points ={{590,781},{685,781}}, color ={
   0,0,255}));
75 connect(u2, C1.p) annotation (Line(points
   ={{265,781},{290,650},{290,650}}, color ={ 0,0,255}));
76 connect(C1.n, u0.p) annotation (Line(points
   ={{290,650},{290,491},{58,491}}, color ={ 0,0,255}));
77 connect(u3, C2.p) annotation (Line(points ={{485,781},{500,650}}, color ={
   0,0,255}));
78 connect(C2.n, u0.p) annotation (Line(points
   ={{500,650},{500,491},{58,491}}, color ={ 0,0,255}));
79 connect(u4, R1.p) annotation (Line(points ={{685,781},{720,650}}, color ={
   0,0,255}));
80 connect(R1.n, u0.p) annotation (Line(points
   ={{720,650},{720,491},{58,491}}, color ={ 0,0,255}));
81
82 annotation(
83        uses(Modelica(version = "4.0.0")),
84        Diagram(coordinateSystem(extent = { { 0, 0}, { 1400, 1400} })),
85        Icon(coordinateSystem(extent = {{0, 0}, {1400, 1400}}))),
86        version = "");
87
88 end Test5Composite;
```

6.7 Modelo de Test 6

Para el modelo del Test 6 hemos elegido el ejercicio planteado en enero 2017 como prueba de evaluación continua en la asignatura de Métodos de Simulación y Modelado.

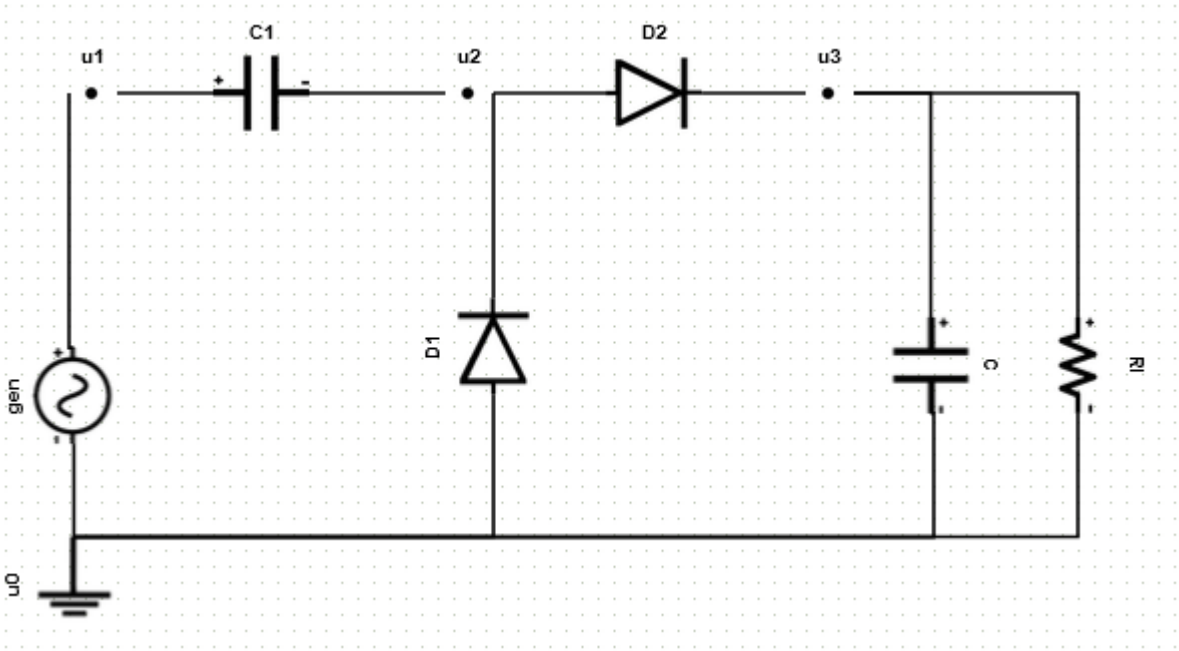


Figura 6-19 Modelo de Test 6

Se muestra un circuito eléctrico compuesto por un generador sinusoidal de tensión, dos diodos, dos condensadores y una resistencia.

Los valores de la resistencia y las capacidades son conocidos y constantes: RL = 1000 ohm, C1 = C2 = 100 μF. El generador sinusoidal de tensión tiene una amplitud de U1 = 120 V y una frecuencia de 60 Hz. Los diodos son descritos empleando el modelo de Shockley. Los parámetros del modelo tienen los valores siguientes: corriente de saturación, Is = 10−9 A; tensión térmica, Vt = 0.025 V.

A continuación mostramos el modelo atómico y compuesto:

```
model Test6Flat
import SI = Modelica.Units.SI;
  SI.Current i_gen;// " Corriente gen"
  SI.Current i_Rl;// " Corriente Rl"
  SI.Current i_D2(start=0, fixed=false);// " Corriente D2"
  SI.Current i_D1(start=0, fixed=false);// " Corriente D1"
  SI.Current i_C1;
  SI.Voltage u_C1(start=0, fixed=true);// "Voltaje C1"
  SI.Current i_C;
  SI.Voltage u_C(start=0, fixed=true);// "Voltaje C"
  SI.Voltage u1;// "Voltaje u1"
  SI.Voltage u2;// "Voltaje u2"
  SI.Voltage u3;// "Voltaje u3"
// "Parámetros del generador gen"
  parameter SI.Voltage Ugen = 120;
  parameter SI.Frequency frec_gen = 60;
  parameter SI.AngularFrequency w_gen = 2 * Modelica.Constants.pi *
frec_gen;
  parameter SI.Angle phi_gen = 0;
// "Parámetros de la Resistencia Rl"
  parameter SI.Resistance Rl = 1000;
// "Parámetros del diodo D2"
  parameter SI.Current IsD2 = 1e-09;
  parameter SI.Voltage VtD2 = 0.025;
// "Parámetros del diodo D1"
  parameter SI.Current IsD1 = 1e-09;
  parameter SI.Voltage VtD1 = 0.025;
// "Parámetros del condensador C1"
  parameter SI.Capacitance C1 = 0.0001;
// "Parámetros del condensador C"
  parameter SI.Capacitance C = 0.0001;

equation
// Ecuaciones en los nodos
  i_C1 = i_gen;
  i_D2 = i_C1 + i_D1;
  i_C + i_Rl = i_D2;
// Relaciones constitutivas
  u1 = Ugen * sin( w_gen * time + phi_gen);
  u3 = i_Rl * Rl;
  i_D2  = IsD2 * ( exp((u2-u3) / VtD2) -1);
  i_D1  = IsD1 * ( exp((-u2) / VtD1) -1);
  i_C1 = C1 * der( u_C1 );
  u1 - u2 = u_C1;
  i_C = C * der( u_C );
  u3 = u_C;
end Test6Flat;
```

```
model Test6Composite
import Analog = Modelica.Electrical.Analog;
Analog.Sources.SineVoltage gen(V = 120, offset = 0 , f = 60)
 annotation(Placement(visible = true, transformation(
        origin = { 53, 526},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -90)));

Analog.Basic.Resistor Rl(R = 1000)
 annotation(Placement(visible = true, transformation(
        origin = { 680, 545},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -270)));

Analog.Semiconductors.Diode  D2(Ids = 1E-09, Vt = 0.025)
 annotation(Placement(visible = true, transformation(
        origin = { 415, 715},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Semiconductors.Diode  D1(Ids = 1E-09, Vt = 0.025)
 annotation(Placement(visible = true, transformation(
        origin = { 315, 557},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -90)));

Analog.Basic.Capacitor C1(C = 1E-06)
 annotation(Placement(visible = true, transformation(
        origin = { 170, 715},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Basic.Capacitor C(C = 1E-06)
 annotation(Placement(visible = true, transformation(
        origin = { 588, 545},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -270)));

Analog.Basic.Ground u0
 annotation(Placement(visible = true, transformation(
        origin = { 33, 408},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -90)));

Analog.Interfaces.NegativePin u1
 annotation(Placement(transformation(extent ={ { 55,705},{ 75,725} })));

Analog.Interfaces.NegativePin u2
 annotation(Placement(transformation(extent ={ { 290,705},{ 310,725}
}})));
```

```
50
51 Analog.Interfaces.NegativePin u3
52  annotation(Placement(transformation(extent ={ { 515,705},{ 535,725}
53 })));

54
55 equation
56 connect(u0.p, gen.n) annotation (Line(points ={{53,408},{53,526}}, color
   ={ 0,0,255}));
57 connect(gen.p, u1) annotation (Line(points ={{53,526},{65,715},{65,715}},
58 color ={ 0,0,255}));
   connect(u2, D2.p) annotation (Line(points ={{300,715},{415,715}}, color
59 ={ 0,0,255}));
   connect(D2.n, u3) annotation (Line(points ={{415,715},{525,715}}, color
60 ={ 0,0,255}));
   connect(u1, C1.p) annotation (Line(points ={{65,715},{170,715}}, color ={
61 0,0,255}));
   connect(C1.n, u2) annotation (Line(points ={{170,715},{300,715}}, color
62 ={ 0,0,255}));
   connect(u3, C.p) annotation (Line(points ={{525,715},{588,545}}, color ={
63 0,0,255}));
   connect(C.n, u0.p) annotation (Line(points
64 ={{588,545},{588,408},{53,408}}, color ={ 0,0,255}));
   connect(D1.n, u2) annotation (Line(points
65 ={{315,557},{300,715},{300,715}}, color ={ 0,0,255}));
   connect(u0.p, D1.p) annotation (Line(points
66 ={{53,408},{315,557},{315,557}}, color ={ 0,0,255}));
   connect(u3, R1.p) annotation (Line(points ={{525,715},{680,545}}, color
67 ={ 0,0,255}));
   connect(R1.n, u0.p) annotation (Line(points
68 ={{680,545},{680,408},{53,408}}, color ={ 0,0,255}));

69 annotation(
70         uses(Modelica(version = "4.0.0")),
71         Diagram(coordinateSystem(extent = { { 0, 0}, { 1400, 1400} })),
72         Icon(coordinateSystem(extent = {{0, 0}, {1400, 1400}})),
73         version = "");
74
75 end Test6Composite;
```

Tras realizar la simulación configurando el número de intervalos a 5000, podemos observar la tensión en los nodos u3 (curva creciente y escalonada), y la tensión en el nodo u1. Se comprueba nuevamente que ambos modelos coinciden sus valores durante las simulaciones.

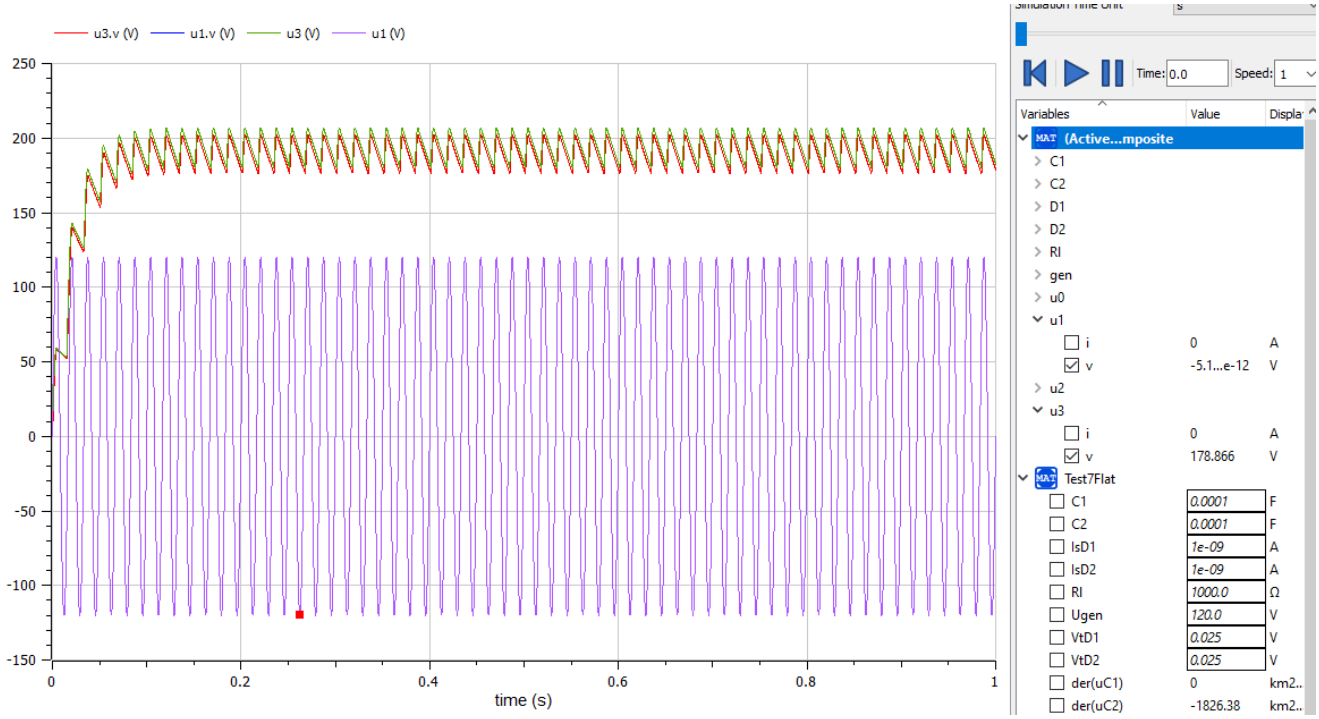


Figura 6-20 Voltaje de distintos nodos con el modelo atómico y compuesto superpuestos

6.8 Modelo de Test 7

El test 7 es una variación del test 6, añadiendo más componentes. En este caso tenemos componentes en serie y paralelo mezclados, y dos fuentes de tensión sinusoidal

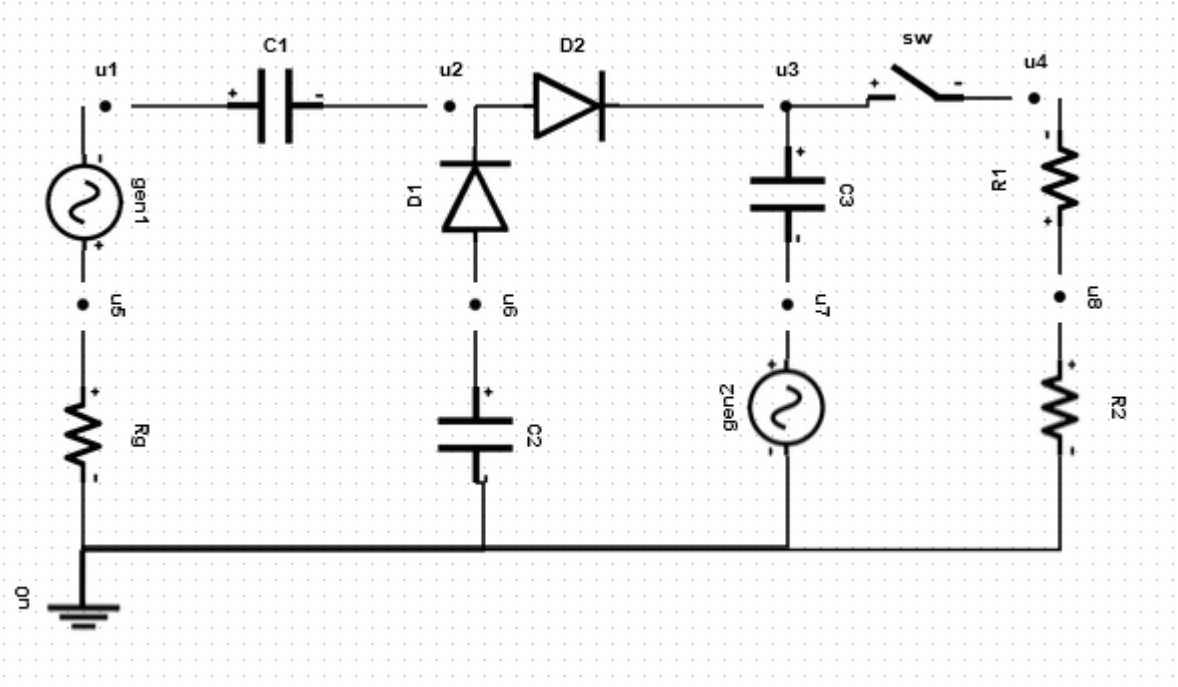


Figura 6-21 Modelo de test 7

El correspondiente código del modelo es el siguiente:

```
model Test7Flat
import SI = Modelica.Units.SI;
  SI.Current i_gen1;// "Corriente gen1"
  SI.Current i_gen2;// "Corriente gen2"
  SI.Current i_sw;// "Corriente sw"
  SI.Voltage v_sw;// "Voltaje sw"
  Boolean off_sw(start=true, fixed=true);
  SI.Current i_R2;// "Corriente R2"
  SI.Current i_Rg;// "Corriente Rg"
  SI.Current i_R1;// "Corriente R1"
  SI.Current i_D2(start=0, fixed=false);// "Corriente D2"
  SI.Current i_D1(start=0, fixed=false);// "Corriente D1"
  SI.Current i_C1;// "Corriente C1"
  SI.Voltage u_C1(start=0, fixed=true);// "Voltaje C1"
  SI.Current i_C3;// "Corriente C3"
  SI.Voltage u_C3(start=0, fixed=true);// "Voltaje C3"
  SI.Current i_C2;// "Corriente C2"
  SI.Voltage u_C2(start=0, fixed=true);// "Voltaje C2"
  SI.Voltage u1;// "Voltaje u1"
  SI.Voltage u2;// "Voltaje u2"
  SI.Voltage u3;// "Voltaje u3"
  SI.Voltage u5;// "Voltaje u5"
  SI.Voltage u6;// "Voltaje u6"
  SI.Voltage u7;// "Voltaje u7"
  SI.Voltage u4;// "Voltaje u4"
  SI.Voltage u8;// "Voltaje u8"
// "Parámetros del generador gen1"
  parameter SI.Voltage Ugen1 = 120;
  parameter SI.Frequency frec_gen1 = 60;
  parameter SI.AngularFrequency w_gen1 = 2 * Modelica.Constants.pi *
frec_gen1;
  parameter SI.Angle phi_gen1 = 0;
// "Parámetros del generador gen2"
  parameter SI.Voltage Ugen2 = 5;
  parameter SI.Frequency frec_gen2 = 100;
  parameter SI.AngularFrequency w_gen2 = 2 * Modelica.Constants.pi *
frec_gen2;
  parameter SI.Angle phi_gen2 = 0;
// "Parámetros del interruptor sw"
  parameter SI.Time Tsw = 0.5;
// "Closed switch resistancer sw"
  parameter SI.Resistance Ron(final min=0) = 1e-5;
  parameter SI.Conductance Goff(final min=0) = 1e-5;
  Real s_sw(final unit="1") "Auxiliary variable";
  constant SI.Current unitCurrent=1 annotation (HideResult=true);
  constant SI.Voltage unitVoltage=1 annotation (HideResult=true);
// "Parámetros de la Resistencia R2"
  parameter SI.Resistance R2 = 1000;
// "Parámetros de la Resistencia Rg"
  parameter SI.Resistance Rg = 100;
```

```
49 // "Parámetros de la Resistencia R1"
50   parameter SI.Resistance R1 = 100;
51 // "Parámetros del diodo D2"
52   parameter SI.Current IsD2 = 1e-09;
53   parameter SI.Voltage VtD2 = 0.025;
54 // "Parámetros del diodo D1"
55   parameter SI.Current IsD1 = 1e-09;
56   parameter SI.Voltage VtD1 = 0.025;
57 // "Parámetros del condensador C1"
58   parameter SI.Capacitance C1 = 0.0001;
59 // "Parámetros del condensador C3"
60   parameter SI.Capacitance C3 = 0.0001;
61 // "Parámetros del condensador C2"
62   parameter SI.Capacitance C2 = 1e-06;
63
64 equation
65 // Ecuaciones en los nodos
66   i_C1 + i_gen1 = 0;
67   i_D2 = i_C1 + i_D1;
68   i_C3 + i_sw = i_D2;
69   i_Rg = i_gen1;
70   i_C2 + i_D1 = 0;
71   0 = i_C3 + i_gen2;
72   0 = i_R1 + i_sw;
73   i_R1 + i_R2 = 0;
74 // Relaciones constitutivas
75   u5 - u1 = Ugen1 * sin( w_gen1 * time + phi_gen1);
76   u7 = Ugen2 * sin( w_gen2 * time + phi_gen2);
77   u8 = i_R2 * R2;
78   u5 = i_Rg * Rg;
79   u8 - u4 = i_R1 * R1;
80   i_D2  = IsD2 * ( exp((u2 - u3) / VtD2) -1);
81   i_D1  = IsD1 * ( exp((u6 - u2) / VtD1) -1);
82   i_C1 = C1 * der( u_C1 );
83   u1 - u2 = u_C1;
84   i_C3 = C3 * der( u_C3 );
85   u3 - u7 = u_C3;
86   i_C2 = C2 * der( u_C2 );
87   u6 = u_C2;
88   v_sw = u3 - u4;
89   when sample(0,Tsw) then
90     off_sw = not pre(off_sw);
91   end when;
92   i_sw = (s_sw*unitVoltage)*(if off_sw then Goff else 1);
93   v_sw = (s_sw*unitCurrent)*(if off_sw then 1 else Ron);
94 end Test7Flat;
```

```
model Test7Composite
import Analog = Modelica.Electrical.Analog;
Analog.Sources.SineVoltage gen1(V = 120, offset = 0 , f = 60)
 annotation(Placement(visible = true, transformation(
        origin = { 70, 655},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -270)));

Analog.Sources.SineVoltage gen2(V = 5, offset = 0 , f = 100)
 annotation(Placement(visible = true, transformation(
        origin = { 510, 527},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -90)));

Modelica.Blocks.Sources.BooleanPulse booleanPulse(period= 1);
Analog.Ideal.IdealOpeningSwitch sw
 annotation(Placement(visible = true, transformation(
        origin = { 590, 720},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Basic.Resistor R2(R = 1000)
 annotation(Placement(visible = true, transformation(
        origin = { 680, 527},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -270)));

Analog.Basic.Resistor Rg(R = 100)
 annotation(Placement(visible = true, transformation(
        origin = { 70, 510},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -270)));

Analog.Basic.Resistor R1(R = 100)
 annotation(Placement(visible = true, transformation(
        origin = { 680, 670},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -90)));

Analog.Semiconductors.Diode  D2(Ids = 1E-09, Vt = 0.025)
 annotation(Placement(visible = true, transformation(
        origin = { 375, 715},
        extent = { { -20, -20}, { 20, 20} },
        rotation = 0)));

Analog.Semiconductors.Diode  D1(Ids = 1E-09, Vt = 0.025)
 annotation(Placement(visible = true, transformation(
        origin = { 315, 660},
        extent = { { -20, -20}, { 20, 20} },
        rotation = -90)));
```

```
51
52 Analog.Basic.Capacitor C1(C = 0.0001)
53  annotation(Placement(visible = true, transformation(
54         origin = { 190, 715},
55         extent = { { -20, -20}, { 20, 20} },
56         rotation = 0)));
57
58 Analog.Basic.Capacitor C3(C = 0.0001)
59  annotation(Placement(visible = true, transformation(
60         origin = { 510, 660},
61         extent = { { -20, -20}, { 20, 20} },
62         rotation = -270)));
63
64 Analog.Basic.Capacitor C2(C = 1E-06)
65  annotation(Placement(visible = true, transformation(
66         origin = { 315, 510},
67         extent = { { -20, -20}, { 20, 20} },
68         rotation = -270)));
69
70 Analog.Basic.Ground u0
71  annotation(Placement(visible = true, transformation(
72         origin = { 50, 408},
73         extent = { { -20, -20}, { 20, 20} },
74         rotation = -90)));
75
76 Analog.Interfaces.NegativePin u1
77  annotation(Placement(transformation(extent ={ { 75,705},{ 95,725} })));
78
79 Analog.Interfaces.NegativePin u2
80  annotation(Placement(transformation(extent ={ { 290,705},{ 310,725}
81 })));
82
83 Analog.Interfaces.NegativePin u3
84  annotation(Placement(transformation(extent ={ { 500,705},{ 520,725}
85 })));
86
87 Analog.Interfaces.NegativePin u5
88  annotation(Placement(transformation(extent ={ { 60,580},{ 80,600} })));
89
90 Analog.Interfaces.NegativePin u6
91  annotation(Placement(transformation(extent ={ { 305,580},{ 325,600}
92 })));
93
94 Analog.Interfaces.NegativePin u7
95  annotation(Placement(transformation(extent ={ { 500,580},{ 520,600}
96 })));
97
98 Analog.Interfaces.NegativePin u4
99  annotation(Placement(transformation(extent ={ { 655,710},{ 675,730}
100 })));
```

```
101
102 Analog.Interfaces.NegativePin u8
103  annotation(Placement(transformation(extent ={ { 670,585},{ 690,605}
104 })));
105
106
107 equation
108 connect(booleanPulse.y, sw.control);
109 connect(u0.p, gen2.n) annotation (Line(points
    ={{70,408},{510,408},{510,527}}, color ={ 0,0,255}));
110 connect(gen2.p, u7) annotation (Line(points ={{510,527}}, color ={
    0,0,255}));
111 connect(R2.n, u0.p) annotation (Line(points
    ={{680,527},{680,408},{70,408}}, color ={ 0,0,255}));
112 connect(Rg.n, u0.p) annotation (Line(points ={{70,510},{70,408}}, color
    ={ 0,0,255}));
113 connect(C2.n, u0.p) annotation (Line(points
    ={{315,510},{315,408},{70,408}}, color ={ 0,0,255}));
114 connect(u6, C2.p) annotation (Line(points ={{315,590}}, color ={
    0,0,255}));
115 connect(D1.p, u6) annotation (Line(points ={{315,660}}, color ={
    0,0,255}));
116 connect(D1.n, u2) annotation (Line(points
    ={{315,660},{300,715},{300,715}}, color ={ 0,0,255}));
117 connect(u2, D2.p) annotation (Line(points ={{300,715},{375,715}}, color
    ={ 0,0,255}));
118 connect(D2.n, u3) annotation (Line(points ={{375,715},{510,715}}, color
    ={ 0,0,255}));
119 connect(u3, C3.p) annotation (Line(points ={{510,715}}, color ={
    0,0,255}));
120 connect(C3.n, u7) annotation (Line(points ={{510,660}}, color ={
    0,0,255}));
121 connect(u3, sw.p) annotation (Line(points
    ={{510,715},{590,720},{590,720}}, color ={ 0,0,255}));
122 connect(sw.n, u4) annotation (Line(points ={{590,720},{665,720}}, color
    ={ 0,0,255}));
123 connect(u4, R1.n) annotation (Line(points
    ={{665,720},{680,670},{680,670}}, color ={ 0,0,255}));
124 connect(C1.n, u2) annotation (Line(points ={{190,715},{300,715}}, color
    ={ 0,0,255}));
125 connect(u1, C1.p) annotation (Line(points ={{85,715},{190,715}}, color ={
    0,0,255}));
126 connect(gen1.n, u1) annotation (Line(points
    ={{70,655},{85,715},{85,715}}, color ={ 0,0,255}));
127 connect(gen1.p, u5) annotation (Line(points ={{70,655}}, color ={
    0,0,255}));
128 connect(u5, Rg.p) annotation (Line(points ={{70,590}}, color ={
    0,0,255}));
129 connect(u8, R1.p) annotation (Line(points ={{680,595},{680,670}}, color
    ={ 0,0,255}));
```

```
130 connect(u8, R2.p) annotation (Line(points ={{680,595}}, color ={
    0,0,255}));
131
132 annotation(
133         uses(Modelica(version = "4.0.0")),
134         Diagram(coordinateSystem(extent = { { 0, 0}, { 1400, 1400} })),
135         Icon(coordinateSystem(extent = {{0, 0}, {1400, 1400}})),
136         version = "");
137
138 end Test7Composite;
```

Analizamos la tensión en el condensador C2 y en el nodo u5, vemos que ambos modelos coinciden. Figuras 6-22 y 6-23.

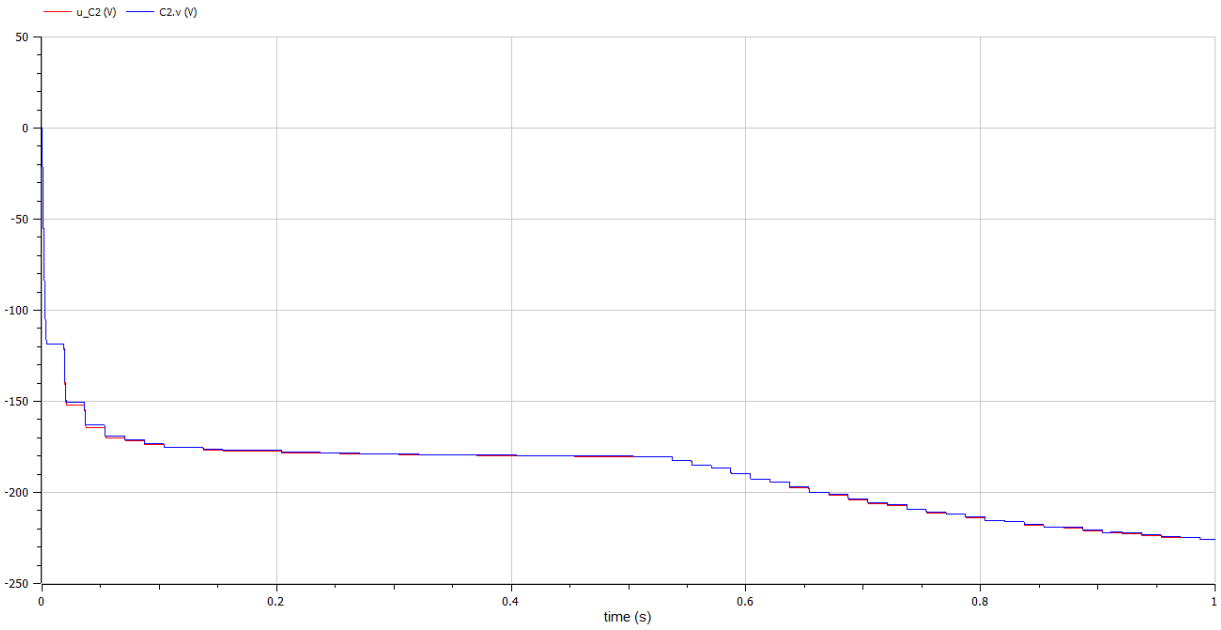


Figura 6-22 Voltaje en el condensador C2

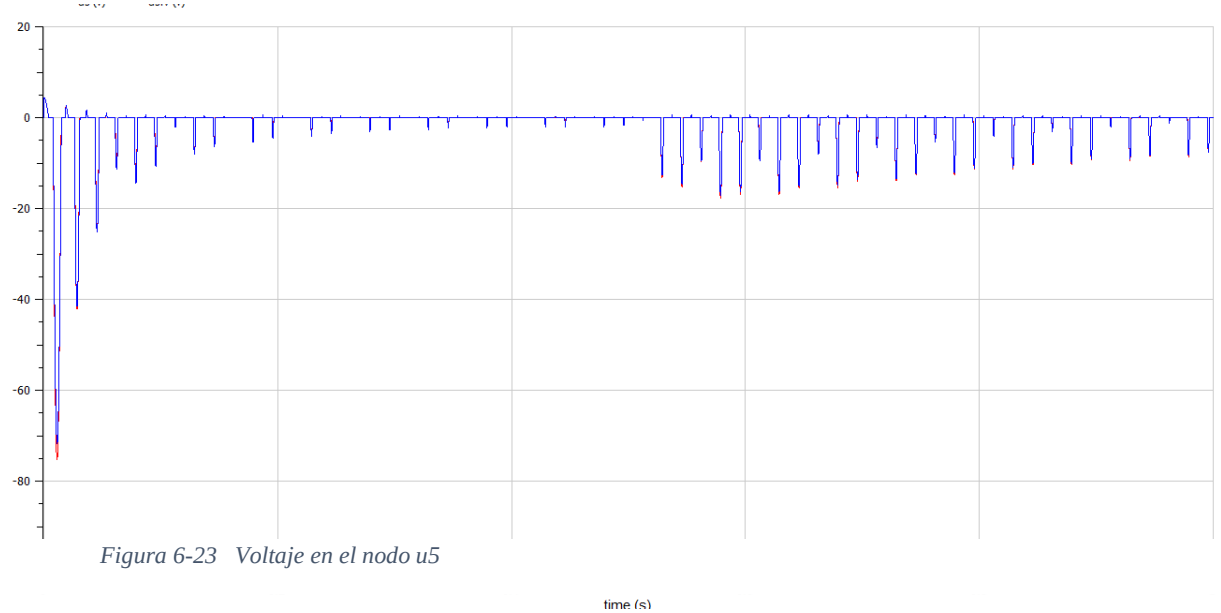


Figura 6-23 Voltaje en el nodo u5

6.9 Validación de requisitos

Para la validación del resto de funcionales de la aplicación hemos usado la Tabla 3-1 y hemos realizado distintos test que verificaran el funcionamiento correcto de cada registro de la tabla. La Tabla 6-1 muestra los mismos requisitos funcionales verificados con casos de prueba y test, la descripción de estos test se muestra en la tercera columna a la derecha, todos estos test se han hecho depurando la aplicación con distintos modelos.

Tabla 6-1 Verificación de requisitos funcionales

Requisitos funcionales		Pruebas unitarias
RF1 – Gestión de usuarios		• Se ha registrado un usuario • a continuación se ha realizado un login con el usuario correctamente. • Se ha cerrado la sesión del usuario. • Se ha realizado un cambio de contraseña verificando que se recibe un email de confirmación y se ha procedido de nuevo a realizar login con la nueva contraseña
	RF1-1 Registro de usuarios	
	RF1-2 Login de usuarios	
	RF1-3 Recuperación de contraseña	
RF2 – Gestión de trabajos		• Se crea un proyecto • Se accede al proyecto • Se renombra el proyecto • Se elimina el proyecto • Dentro de un proyecto se crea un circuito • Se accede al circuito y se renombra • Se elimina el circuito
	RF2-1 _Gestión de proyectos	
	RF2-2 Gestión de circuitos	
RF3 – Gestión de modelos		
	RF3-1 Guardado de modelos	• Los cambios en los modelos son persistidos en base de datos
	RF3-2 edición de modelos	• Se puede acceder a los modelos y realizar modificaciones verificando que se graban los cambios en la base de datos
RF4 – Construir un editor		
	RF4-1 Crear paleta de componentes	• La paleta de elementos muestra todos los componentes desarrollados • Es posible seleccionarlos y arrastrarlos al lienzo
	RF4-2 Propiedades de componentes	• Al seleccionar un componente se pueden editar sus propiedades verificando como se guardan los valores en el modelo
	RF4-3 Conexiones entre componentes	• Los componentes se pueden conectar en el lienzo, estableciendo distintas combinaciones entre los pines

	RF4-4 Barra de herramientas del modelo	• Es posible hacer y deshacer cambios • Verificamos que se puede cargar un modelo externo y salvarlo
RF5 – Comprobaciones		
	RF5-1 Comprobaciones de conexiones	• Se construyen distintos modelos, donde se verifican las conexiones ○ Toma de tierra con generador ○ Toma de tierra con generador y resistencia ○ Generador con diodo y condensador ○ Generador con interruptor y condensador
	RF5-2 Comprobaciones de fuentes	• Verificado en los test mediante modelos descritos con distintas fuentes en paralelo y en serie
	RF5-3 Comprobaciones de nodos	• Se crean modelos sin nodos y vemos el mensaje de comprobación
	RF5-4 Comprobaciones nodo tierra	• Todos los circuitos deben tener un único nodo tierra. Los componentes solo pueden conectarse a un nodo o a tierra.
RF6 -Generación de código		
	RF6-1 Código en formato atómico	• Verificado en los puntos 6.1 – 6.5
	RF6-2 Código en formato compuesto	• Verificado en los puntos 6.1 – 6.5
	RF6-3 Posicionamiento de elementos	• Verificado en los puntos 6.1 – 6.5

6.10 Conclusiones

Hemos realizado 7 experimentos en este capítulo, para intentar demostrar como el modelo implementado de forma gráfica en nuestra aplicación, genera un código totalmente funcional para simulación, se ha intentado demostrar el comportamiento de la tensión y la corriente en los modelos generados explicando de manera práctica lo que sobre un marco teórico debería funcionar, los teoremas de Kirchhoff o de Thévenin han servido de base para estos ejercicios.

Los puntos clave a verificar en estos test han sido:

- El código generado se genera sin errores de sintaxis
- Los modelos en su versión plana como compuesta son equivalentes
- El comportamiento de los componentes por separado
- Comprobamos como cambios introducidos en el modelo pueden afectar al conjunto

7 Conclusiones y trabajos futuros

7.1 Introducción

En este capítulo se hace una revisión de todo el proyecto en su conjunto, de los resultados obtenidos y de los puntos a mejorar en la aplicación.

7.2 Conclusiones

El objetivo de este trabajo era realizar una aplicación, en la cual un usuario fuera capaz de generar modelos eléctricos de forma gráfica y a su vez, generar una traducción a código Modelica. Para ello, hemos tenido que descomponer el problema y seguir una serie de pasos para que la aplicación pudiera ser funcional y estuviera bien estructurada.

En primer lugar, hemos hecho una revisión de los principales programas usados en modelado y simulación, con especial detenimiento en el lenguaje Modelica. Además, hemos hecho una revisión de los hitos tecnológicos que han producido en la simulación a lo largo de la historia. Hemos analizado distintas herramientas que usan el lenguaje Modelica como base para realizar simulaciones.

La arquitectura tecnológica ha sido un factor clave a definir desde el inicio del proyecto. Desde un principio se pensó en una solución basada en un entorno web, con lo que se hacía obligatorio prestar atención a la arquitectura cliente servidor y a un patrón de diseño como Modelo-Vista-Controlador. Una solución de tipo web aportará una serie de ventajas tanto en el diseño como en la usabilidad. Las tecnologías web nos permiten usar herramientas gráficas de codificación de bloques y nos dan un producto en un entorno accesible y colaborativo.

La definición de una buena interfaz de usuario, amigable e intuitiva es una característica que debe estar presente en cualquier aplicación web. Además de la accesibilidad, el lenguaje JavaScript nos proporciona una interacción con los elementos de esta interfaz, haciendo posible la gestión de eventos, la creación de objetos dinámicamente y su manipulación. Gracias a JavaScript podemos definir un editor capaz de arrastrar elementos desde una paleta y formar un modelo con conexiones y propiedades.

Para el cálculo de una traducción a Modelica del modelo atómico nos hemos basado en los principios matemáticos y físicos que rigen los circuitos analógicos. Ha sido muy importante explicar el comportamiento de la tensión y las corrientes basándonos en las reglas de Kirchhoff o

Thévenin y generar un sistema de ecuaciones coherente con el modelo. Para esta tarea , además, han sido muy importantes definir una serie de comprobaciones sobre el modelo.

La traducción al modelo compuesto es viable gracias a la compilación de objetos y funciones predefinidas de la librería Standard de Modelica, las interfaces de objetos definidas en la librería hacen que la instanciación y conexión sea muy fácil. Nuestra aplicación instanciará un objeto por cada elemento definido en nuestra interfaz gráfica, asignando correctamente las propiedades tanto de comportamiento como de visualización, las conexiones entre elementos tendrán en cuenta los pines positivos y negativos para generar un modelo completo en Modelica y poder realizar una simulación completa.

Para la fase de test hemos seguido la estrategia de interpretar valores lógicos de tensión y corriente aplicando los principios matemáticos a los que hacíamos referencia cuando hablábamos del modelo atómico. La aplicación desarrollada hace que la explicación numérica y el desarrollo matemático de una malla eléctrica sea una potente herramienta para un docente o un profesional que necesite resolver y comprobar la validez de un problema propuesto en el dominio eléctrico. Gracias a esta herramienta se puede obtener una mayor compresión del funcionamiento de un modelo eléctrico y de cómo se descompone el problema de forma matemática.

7.3 Trabajos futuros

Aunque el proyecto es totalmente funcional en su versión más básica, tiene muchos puntos que pueden ser mejorados de cara a una versión futura, algunos de estos puntos son los que comento a continuación.

- La implementación de más componentes eléctricos, tanto analógicos como digitales, si bien la aplicación trabaja con elementos de dos pines, se podría implementar la lógica para definir, por ejemplo distintos tipos de transistores, puertas lógicas, sensores, etc.
- Sería interesante la definición de objetos estándar desde cero por parte del usuario, de tal manera que a partir de unas ecuaciones, imágenes y propiedades que defina el usuario se pudieran crear componentes nuevos y reutilizables en la aplicación.
- También sería un buen punto de mejora la posibilidad de definir elementos que sean composiciones de otros modelos, como una caja negra y que pudieran ser conectados a otro modelo.
- En lo que respecta a la definición de las propiedades de los componentes, los valores iniciales son fijos y predefinidos, por ejemplo, una resistencia siempre se creara inicialmente con un valor de 100 ohmios, sería interesante que estos valores iniciales se pudieran configurar.

Ya sea en conjunto o por lotes, todas estas mejoras podrían implementarse en las posteriores iteraciones de desarrollo de este proyecto.

Bibliografía

Acebes L., Alves R., Merino A., & de Prada C. . (2010). *Un entorno de modelado inteligente y simulación distribuida de plantas de proceso.* Valencia: Revista Iberoamericana de Automática e Informática Industrial, Valencia.

Alonso, G. (2012). *Simulación de sistemas de control continuos con SIMULINK*. Obtenido de Universidad de Oviedo - Ingeniería de Sistemas y Automática: http://isa.uniovi.es/~alonsog/Regulacion/PL%2006%20Simulacion%20de%20sistemas%20de%20control%20continuo%20con%20SIMULINK.pdf

Åström, K., Elmqvist, H., & Mattsson, S. (1998). *Evolution of continuous-time modeling and simulation, The 12th European Simulation Multiconference.* Manchester, UK.

Cellier, F. E. (1991). *Continuous System Modeling.* Tucson, Universidad de Arizona: Springer-Verlag.

Craik, K. H. (1968). *The Comprehension of The Everyday Physical Environment.* Whasington, DC: Journal of the American Institute of planners.

Dassault Systèmes. (2023). *Dassault Systèmes*. Obtenido de https://www.3ds.com/es/productos-y-servicios/catia/productos/dymola/

De la Torre Llorente, C. C. (2010). *Guía de Arquitectura N-Capas orientada al Dominio con .NET*. Madrid: Krasis Consulting, S. L.

Elmqvist, H. (1977). SIMNON - An Interactive Simulation Program for Non-Linear Systems. *Simulation '77 : Proceedings of the international symposium* (págs. 85-89). Montreaux: M. H. Hamza.

Elmqvist, H. (2014). *Modelica Evolution - From My Perspective*. Obtenido de LiU Electronic Press: https://ep.liu.se/konferensartikel.aspx?series=ecp&issue=96&Article_No=1

ESI. (2022). *ESI Group -SimulationX*. Obtenido de ESI Group -SimulationX: https://www.esi-group.com/products/system-simulation

Foundation, A. S. (2022). *Apache License, Version 2.0*. Obtenido de https://www.apache.org/licenses/LICENSE-2.0

Fritzson, P. (2003). *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. California: Wiley-IEEE Press.

Fritzson, P. (2011). *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica.* California: Wiley-IEEE Press.

Fritzson, P. A. (2005). *The OpenModelica modeling, simulation, and development environment.* Trondheim, Norway: In 46th Conference on Simulation and Modelling of the Scandinavian Simulation Society (SIMS2005).

Fritzson, P. A. (2006). *OpenModelica-A free open-source environment for system modeling, simulation, and teaching.* Munich, Germany: In 2006 IEEE Conference on Computer Aided Control System Design.

Garipov, E. M. (1997). *Improvement of the Student Knowledge on MATLAB/SIMULINK by Program Examples in System Identification.* Sofia, Bulgaria: IFAC Proceedings Volumes, 77-82.

Hassan, Y. M. (2004). *Diseño web centrado en el usuario: usabilidad y arquitectura de la información.* Madrid: Hipertext. net.

He, F. J. (2016). *Network modeling and visualization platform based on moodle and mxgraph.* Zhengzhou, China: In 3d International Conference on Applied Social Science Research (ICASSR 2015). Atlantis Press.

Huntsinger, R. C. (1988). *Continuous System Simulation Languages (CSSL's).* San Diego, CA, USA: 1988 Winter simulation conference Proceedings.

JGraph. (2006-2017). *mxGraph*. Obtenido de https://jgraph.github.io/mxgraph/

Klir, G. J. (1985). *Architecture of systems complexity.* New York: Saunders New York.

Luján-Mora, S. (2002). *Programación de aplicaciones web: historia, principios básicos y clientes web.* Editorial Club Universitario.

Martin, C., Urquía, A., & Dormido, S. (2009). *Object-oriented modelling of virtual-laboratories for control education. In Web-based control and robotics education.* Madrid: Springer.

Modelica Association. (2023). *Modelica Association*. Obtenido de Modelica Language Documents - Version 3.5 - February 2021: https://modelica.org/

Morten Blomhøj, T. H. (2003). *Developing mathematical modelling competence: Conceptual clarification and educational planning.* Oxford: Teaching mathematics and its applications, Oxford University Press.

OpenModelica. (2023). *OpenModelica*. Obtenido de OpenModelica: https://openmodelica.org/?id=78:omconnectioneditoromedit&catid=10:main-category

Ramirez, W. F. (1997). *Computational methods for process simulation.* Boulder, Colorado: Butterworth-Heinemann.

Rumbaugh, J. (2005). *The unified modeling language reference manual.* India: Pearson Education India.

Software, N. (1998-2022). *GoJS*. Obtenido de A Web Framework for Rapidly Building Interactive Diagrams: https://gojs.net/latest/index.html

Urquía, A., & Martín, C. (2013). *Modelado y simulación de eventos discretos.* Madrid: Universidad Nacional de Educación a Distancia–UNED.

Urquía, A., & Martín, C. (2016). *Métodos de simulación y modelado.* Madrid: Universidad Nacional de Educación a Distancia–UNED.

Glosario

.

.NET: plataforma de código abierto para crear aplicaciones de escritorio, web y móviles que se pueden ejecutar de forma nativa en cualquier sistema operativo.

A

ACSL: Advance Continuous Simulation Language, lenguaje de simulación

B

Backend: el backend es la parte del desarrollo web que se encarga de que toda la lógica de una página web funcione

C

CAD: diseño asistido por computadora

CFD: Dinámica Computacional de Fluídos

CSSL: Continuous System Simulation Languages, lenguaje de simulación

D

DAE: ecuaciones algebraico diferenciales

Drag and drop: en las interfaces gráficas de usuario de la computadora, arrastrar y soltar es un gesto de dispositivo señalador en el que el usuario selecciona un objeto virtual "agarrándolo" y arrastrándolo a una ubicación diferente oa otro objeto virtual.

F

Frontend: es la parte de una aplicación que interactúa con los usuarios, es conocida como el lado del cliente

I

IDE: entorno de desarrollo y programación

M

MATLAB: es un sistema de cómputo numérico que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio

Modelica: lenguaje de modelado multidominio, declarativo y orientado a objetos para el modelado orientado a componentes de sistemas complejos

O

ODE: ecuaciones diferenciales ordinarias

OpenModelica: entorno gratuito y de código abierto basado en el lenguaje de modelado Modelica para modelar, simular, optimizar y analizar sistemas dinámicos complejos

Q

QCAD: aplicación informática de diseño asistido por computadora para diseño 2D

S

SIMULINK: lenguaje de modelado usado para MATLAB

Anexo A: Manual de usuario

Este anexo está relacionado con el Capítulo 5 “Construcción del frontend”. En este anexo profundizaremos en la forma en la cual el usuario interactúa con la aplicación.

A-1 Instalación

Para poder instalar la aplicación, es necesario disponer de un servidor Internet Information Services (IIS), disponible en cualquier edición del sistema operativo Windows. Se debe configurar un nuevo espacio web y publicar todo el contenido del proyecto en este espacio. Los pasos concretos de la instalación son los siguientes:

1. Abrir servidor Internet Information Services (IIS)
2. Agregar sitio web
3. Configurar carpeta de la aplicación y puerto y darle a aceptar
4. Abrir SQL Server Management Studio y crear una nueva base de datos, se puede hacer con el comando CREATE DATABASE <nombre>
5. Restaurar el fichero BDTFM.bak en la nueva base de datos
6. Volver a la carpeta de la publicación y editar el fichero appsettings.json, configurar la cadena de conexión apuntando la base de datos correcta.

Si todo funciona correctamente, se debe poder visualizar la pantalla de presentación en el puerto configurado en el IIS.

En primer lugar tenemos una pantalla de presentación de la aplicación en la que vemos algunos pantallazos y una breve introducción del fin de la herramienta.

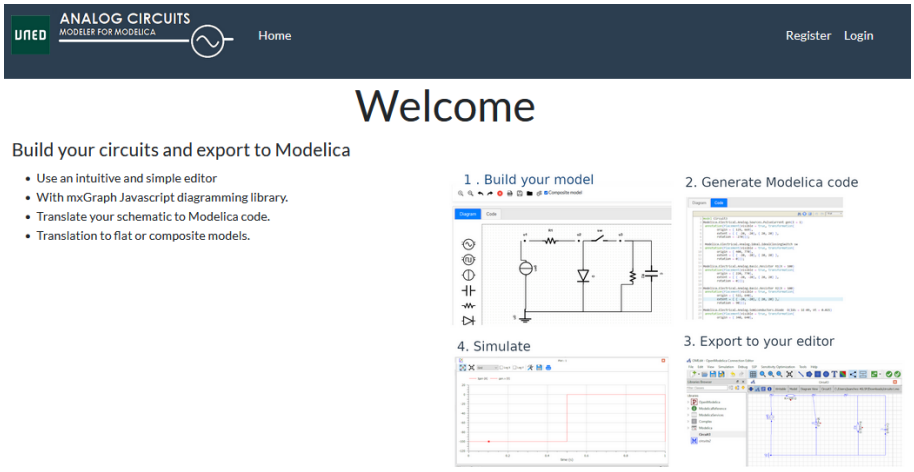


Figura A-1 Página de bienvenida

A-2 Gestión de usuarios

En la parte superior vemos la administración de usuarios, una página de registro de usuarios nuevos y una página de login para usuarios ya registrados. En la página de registro solo tenemos que rellenar un email válido y un password con su confirmación. Las contraseñas son encriptadas antes de almacenarse bajo un algoritmo de cifrado, llamado PBKDF2, del framework Identity de ASP.NET.

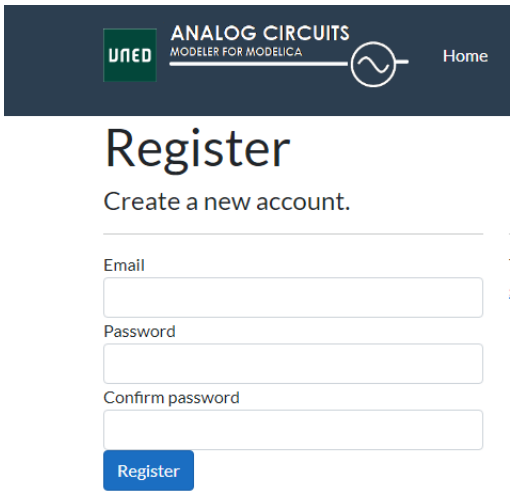


Figura A-2 Página de registro de usuario nuevo

El login de la aplicación consiste en introducir el email y la contraseña de usuario, en caso de éxito se entrará en la sesión donde se cargarán los proyectos y circuitos guardados por el usuario. Además. Habrá una opción para recuperar la contraseña de usuario en caso de perdida.

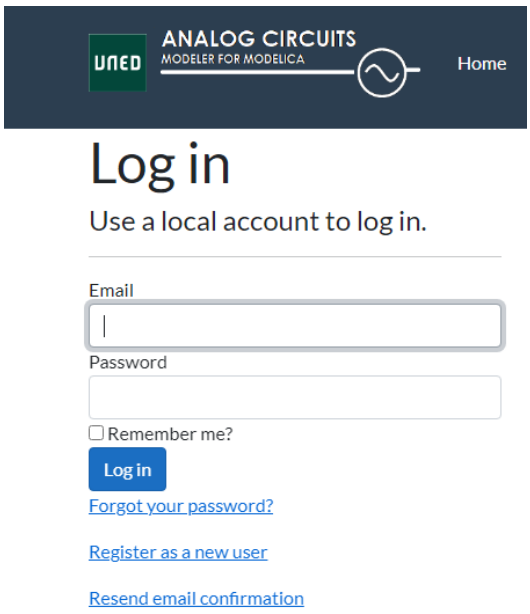


Figura A-3 Página de login

Al realizar el login correctamente se podrá visualizar en la barra superior la entrada "proyectos". Desde donde podremos, crear, eliminar y editar nuevos proyectos, este será el "contenedor" de los circuitos o modelos con los que trabajaremos.

A-3 Gestión de proyectos

Una vez el usuario está logueado, se dispone de una interfaz para crear y gestionar carpetas de proyectos. Desde el menú superior podemos acceder a la opción "Proyectos"

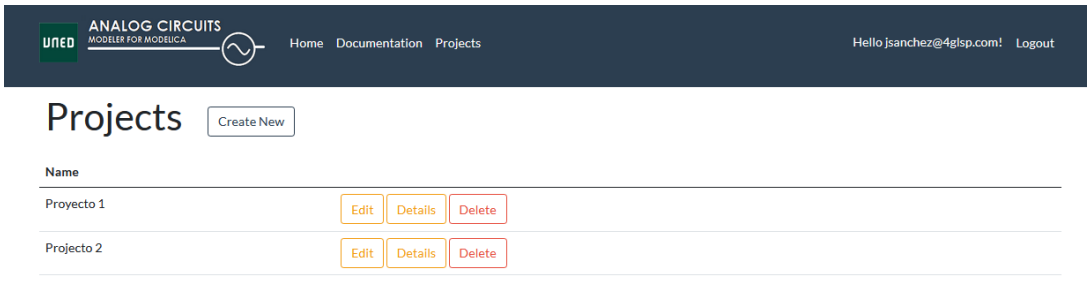


Figura A-4 Página de visualización de proyectos

Para dar de alta un proyecto, bastará con pinchar en “Crear proyecto nuevo” y asignarle un nombre. Internamente este proyecto se guardará en base de datos, el nombré podrá modificars econ posterioridad.

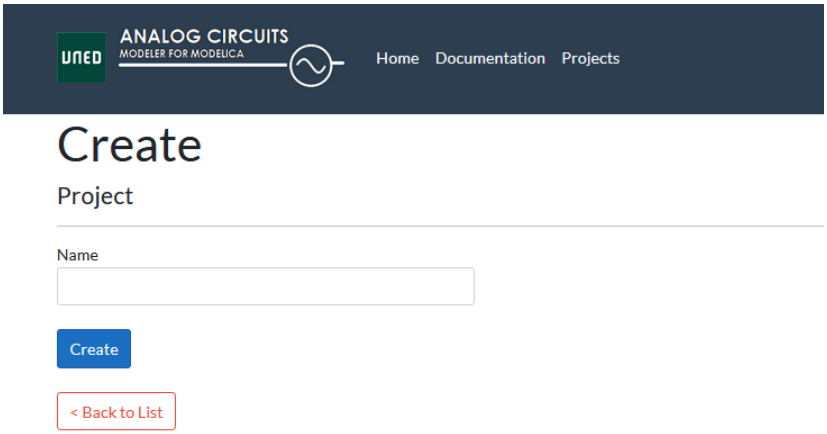


Figura A-5 Página de creación de proyectos

Dentro de cada proyecto, podremos gestionar los circuitos o modelos creados, tanto creación como eliminación o edición de los modelos. Veremos dentro de cada proyecto, los circuitos listados con las opciones “Edit” y “Delete”

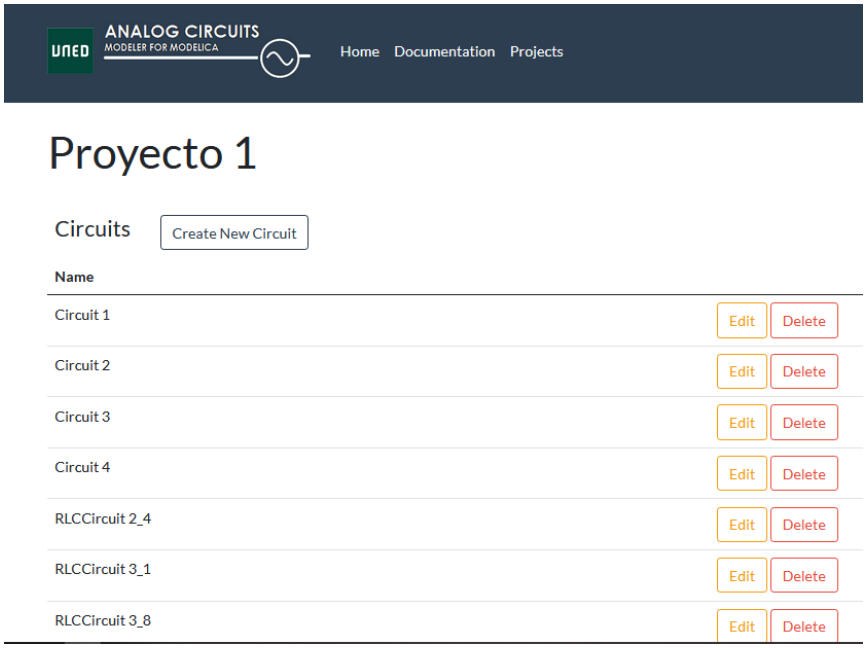


Figura A-6 Página de visualización de circuitos

A-4 Interfaz de edición de circuitos

Una vez creamos un circuito y accedemos, veremos una interfaz como la de la imagen siguiente formada por:

- Un lienzo donde veremos el modelo y podremos editar y conectar elementos
- Una barra de herramientas superior donde podremos
    - Aumentar o reducir el zoom dentro del lienzo
    - Rehacer o deshacer el historial de cambios
    - Eliminar el elemento seleccionado
    - Generar al vista XML del modelo en MxGrpah
    - Guardar el modelo XML del modelo en MxGrpah en un fichero
    - Cargar un modelo XML del modelo MxGrpah a partir de un fichero
    - Generar el código Modelica
    - Elegir si el código Modelica generado será en un formato atómico o compuesto

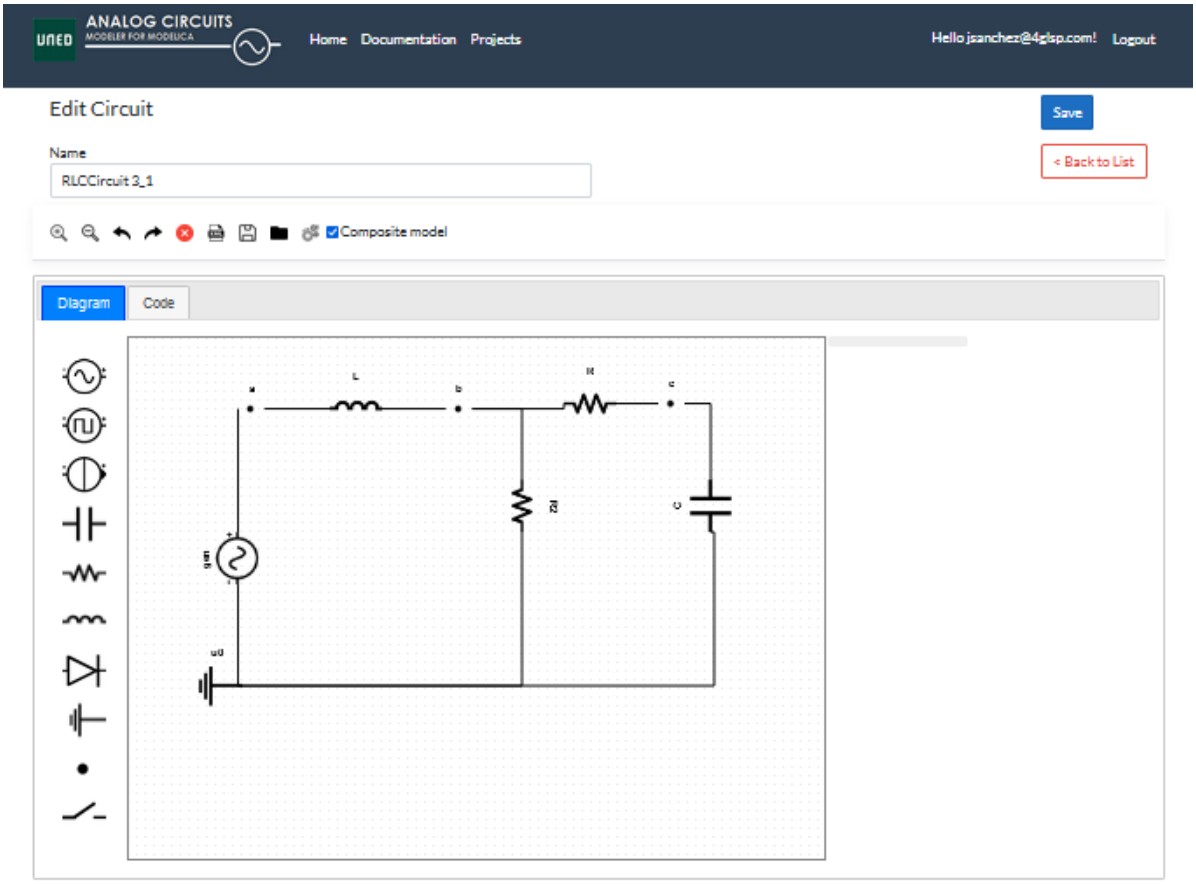


Figura A-7 Editor de circuitos

Al seleccionar cualquier elemento del modelo, podremos editar sus propiedades, estas propiedades son fijas en cada elementos, es decir, según el tipo veremos unas propiedades u otras.

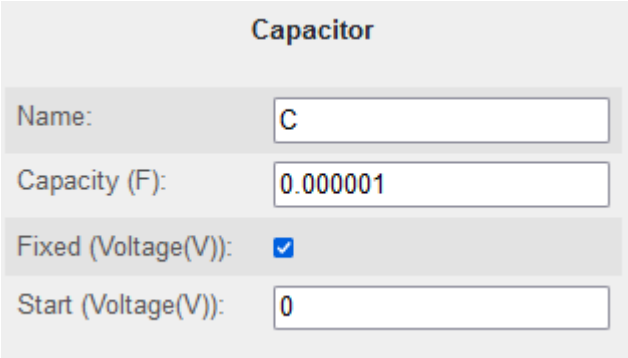


Figura A-8 Propiedades de elementos del modelo

Una vez realizado cualquier modelo, se puede guardar pinchando sobre el botón "Save". El modelo será persistido en la base de datos, guardando en formato XML el diagrama realizado.

A-5 Diagnosis de errores

Antes de realizar la generación de código se realizarán comprobaciones sobre el modelo que arrojarán mensajes en color rojo. Se podrán corregir los problemas sobre el modelo y volver a intentar la generación de código. Si todo se ha comprobado correctamente, el código se generará y se mostrará en pantalla.

Los posibles mensajes mostrados serán los siguientes:

Tabla A-1 Comprobaciones sobre el modelo

Código	Mensaje	Corrección necesaria
01	A ground connection is necessary for the model to be valid	Se debe crear un nodo tierra en el modelo
02	Only one ground node can be declared	Solo se debe crear un nodo tierra, se debe eliminar alguno de los nodos creados
03	A node between each connection of the switch is necessary for the model to be valid	Se debe crear un nodo entre cada terminal del componente interruptor para poder generar el código correctamente
04	At least one node is necessary for the model to be valid	Al menos debe existir un nodo
05	There cannot be two consecutive nodes	No pueden existir dos nodos consecutivos, se deberá eliminar uno de ellos

06	There cannot be two elements with the same name	Dos componentes del modelo tienen el mismo nombre, uno de los componentes debe renombrarse
07	All elements must be connected for the model to be valid	Existe algún componente sin conectar, es necesario conectar todos los pines del componente indicado
08	Cannot connect current sources in series	Existen dos fuentes de corriente conectadas en serie, es necesario, eliminar una de ellas o conectarlas en paralelo
09	Cannot connect voltage sources in parallel	Existen dos fuentes de tensión conectadas en paralelo, se debe eliminar una de ellas o conectar en serie.

A-6 Generación de código

Al pulsar sobra la opción de generación de código de Modelica, se comprobará el check "Composite Model", si este está activo, se generará el modelo compuesto, si no, se generará el modelo atómico. A continuación veremos el código directamente en la pestaña "Code".

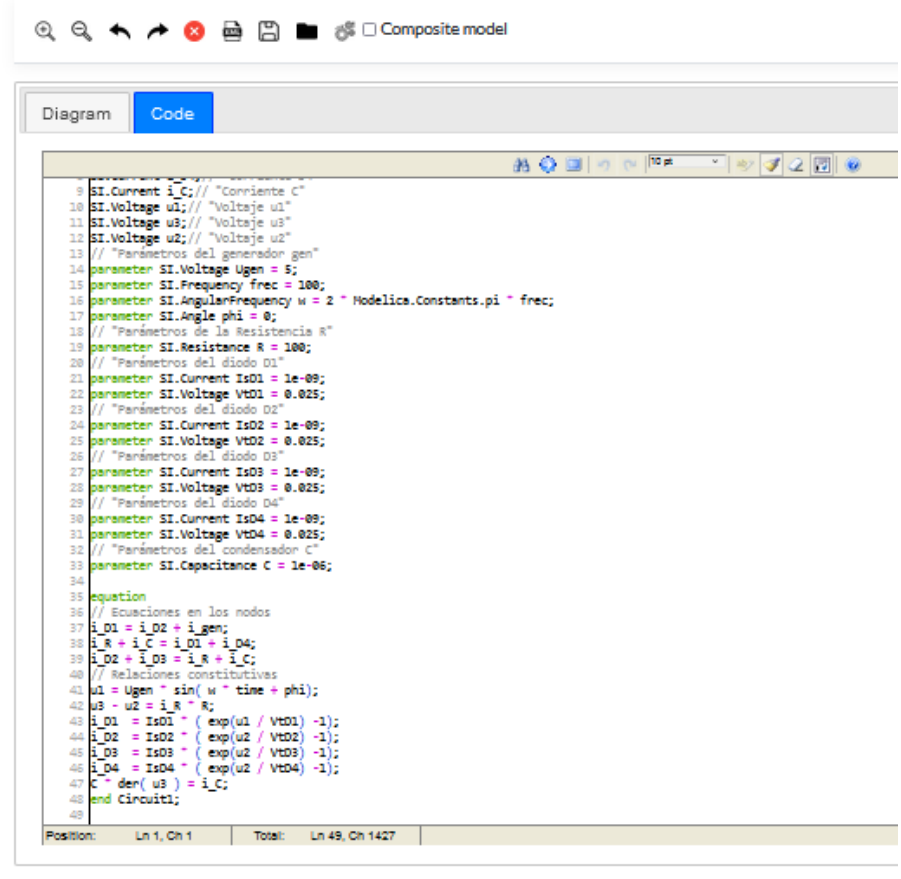


Figura A-9 Código atómico generado a partir del modelo en Modelica

El código generado para el modelo compuesto mostrará toda la definición de elementos del modelo según la Librería Estándar de Modelica. Tanto este código como el del modelo atómico podrán usarse para preparar simulaciones y trabajar sobre él en otros editores como OpenModelica o Dymola.

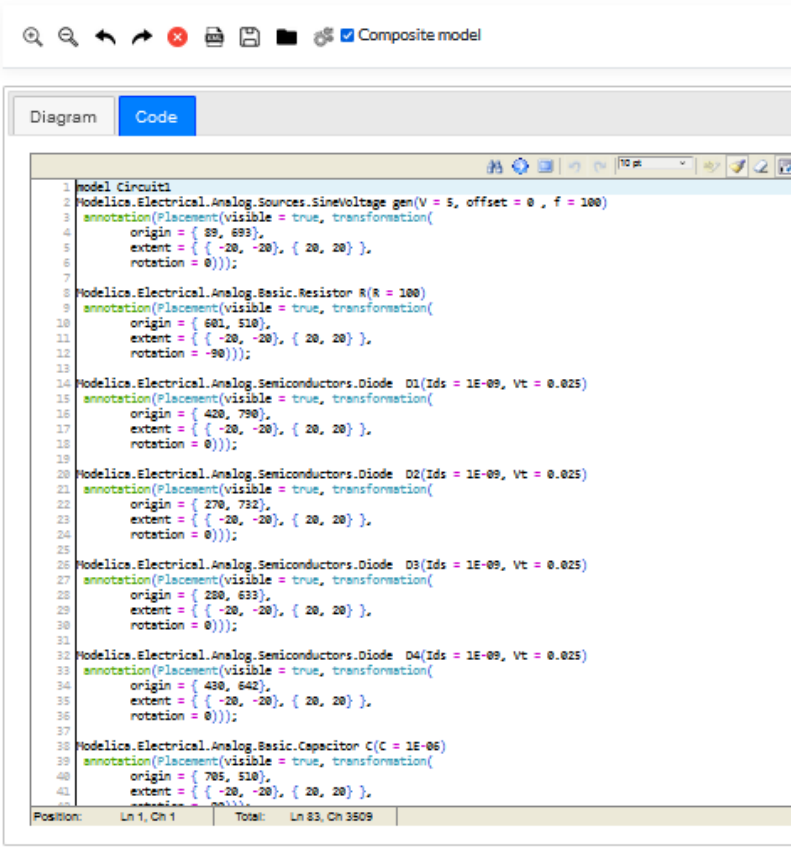


Figura A-10 Código en formato compuesto del modelo generado en Modelica

Anexo B: Código fuente

Este anexo está relacionado con el capítulo 4 “Definición y traducción de los modelos” en el cual expondremos el código fuente de los distintos ficheros que componen la capa de servicio de la aplicación. Por cuestiones de extensión, solo mostramos los ficheros importantes que componen la lógica de la aplicación.

El fichero ModelicaController recoge todo el modelo de la interfaz gráfica parseándolo y convirtiéndolo al modelo interno. Este fichero además es el encargado de realizar las comprobaciones.

B-1 ModelicaController.cs

```
using Microsoft.AspNetCore.Components.Server.Circuits;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using System.Xml;
using System.Xml.Serialization;
using WebAnalogCircuits.Helpers;
using WebAnalogCircuits.Model;
using WebAnalogCircuits.Service;
using static WebAnalogCircuits.ModeloMxGraph;
namespace WebAnalogCircuits.Controllers
{
    public class ModelicaController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }


        List<ModelItem> objects = new List<ModelItem>();
        List<Graph> graph = new List<Graph>();
        public JsonResult ProcesaEsquema(string modelo, bool composite)
        {

            try
            {
                WebAnalogCircuits.Models.Circuit circuit =
SessionHelper.GetObjectFromJson<WebAnalogCircuits.Models.Circuit>(HttpContext.Session,
"CurrentCircuit");
                string name = circuit.Name.Replace(" ", "");

                modelo = modelo.Replace("\n  ", "").Replace("\"", "'").Replace("> <",
"><").Replace(">  <", "><").Replace(">      <", "><");

                var xmlSerializer = new XmlSerializer(typeof(MxGraphModel), new
XmlRootAttribute("mxGraphModel"));
```

```
42                  MxGraphModel instance;
43
44                  using (var stringreader = new StringReader(modelo))
45                  {
46                      instance = (MxGraphModel)xmlSerializer.Deserialize(stringreader);
47                  }
48
49                  foreach (var item in instance.Root.SineVoltage)
50                  {
51                      ModelItem modelItem = new ModelItem();
52                      modelItem.Key = int.Parse(item.Id);
53                      modelItem.Name = item.Name;
54                      modelItem.Type = "SineVoltage";
55                      modelItem.Parameters = new List<ItemParameter>();
56
57                      modelItem.Parameters.Add(NewParameter("U" + item.Name, "Voltage",
58 double.Parse(item.Amplitude.Replace(".", ","))));
59                      modelItem.Parameters.Add(NewParameter("frec_" + item.Name,
60 "Frequency", double.Parse(item.AngularFrec.Replace(".", ","))));
61                      modelItem.Parameters.Add(NewParameter("phi_" + item.Name, "Angle",
62 double.Parse(item.PhaseShift.Replace(".", ","))));
63                      modelItem.Equation = "U" + item.Name + " * sin( w_" + item.Name +
64 " * time + phi_" + item.Name + ")";
65                      ProccesPosition(ref modelItem, item.MxCell);
66
67                      objects.Add(modelItem);
68                  }
69
70                  foreach (var item in instance.Root.PulseVoltage)
71                  {
72
73                      ModelItem modelItem = new ModelItem();
74                      modelItem.Key = int.Parse(item.Id);
75                      modelItem.Name = item.Name;
76                      modelItem.Type = "PulseVoltage";
77                      modelItem.Parameters = new List<ItemParameter>();
78
79                      modelItem.Parameters.Add(NewParameter("U" + item.Name, "Voltage",
80 double.Parse(item.Voltage.Replace(".", ","))));
81                      modelItem.Parameters.Add(NewParameter("T" + item.Name, "Time",
82 double.Parse(item.Period.Replace(".", ","))));
83                      modelItem.Equation = "U" + item.Name;
84                      ProccesPosition(ref modelItem, item.MxCell);
85
86                      objects.Add(modelItem);
87                  }
88
89                  foreach (var item in instance.Root.PulseCurrent)
90                  {
91                      ModelItem modelItem = new ModelItem();
92                      modelItem.Key = int.Parse(item.Id);
93                      modelItem.Name = item.Name;
94                      modelItem.Type = "PulseCurrent";
95                      modelItem.Parameters = new List<ItemParameter>();
96
97                      modelItem.Parameters.Add(NewParameter("I" + item.Name, "Current",
98 double.Parse(item.Current.Replace(".", ","))));
99                      modelItem.Parameters.Add(NewParameter("T" + item.Name, "Time",
100 double.Parse(item.Period.Replace(".", ","))));
101                     modelItem.Equation = "I" + item.Name;
102                     ProccesPosition(ref modelItem, item.MxCell);
103
104                     objects.Add(modelItem);
```

```
105                 }
106
107                 foreach (var item in instance.Root.Switch)
108                 {
109                     ModelItem modelItem = new ModelItem();
110                     modelItem.Key = int.Parse(item.Id);
111                     modelItem.Name = item.Name;
112                     modelItem.Type = "Switch";
113                     modelItem.Parameters = new List<ItemParameter>();
114                     //''TODO'
115                     modelItem.Parameters.Add(NewParameter("O" + item.Name, "On",
116 double.Parse(item.On.Replace(".", ","))));
117                     modelItem.Parameters.Add(NewParameter("T" + item.Name, "Time",
118 double.Parse(item.Period.Replace(".", ","))));
119                     modelItem.Equation = "v_" + item.Name + " = ";
120                     ProccesPosition(ref modelItem, item.MxCell);
121
122                     objects.Add(modelItem);
123                 }
124
125                 foreach (var item in instance.Root.Resistor)
126                 {
127                     ModelItem modelItem = new ModelItem();
128                     modelItem.Key = int.Parse(item.Id);
129                     modelItem.Name = item.Name;
130                     modelItem.Type = "Resistor";
131                     modelItem.Parameters = new List<ItemParameter>();
132
133                     modelItem.Parameters.Add(NewParameter(item.Name, "Resistance",
134 double.Parse(item.Resistance.Replace(".", ","))));
135                     modelItem.Equation = "i_" + modelItem.Name + " * " +
136 modelItem.Name;
137                     ProccesPosition(ref modelItem, item.MxCell);
138
139                     objects.Add(modelItem);
140                 }
141
142                 foreach (var item in instance.Root.Inductor)
143                 {
144                     ModelItem modelItem = new ModelItem();
145                     modelItem.Key = int.Parse(item.Id);
146                     modelItem.Name = item.Name;
147                     modelItem.Type = "Inductor";
148                     modelItem.Parameters = new List<ItemParameter>();
149
150                     modelItem.Parameters.Add(NewParameter(item.Name, "Inductance",
151 double.Parse(item.Induction.Replace(".", ","))));
152
153                     int fixedParam = 0;
154                     if (item.Fixed) { fixedParam = 1; }
155                     modelItem.Parameters.Add(NewParameter("fixed", "fixed",
156 fixedParam));
157
158                     double startParam = 0;
159                     if (!string.IsNullOrEmpty(item.Start)) { startParam =
160 double.Parse(item.Start.Replace(".", ",")); }
161                     modelItem.Parameters.Add(NewParameter("start", "start",
162 startParam));
163
164                     modelItem.Equation = modelItem.Name + " * der(i_" + modelItem.Name
165 + ")";
166                     ProccesPosition(ref modelItem, item.MxCell);
167
```

```
168                     objects.Add(modelItem);
169                 }
170                 foreach (var item in instance.Root.Diode)
171                 {
172                     ModelItem modelItem = new ModelItem();
173                     modelItem.Key = int.Parse(item.Id);
174                     modelItem.Name = item.Name;
175                     modelItem.Type = "Diode";
176                     modelItem.Parameters = new List<ItemParameter>();
177 
178                     modelItem.Parameters.Add(NewParameter("Is" + item.Name, "Current",
179 double.Parse(item.SaturationCurrent.Replace(".", ","))));
180                     modelItem.Parameters.Add(NewParameter("Vt" + item.Name, "Voltage",
181 double.Parse(item.ThermalVoltage.Replace(".", ","))));
182 
183                     int fixedParam = 0;
184                     if (item.Fixed) { fixedParam = 1; }
185                     modelItem.Parameters.Add(NewParameter("fixed", "fixed",
186 fixedParam));
187 
188                     double startParam = 0;
189                     if (!string.IsNullOrEmpty(item.Start)) { startParam =
190 double.Parse(item.Start.Replace(".", ",")); }
191                     modelItem.Parameters.Add(NewParameter("start", "start",
192 startParam));
193 
194                     modelItem.Equation = "Is" + item.Name + " * ( exp(u / Vt" +
195 item.Name + ") -1)";
196                     ProccesPosition(ref modelItem, item.MxCell);
197 
198                     objects.Add(modelItem);
199                 }
200 
201                 foreach (var item in instance.Root.Capacitor)
202                 {
203                     ModelItem modelItem = new ModelItem();
204                     modelItem.Key = int.Parse(item.Id);
205                     modelItem.Name = item.Name;
206                     modelItem.Type = "Capacitor";
207                     modelItem.Parameters = new List<ItemParameter>();
208 
209                     modelItem.Parameters.Add(NewParameter(item.Name, "Capacitance",
210 double.Parse(item.Capacity.Replace(".", ","))));
211 
212                     int fixedParam = 0;
213                     if (item.Fixed) { fixedParam = 1; }
214                     modelItem.Parameters.Add(NewParameter("fixed", "fixed",
215 fixedParam));
216 
217 
218                     double startParam = 0;
219                     if (!string.IsNullOrEmpty(item.Start)) { startParam =
220 double.Parse(item.Start.Replace(".", ",")); }
221                     modelItem.Parameters.Add(NewParameter("start", "start",
222 startParam));
223 
224                     modelItem.Equation = item.Name + " * der( u_" + item.Name + " )";
225                     ProccesPosition(ref modelItem, item.MxCell);
226 
227                     objects.Add(modelItem);
228                 }
229 
230                 foreach (var item in instance.Root.Ground)
```

```
231                 {
232
233                     ModelItem modelItem = new ModelItem();
234                     modelItem.Key = int.Parse(item.Id);
235                     modelItem.Name = item.Name;
236                     modelItem.Type = "Ground";
237                     modelItem.Parameters = new List<ItemParameter>();
238                     ProccesPosition(ref modelItem, item.MxCell);
239
240                     objects.Add(modelItem);
241                 }
242
243                 foreach (var item in instance.Root.Node)
244                 {
245
246                     ModelItem modelItem = new ModelItem();
247                     modelItem.Key = int.Parse(item.Id);
248                     modelItem.Name = item.Name;
249                     modelItem.Type = "Node";
250                     modelItem.Parameters = new List<ItemParameter>();
251                     ProccesPosition(ref modelItem, item.MxCell);
252
253                     objects.Add(modelItem);
254                 }
255
256                 ProcessGraph(ref instance);
257                 ProcesRoutes(ref instance);
258
259                 foreach (var con in instance.Root.MxCell)
260                 {
261                     if (con.Source != null && con.Target != null)
262                     {
263                         ModelItem s = objects.FirstOrDefault(x => x.Key ==
264 int.Parse(con.Source));
265                         ModelItem t = objects.FirstOrDefault(x => x.Key ==
266 int.Parse(con.Target));
267
268                         string[] styles = con.Style.Split(";");
269                         string origin = "1";
270                         string dest = "0";
271                         foreach (string v in styles)
272                         {
273                             if (v.Contains("exitX="))
274                             {
275                                 origin = v.Replace("exitX=", "");
276                             }
277                             if (v.Contains("entryX="))
278                             {
279                                 dest = v.Replace("entryX=", "");
280                             }
281                         }
282
283                         if ((t.Type == "Ground" && (s.Type != "SineVoltage" && s.Type
284 != "PulseVoltage" && s.Type != "PulseCurrent")) ||
285                             (s.Type == "Ground" && (t.Type != "SineVoltage" && t.Type
286 != "PulseVoltage" && t.Type != "PulseCurrent")))
287                             continue;
288
289                         ModelItem conector = objects.FirstOrDefault(x => x.Type ==
290 "Connector" && x.NextItems[0].Name == t.Name && x.PrevItems[0].Name == s.Name);
291                         if (conector == null)
292                         {
293
```

```
294                             conector = objects.FirstOrDefault(x => x.Type ==
295 "Connector" && x.NextItems[0].Name == s.Name && x.PrevItems[0].Name == t.Name);
296                         }
297                     }
298                 }
299 
300                 string message = string.Empty;
301                 string modelicaCode = string.Empty;
302 
303                 foreach (ModelItem item in objects)
304                 {
305                     if (item.Type != "Node" && item.Type != "Ground" && item.Type !=
306 "Connector")
307                     {
308                         if (((item.NextItems != null  &&
309 item.NextItems.FirstOrDefault(x => x.Type != "Ground" && x.Type != "Node") != null))
310 ||
311                             (item.PrevItems != null &&item.PrevItems.FirstOrDefault(x
312 => x.Type != "Ground" && x.Type != "Node") != null))
313                         {
314                             message = "Check rule: A component can only be connected
315 to one node or to ground: " + item.Name;
316                             return Json(message);
317                         }
318                     }
319                 }
320 
321                 if (objects.Count(x => x.Type == "Ground") != 0)
322                 {
323                     ModelItem g = objects.FirstOrDefault(x => x.Type == "Ground");
324                     if ((g.PrevItems  != null && g.PrevItems.FirstOrDefault(x =>
325 x.Type == "Node") != null) || (g.NextItems != null && g.NextItems.FirstOrDefault(x =>
326 x.Type == "Node") != null))
327                     {
328                         message = "Check rule: A node cannot be connected to the
329 ground node";
330                         return Json(message);
331                     }
332                 }
333                 new Normalization().Normalize(ref objects, instance, ref graph);
334                 new Normalization().Normalize(ref objects, instance, ref graph);
335 
336                 bool valid = new ValidationsChecks().Validations(objects, ref message,
337 ref graph);
338                 if (!valid)
339                 {
340                     return Json(message);
341                 }
342 
343                 if (composite)
344                 {
345                     CompositeService service = new CompositeService();
346                     modelicaCode = service.TranslateModel(objects, name);
347                 }
348                 else
349                 {
350                     FlatService service = new FlatService();
351                     modelicaCode = service.TranslateModel(objects, name);
352                 }
353                 return Json(modelicaCode);
354             }
355             catch (Exception ex)
356             {
```

```
357                 int i = 0;
358             }
359             return Json(null);
360         }
361 
362         public void ProccesPosition(ref ModelItem modelItem, MxCell item)
363         {
364             if (item.Style.Split(";").Count(x => x.Contains("rotation")) > 0)
365             {
366                 string rot = item.Style.Split(";").First(x => x.Contains("rotation"));
367                 modelItem.Rotation = (decimal)double.Parse(rot.Replace("rotation=",
368 ""));
369             }
370             if (modelItem.Type == "Node")
371             {
372                 modelItem.PositionX =
373 Math.Round((decimal)double.Parse(item.MxGeometry.X.Replace(".", ",")), 0) - 15;
374                 modelItem.PositionY =
375 Math.Round((decimal)double.Parse(item.MxGeometry.Y.Replace(".", ",")), 0) - 15;
376             }
377             else
378             {
379                 modelItem.PositionX =
380 Math.Round((decimal)double.Parse(item.MxGeometry.X.Replace(".", ",")), 0);
381                 modelItem.PositionY =
382 Math.Round((decimal)double.Parse(item.MxGeometry.Y.Replace(".", ",")), 0);
383             }
384 
385             modelItem.PositionX = (modelItem.PositionX);
386             modelItem.PositionY = 800 - (modelItem.PositionY);
387         }
388 
389         public ItemParameter NewParameter(string name, string magnitude, double value)
390         {
391             ItemParameter parameter = new ItemParameter();
392             parameter.Name = name;
393             parameter.Magnitude = magnitude;
394             parameter.Value = value;
395             return parameter;
396         }
397 
398         private void ProcessRouteRecursive(Graph g, ref MxGraphModel instance)
399         {
400             if (!g.Processed)
401             {
402                 if (g.Item.Type == "Ground")
403                 {
404                     if (g.NegativePin != null && g.NegativePin.Count > 0)
405                     {
406                         foreach (Graph p in g.NegativePin)
407                         {
408                             if (p.Item.Type == "SineVoltage" || p.Item.Type ==
409 "PulseVoltage" || p.Item.Type == "PulseCurrent")
410                             {
411                                 if (g.Item.NextItems == null) g.Item.NextItems = new
412 List<ModelItem>();
413                                 if (!g.Item.NextItems.Contains(p.Item))
414                                 {
415                                     g.Item.NextItems.Add(p.Item);
416                                     g.Item.nextItemPin = "-";
417                                     CreateConnections(g.Item, p.Item, instance);
418                                     if (!p.Processed)
419                                         ProcessRouteRecursive(p, ref instance);
```

```
420                                 }
421                             }
422                         }
423                     }
424                 }
425                 else if (g.Item.Type == "SineVoltage" || g.Item.Type == "PulseVoltage"
426 || g.Item.Type == "PulseCurrent")
427                 {
428                     if (g.PositivePin != null && g.PositivePin.Count > 0)
429                     {
430                         foreach (Graph p in g.PositivePin)
431                         {
432                             if (g.Item.NextItems == null) g.Item.NextItems = new
433 List<ModelItem>();
434                             if (!g.Item.NextItems.Contains(p.Item))
435                             {
436                                 g.Item.NextItems.Add(p.Item);
437                                 g.Item.nextItemPin = "+";
438                                 CreateConnections(g.Item, p.Item, instance);
439                                 if (!p.Processed)
440                                     ProcessRouteRecursive(p, ref instance);
441                             }
442                         }
443                     }
444                     if (g.NegativePin != null && g.NegativePin.Count > 0)
445                     {
446                         foreach (Graph p in g.NegativePin)
447                         {
448                             if (g.Item.PrevItems == null) g.Item.PrevItems = new
449 List<ModelItem>();
450                             if (!g.Item.PrevItems.Contains(p.Item))
451                             {
452                                 g.Item.PrevItems.Add(p.Item);
453                                 g.Item.prevItemPin = "-";
454                                 CreateConnections(p.Item, g.Item, instance);
455                                 if (!p.Processed)
456                                     ProcessRouteRecursive(p, ref instance);
457                             }
458                         }
459                     }
460                 }
461                 else
462                 {
463                     if (g.NegativePin != null && g.NegativePin.Count > 0)
464                     {
465                         if (g.Item.NextItems == null) g.Item.NextItems = new
466 List<ModelItem>();
467                         if (g.Item.PrevItems == null) g.Item.PrevItems = new
468 List<ModelItem>();
469                         foreach (Graph p in g.NegativePin)
470                         {
471                             string nPin = g.NegativePinConnection.FirstOrDefault(x =>
472 x.Name == p.Name).Pin;
473                             if (p.Item.NextItems == null) p.Item.NextItems = new
474 List<ModelItem>();
475                             if (p.Item.Type == "Ground" &&
476 !p.Item.NextItems.Contains(g.Item))
477                             {
478                                 if (!g.Item.NextItems.Contains(p.Item))
479                                 {
480                                     g.Item.NextItems.Add(p.Item);
481                                     g.Item.nextItemPin = "-";
482                                     CreateConnections(g.Item, p.Item, instance);
```

```
                                    if (!p.Processed)
                                        ProcessRouteRecursive(p, ref instance);
                                }
                            }
                            else if ((p.Item.NextItems.Contains(g.Item) && p.Item.Type
!= "Ground") && g.Item.prevItemPin == "-" || (p.Item.NextItems.Count < 2 &&
p.Item.NextItems.Contains(g.Item) && p.Item.Type == "Ground" && g.Item.prevItemPin ==
"-"))

                            //else if ((p.Item.NextItems.Contains(g.Item) &&
p.Item.Type != "Ground")  || (p.Item.NextItems.Count < 2 &&
p.Item.NextItems.Contains(g.Item) && p.Item.Type == "Ground" && g.Item.prevItemPin ==
"-"))
                            {
                                if (!g.Item.PrevItems.Contains(p.Item))  //if
(!g.Item.PrevItems.Contains(p.Item) && !g.Item.NextItems.Contains(p.Item))
                                {
                                    g.Item.PrevItems.Add(p.Item);
                                    if (g.Item.prevItemPin == null) g.Item.prevItemPin
= "-";
                                    if (g.Item.nextItemPin == null) g.Item.nextItemPin
= "+";
                                    CreateConnections(g.Item, p.Item, instance);
                                }
                            }
                            else
                            {
                                if (!g.Item.NextItems.Contains(p.Item))
                                {
                                    g.Item.NextItems.Add(p.Item);
                                    if (g.Item.nextItemPin == null) g.Item.nextItemPin
= "-";
                                    if (g.Item.prevItemPin == null) g.Item.prevItemPin
= "+";
                                    CreateConnections(g.Item, p.Item, instance);
                                    if (!p.Processed)
                                        ProcessRouteRecursive(p, ref instance);
                                }
                            }
                        }
                    }
                    if (g.PositivePin != null && g.PositivePin.Count > 0)
                    {
                        foreach (Graph p in g.PositivePin)
                        {
                            if (p.Item.Type == "Ground" &&
!p.Item.NextItems.Contains(g.Item))
                            {
                                if (!g.Item.NextItems.Contains(p.Item))
                                {
                                    g.Item.NextItems.Add(p.Item);
                                    CreateConnections(g.Item, p.Item, instance);
                                    if (!p.Processed)
                                        ProcessRouteRecursive(p, ref instance);
                                }
                            }
                            else if (p.Item.Type == "Ground" &&
(g.Item.NextItems.Contains(p.Item) || g.Item.PrevItems.Contains(p.Item)))
                            {
                                continue;
                            }
                            else if (g.Item.nextItemPin == "+")
                            {
```

```
546                                 if (!g.Item.NextItems.Contains(p.Item))
547                                 {
548                                     if (g.Item.NextItems == null) g.Item.NextItems =
549 new List<ModelItem>();
550                                     g.Item.NextItems.Add(p.Item);
551                                     g.Item.nextItemPin = "+";
552                                     CreateConnections(p.Item, g.Item, instance);
553                                     if (!p.Processed)
554                                         ProcessRouteRecursive(p, ref instance);
555                                 }
556 
557                             }
558                             else if (!g.Item.PrevItems.Contains(p.Item))
559                             {
560                                 if (g.Item.PrevItems == null) g.Item.PrevItems = new
561 List<ModelItem>();
562                                 g.Item.PrevItems.Add(p.Item);
563                                 if (g.Item.prevItemPin == null) g.Item.prevItemPin =
564 "+";
565                                 CreateConnections(p.Item, g.Item, instance);
566                                 if (!p.Processed)
567                                     ProcessRouteRecursive(p, ref instance);
568                             }
569                         }
570                     }
571                 }
572                 if (g.Item.Type == "Ground")
573                 {
574                     if (g.NegativePin != null && g.NegativePin.Count > 0)
575                     {
576                         foreach (Graph p in g.NegativePin)
577                         {
578                             if (p.Item.Type != "SineVoltage" && p.Item.Type !=
579 "PulseVoltage" && p.Item.Type != "PulseCurrent")
580                             {
581                                 if (g.Item.NextItems == null) g.Item.NextItems = new
582 List<ModelItem>();
583                                 if (!g.Item.NextItems.Contains(p.Item))
584                                 {
585                                     g.Item.NextItems.Add(p.Item);
586                                     g.Item.nextItemPin = "-";
587                                     CreateConnections(p.Item, g.Item, instance);
588                                     if (!p.Processed)
589                                         ProcessRouteRecursive(p, ref instance);
590                                 }
591                             }
592                         }
593                     }
594                 }
595                 g.Processed = true;
596             }
597         }
598 
599         private void CreateConnections(ModelItem next, ModelItem prev, MxGraphModel
600 instance)
601         {
602             foreach (var item in instance.Root.MxCell)
603             {
604                 if (objects.FirstOrDefault(x => x.Key == int.Parse(item.Id)) == null)
605                 {
606                     if (item.Source != null && item.Target != null)
607                     {
608 
```

```
609                         ModelItem s = objects.FirstOrDefault(x => x.Key ==
610 int.Parse(item.Source));
611                         ModelItem t = objects.FirstOrDefault(x => x.Key ==
612 int.Parse(item.Target));
613                         if ((s.Key == next.Key && t.Key == prev.Key) || (t.Key ==
614 next.Key && s.Key == prev.Key))
615                         {
616                             ModelItem modelItem = new ModelItem();
617                             modelItem.Key = int.Parse(item.Id);
618                             modelItem.Name = item.Id.ToString();
619                             modelItem.Type = "Connector";
620                             modelItem.Parameters = new List<ItemParameter>();
621                             modelItem.Points = new List<ItemPoint>();
622                             modelItem.LastConnection = false;
623                             if (item.MxGeometry.Array != null &&
624 item.MxGeometry.Array.MxPoint != null)
625                             {
626                                 foreach (MxPoint point in
627 item.MxGeometry.Array.MxPoint)
628                                 {
629                                     ItemPoint itemPoint = new ItemPoint();
630                                     itemPoint.PositionX = decimal.Parse(point.X);
631                                     itemPoint.PositionY = 800 -
632 decimal.Parse(point.Y);
633                                     modelItem.Points.Add(itemPoint);
634                                 }
635                             }
636
637                             modelItem.NextItems = new List<ModelItem>();
638                             modelItem.NextItems.Add(prev);
639
640                             modelItem.PrevItems = new List<ModelItem>();
641                             modelItem.PrevItems.Add(next);
642
643                             objects.Add(modelItem);
644                         }
645                     }
646                 }
647             }
648         }
649
650         private void ProcesRoutes(ref MxGraphModel instance)
651         {
652             foreach (Graph g in graph)
653             {
654                 if (g.Item.Type == "Ground")
655                 {
656                     ProcessRouteRecursive(g, ref instance);
657                 }
658             }
659         }
660
661         private void ProcessGraph(ref MxGraphModel instance)
662         {
663             foreach (var item in instance.Root.MxCell)
664             {
665                 if (item.Source != null && item.Target != null)
666                 {
667                     ModelItem modelItem = new ModelItem();
668                     modelItem.Key = int.Parse(item.Id);
669                     modelItem.Name = item.Id.ToString();
670                     modelItem.Type = "Connector";
671                     modelItem.Parameters = new List<ItemParameter>();
```

```
672                     modelItem.Points = new List<ItemPoint>();
673                     modelItem.LastConnection = false;
674                     if (item.MxGeometry.Array != null && item.MxGeometry.Array.MxPoint
675 != null)
676                     {
677                         foreach (MxPoint point in item.MxGeometry.Array.MxPoint)
678                         {
679                             ItemPoint itemPoint = new ItemPoint();
680                             itemPoint.PositionX = decimal.Parse(point.X);
681                             itemPoint.PositionY = 800 - decimal.Parse(point.Y);
682                             modelItem.Points.Add(itemPoint);
683                         }
684                     }
685 
686                     ModelItem s = objects.FirstOrDefault(x => x.Key ==
687 int.Parse(item.Source));
688                     ModelItem t = objects.FirstOrDefault(x => x.Key ==
689 int.Parse(item.Target));
690 
691 
692                     string[] styles = item.Style.Split(";");
693                     string origin = "1";
694                     string dest = "0";
695                     foreach (string v in styles)
696                     {
697                         if (v.Contains("exitX="))
698                         {
699                             origin = v.Replace("exitX=", "");
700                         }
701                         if (v.Contains("entryX="))
702                         {
703                             dest = v.Replace("entryX=", "");
704                         }
705                     }
706 
707                     Graph ta = new Graph();
708                     if (graph.FirstOrDefault(x => x.Key == t.Key) == null)
709                     {
710                         ta.Key = t.Key;
711                         ta.Name = t.Name;
712                         ta.Item = t;
713                         ta.Processed = false;
714                         ta.PositivePin = new List<Graph>();
715                         ta.NegativePin = new List<Graph>();
716                         ta.PositivePinConnection = new List<ConnectionPin>();
717                         ta.NegativePinConnection = new List<ConnectionPin>();
718                         graph.Add(ta);
719                     }
720                     else
721                     {
722                         ta = graph.FirstOrDefault(x => x.Key == t.Key);
723                     }
724 
725                     Graph so = new Graph();
726                     if (graph.FirstOrDefault(x => x.Key == s.Key) == null)
727                     {
728                         so.Key = s.Key;
729                         so.Name = s.Name;
730                         so.Item = s;
731                         so.Processed = false;
732                         so.PositivePin = new List<Graph>();
733                         so.NegativePin = new List<Graph>();
734                         so.PositivePinConnection = new List<ConnectionPin>();
```

```
735                         so.NegativePinConnection = new List<ConnectionPin>();
736                         graph.Add(so);
737                     }
738                     else
739                     {
740                         so = graph.FirstOrDefault(x => x.Key == s.Key);
741                     }
742 
743                     ConnectionPin o = new ConnectionPin();
744                     o.Item = so.Item;
745                     o.Key = so.Key;
746                     o.Name = so.Name;
747                     if (origin == "1" && (s.Type != "SineVoltage" && s.Type !=
748 "PulseVoltage" && s.Type != "PulseCurrent"))
749                     {
750                         o.Pin = "-";
751                     }
752                     else if (origin == "1" && (s.Type == "SineVoltage" || s.Type ==
753 "PulseVoltage" || s.Type == "PulseCurrent"))
754                     {
755                         o.Pin = "+";
756                     }
757                     if (origin == "0" && (s.Type != "SineVoltage" && s.Type !=
758 "PulseVoltage" && s.Type != "PulseCurrent"))
759                     {
760                         o.Pin = "+";
761                     }
762                     else if (origin == "0" && (s.Type == "SineVoltage" || s.Type ==
763 "PulseVoltage" || s.Type == "PulseCurrent"))
764                     {
765                         o.Pin = "-";
766                     }
767                     if (dest == "0")
768                     {
769 
770 
771                         if (t.Type != "SineVoltage" && t.Type != "PulseVoltage" &&
772 t.Type != "PulseCurrent")
773                         {
774                             ta.PositivePin.Add(so);
775                             ta.PositivePinConnection.Add(o);
776                         }
777                         else
778                         {
779                             ta.NegativePin.Add(so);
780                             ta.NegativePinConnection.Add(o);
781                         }
782                     }
783                     else
784                     {
785                         if (t.Type != "SineVoltage" && t.Type != "PulseVoltage" &&
786 t.Type != "PulseCurrent")
787                         {
788                             ta.NegativePin.Add(so);
789                             ta.NegativePinConnection.Add(o);
790                         }
791                         else
792                         {
793                             ta.PositivePin.Add(so);
794                             ta.PositivePinConnection.Add(o);
795                         }
796                     }
797 
```

```
798
799                     ConnectionPin d = new ConnectionPin();
800                     d.Item = ta.Item;
801                     d.Key = ta.Key;
802                     d.Name = ta.Name;
803                     if (dest == "1" && (t.Type != "SineVoltage" && t.Type !=
804 "PulseVoltage" && t.Type != "PulseCurrent"))
805                     {
806                         d.Pin = "-";
807                     }
808                     else if (dest == "1" && (t.Type == "SineVoltage" || t.Type ==
808 "PulseVoltage" || t.Type == "PulseCurrent"))
809                     {
810                         d.Pin = "+";
811                     }
812                     if (dest == "0" && (t.Type != "SineVoltage" && t.Type !=
813 "PulseVoltage" && t.Type != "PulseCurrent"))
814                     {
815                         d.Pin = "+";
816                     }
817                     else if (dest == "0" && (t.Type == "SineVoltage" || t.Type ==
818 "PulseVoltage" || t.Type == "PulseCurrent"))
819                     {
820                         d.Pin = "-";
821                     }
822
823                     if (origin == "0")
824                     {
825                         if (s.Type != "SineVoltage" && s.Type != "PulseVoltage" &&
826 s.Type != "PulseCurrent")
827                         {
828                             so.PositivePin.Add(ta);
829                             so.PositivePinConnection.Add(d);
830                         }
831                         else
832                         {
833                             so.NegativePin.Add(ta);
834                             so.NegativePinConnection.Add(d);
835                         }
836                     }
837                     else
838                     {
839
840                         if (s.Type != "SineVoltage" && s.Type != "PulseVoltage" &&
841 s.Type != "PulseCurrent")
842                         {
843                             so.NegativePin.Add(ta);
844                             so.NegativePinConnection.Add(d);
845                         }
846                         else
847                         {
848                             so.PositivePin.Add(ta);
849                             so.PositivePinConnection.Add(d);
850                         }
851                     }
852
853                 }
854
855             }
856
857         }
858
859     }
```

```
860 }
861
862
863
864
865
```

B-2 FlatService.cs

El siguiente código es el usado para realizar la generación de código al modelo atómico.

```
 1 using System;
 2 using System.Collections.Generic;
 3 using System.Linq;
 4 using System.Threading.Tasks;
 5 using WebAnalogCircuits.Model;
 6
 7 namespace WebAnalogCircuits.Service
 8 {
 9     public class FlatService
10     {
11         public string TranslateModel(List<ModelItem> objects, string name)
12         {
13             try
14             {
15                 bool SpanishComments = true;
16                 string result = string.Empty;
17                 result += "model " + name + "Flat\n";
18                 result += "import SI = Modelica.Units.SI;\n";
19
20                 result = DeclareVoltage(objects, SpanishComments, result);
21                 result = DeclareParameters(objects, SpanishComments, result);
22
23                 result += "\nequation\n";
24                 result = NodeEquations(objects, SpanishComments, result);
25                 result = ConstitutiveRelations(objects, SpanishComments, result);
26
27                 result += "end " + name + "Flat;\n";
28
29                 return result;
30             }
31             catch (Exception ex)
32             {
33                 return null;
34             }
35         }
36
37         //Declaro las variables necesarias para inicializar los voltajes y las tensiones
38         public string DeclareVoltage(List<ModelItem> objects, bool SpanishComments, string
39 result)
40         {
41
42             foreach (ModelItem item in objects)
43             {
44                 if (item.Type == "Ground")
45                 {
46                     continue;
```

```
47                 }
48                 // Por cada nodo declaramos un voltaje
49                 if (item.Type == "Node")
50                 {
51                     result += "  SI.Voltage " + item.Name + ";";
52 
53                     if (SpanishComments) { result += "// \"Voltaje " + item.Name + "\""; }
54                     result += "\n";
55                 }
56                 // Por cada switch declaremos un voltaje, tensión y booleano de estado Off
57                 else if (item.Type == "Switch")
58                 {
59                     result += "  SI.Current i_" + item.Name + ";";
60                     if (SpanishComments) { result += "// \"Corriente " + item.Name + "\"";
61 }
62                     result += "\n";
63 
64                     result += "  SI.Voltage v_" + item.Name + ";";
65                     if (SpanishComments) { result += "// \"Voltaje " + item.Name + "\""; }
66                     result += "\n";
67                     foreach (ItemParameter par in item.Parameters)
68                     {
69                         if (par.Magnitude == "On")
70                         {
71                             string val = "true";
72                             if (par.Value == 0) { val = "false"; }
73                             result += "  Boolean off_" + item.Name + "(start=" + val + ",
74 fixed=true);";
75                             result += "\n";
76                         }
77                     }
78                 }
79                 // Para las fuenets de onda cuadrada se declara un voltaje, una tensión y
80 un booleano Up
81                 else if (item.Type == "PulseVoltage")
82                 {
83 
84                     result += "  SI.Current i_" + item.Name + ";";
85                     result += "\n";
86 
87                     foreach (ItemParameter par in item.Parameters)
88                     {
89                         if (par.Magnitude == "Voltage")
90                         {
91                             string val = par.Value.ToString();
92                             result += "  SI.Voltage u_" + item.Name + ";";
93                             result += "\n";
94                             result += "  Boolean up_" + item.Name + ";";
95                             result += "\n";
96                         }
97                     }
98                 }
99                 // Para las fuenets de onda cuadrada se declara un voltaje, una tensión y
100 un booleano Up
101                 else if (item.Type == "PulseCurrent")
102                 {
103 
104                     result += "  SI.Current i_" + item.Name + ";";
105                     result += "\n";
106                     result += "  SI.Voltage u_" + item.Name + ";";
107                     result += "\n";
108                     result += "  Boolean up_" + item.Name + ";";
109                     result += "\n";
```

```
110
111                 }
112                 else if (item.Type != "Connector")
113                 {
114                     // Para el resto de elementos tenemos una tensión
115                     if (item.Type != "Capacitor")
116                     {
117                         string fixedParam = ReadFixedParam(item);
118                         result += "  SI.Current i_" + item.Name + fixedParam + ";";
119                         if (SpanishComments) { result += "// \"Corriente " + item.Name +
120 "\""; }
121                     }
122                     // Los condensadores declaran voltaje y tensión
123                     else
124                     {
125                         string fixedParam = ReadFixedParam(item);
126                         result += "  SI.Current i_" + item.Name + ";";
127                         if (SpanishComments) { result += "// \"Corriente " + item.Name +
128 "\"\n"; }
129                         result += "  SI.Voltage u_" + item.Name + fixedParam + ";";
130                         if (SpanishComments) { result += "// \"Voltaje " + item.Name +
131 "\""; }
132                     }
133                     result += "\n";
134                 }
135             }
136             return result;
137         }
138
139         //Declaramos todos los parámetros configurados para cada elemento
140         public string DeclareParameters(List<ModelItem> objects, bool SpanishComments,
141 string result)
142         {
143             bool declareFirstSwitch = false;
144
145             foreach (ModelItem item in objects)
146             {
147                 if (item.Type == "SineVoltage")
148                 {
149                     if (SpanishComments) { result += "// \"Parámetros del generador " +
150 item.Name + "\"\n"; }
151                     foreach (ItemParameter par in item.Parameters)
152                     {
153                         if (par.Name.Contains("frec"))
154                         {
155                             result += "  " + "parameter SI." + par.Magnitude + " " +
156 par.Name + " = " + par.Value.ToString().Replace(",", ".").Replace("E", "e") + ";\n";
157                             result += "  " + "parameter SI.AngularFrequency w_" + item.Name
158 + " = 2 * Modelica.Constants.pi * " + par.Name + ";\n";
159                         }
160                         else
161                         {
162                             result += "  " + "parameter SI." + par.Magnitude + " " +
163 par.Name + " = " + par.Value.ToString().Replace(",", ".").Replace("E", "e") + ";\n";
164                         }
165                     }
166                 }
167
168                 if (item.Type == "PulseVoltage")
169                 {
170                     if (SpanishComments) { result += "// \"Parámetros del generador " +
171 item.Name + "\"\n"; }
172                     foreach (ItemParameter par in item.Parameters)
```

```
173                     {
174                         if (par.Magnitude == "Time") { par.Value = par.Value / 2; }
175                         result += "  " + "parameter SI." + par.Magnitude + " " + par.Name +
176 " = " + par.Value.ToString().Replace(",", ".").Replace("E", "e") + ";\n";
177                     }
178                 }
179 
180                 if (item.Type == "PulseCurrent")
181                 {
182                     if (SpanishComments) { result += "// \"Parámetros del generador " +
183 item.Name + "\"\n"; }
184                     foreach (ItemParameter par in item.Parameters)
185                     {
186                         if (par.Magnitude == "Time") { par.Value = par.Value / 2; }
187                         result += "  " + "parameter SI." + par.Magnitude + " " + par.Name +
188 " = " + par.Value.ToString().Replace(",", ".").Replace("E", "e") + ";\n";
189                     }
190                 }
191 
192                 if (item.Type == "Resistor")
193                 {
194                     if (SpanishComments) { result += "// \"Parámetros de la Resistencia " +
195 item.Name + "\"\n"; }
196                     foreach (ItemParameter par in item.Parameters)
197                     {
198                         result += "  " + "parameter SI." + par.Magnitude + " " + par.Name +
199 " = " + par.Value.ToString().Replace(",", ".").Replace("E", "e") + ";\n";
200                     }
201                 }
202 
203                 if (item.Type == "Inductor")
204                 {
205                     if (SpanishComments) { result += "// \"Parámetros del inductor " +
206 item.Name + "\"\n"; }
207                     foreach (ItemParameter par in item.Parameters)
208                     {
209                         if (par.Name != "fixed" && par.Name != "start")
210                             result += "  " + "parameter SI." + par.Magnitude + " " +
211 par.Name + " = " + par.Value.ToString().Replace(",", ".").Replace("E", "e") + ";\n";
212                     }
213                 }
214 
215                 if (item.Type == "Capacitor")
216                 {
217                     if (SpanishComments) { result += "// \"Parámetros del condensador " +
218 item.Name + "\"\n"; }
219 
220                     foreach (ItemParameter par in item.Parameters)
221                     {
222                         if (par.Name != "fixed" && par.Name != "start")
223                             result += "  " + "parameter SI." + par.Magnitude + " " +
224 par.Name + " = " + par.Value.ToString().Replace(",", ".").Replace("E", "e") + ";\n";
225                     }
226                 }
227 
228                 if (item.Type == "Diode")
229                 {
230                     if (SpanishComments) { result += "// \"Parámetros del diodo " +
231 item.Name + "\"\n"; }
232                     foreach (ItemParameter par in item.Parameters)
233                     {
234                         if (par.Name != "fixed" && par.Name != "start")
235 
```

```
236                             result += "  " + "parameter SI." + par.Magnitude + " " +
237 par.Name + " = " + par.Value.ToString().Replace(",", ".").Replace("E", "e") + ";\n";
238                     }
239                 }
240 
241                 if (item.Type == "Switch")
242                 {
243                     if (SpanishComments) { result += "// \"Parámetros del interruptor " +
244 item.Name + "\"\n"; }
245                     foreach (ItemParameter par in item.Parameters)
246                     {
247                         if (par.Magnitude == "Time")
248                         {
249                             par.Value = par.Value / 2;
250                             result += "  " + "parameter SI." + par.Magnitude + " " +
251 par.Name + " = " + par.Value.ToString().Replace(",", ".").Replace("E", "e") + ";\n";
252                         }
253                     }
254                     if (SpanishComments) { result += "// \"Closed switch resistancer " +
255 item.Name + "\"\n"; }
256                     if (!declareFirstSwitch)
257                     {
258                         result += "  " + "parameter SI.Resistance Ron(final min=0) = 1e-
259 5;\n";
260                         result += "  " + "parameter SI.Conductance Goff(final min=0) = 1e-
261 5;\n";
262                         result += "  " + "Real s(final unit=\"1\") \"Auxiliary
263 variable\";\n";
264                         result += "  " + "constant SI.Current unitCurrent=1 annotation
265 (HideResult=true); \n";
266                         result += "  " + "constant SI.Voltage unitVoltage=1 annotation
267 (HideResult=true); \n";
268                     }
269                     declareFirstSwitch = true;
270                 }
271             }
272             return result;
273         }
274 
275         public string ReadFixedParam(ModelItem item)
276         {
277             string fixedParam = "";
278             var fixedPa = item.Parameters.FirstOrDefault(x => x.Name == "fixed");
279             var startPa = item.Parameters.FirstOrDefault(x => x.Name == "start");
280             if (fixedPa != null)
281             {
282                 if (fixedPa.Value == 1) { fixedParam = "(start=" + startPa.Value + ",
283 fixed=true)"; }
284                 else if (fixedPa.Value == 0) { fixedParam = "(start=" + startPa.Value + ",
285 fixed=false)"; }
286             }
287             return fixedParam;
288         }
289 
290         // Ecuaciones en los nodos
291         public string NodeEquations(List<ModelItem> objects, bool SpanishComments, string
292 result)
293         {
294             result += "// Ecuaciones en los nodos\n";
295             List<string> nodeEquations = new List<string>();
296             foreach (ModelItem item in objects)
297             {
298                 if (item.Type == "Node")
```

```
299                 {
300                     List<string> positives = new List<string>();
301                     List<string> negatives = new List<string>();
302 
303                     // si un nodo está conectado con tierra su voltaje es 0
304                     if (item.NextItems.Count == 1 && item.NextItems[0].Type == "Ground")
305                     {
306                         nodeEquations.Add("  " + item.Name + " = 0");
307                         continue;
308                     }
309                     // los componentes conectados en serie con un nodo suman sus
310 intensidades y las de un extremo equivalen al otro
311                     string eq = string.Empty;
312                     foreach (ModelItem p in item.NextItems)
313                     {
314                         if (p.NextItems.FirstOrDefault(x => x.Name == item.Name) != null)
315                         {
316                             if (p.Type == "SineVoltage" || p.Type == "PulseVoltage" ||
317 p.Type == "PulseCurrent")
318                             {
319                                 if (p.nextItemPin == "+")
320                                     negatives.Add("i_" + p.Name);
321                                 else
322                                     positives.Add("i_" + p.Name);
323                             }
324                             else
325                             {
326                                 if (p.nextItemPin == "+")
327                                     positives.Add("i_" + p.Name);
328                                 else
329                                     negatives.Add("i_" + p.Name);
330                             }
331                         }
332                         else
333                         {
334                             if (p.Type == "SineVoltage" || p.Type == "PulseVoltage" ||
335 p.Type == "PulseCurrent")
336                             {
337                                 if (p.prevItemPin == "+")
338                                     negatives.Add("i_" + p.Name);
339                                 else
340                                     positives.Add("i_" + p.Name);
341                             }
342                             else
343                             {
344                                 if (p.prevItemPin == "+")
345                                     positives.Add("i_" + p.Name);
346                                 else
347                                     negatives.Add("i_" + p.Name);
348                             }
349                         }
350                     }
351                     //eq += " = ";
352                     foreach (ModelItem p in item.PrevItems)
353                     {
354                         if (p.NextItems.FirstOrDefault(x => x.Name == item.Name) != null)
355                         {
356                             if (p.Type == "SineVoltage" || p.Type == "PulseVoltage" ||
357 p.Type == "PulseCurrent")
358                             {
359                                 if (p.nextItemPin == "+")
360                                     negatives.Add("i_" + p.Name);
361                                 else
```

```
362                                     positives.Add("i_" + p.Name);
363                                 }
364                                 else
365                                 {
366                                     if (p.nextItemPin == "+")
367                                         positives.Add("i_" + p.Name);
368                                     else
369                                         negatives.Add("i_" + p.Name);
370                                 }
371                             }
372                             else
373                             {
374                                 if (p.Type == "SineVoltage" || p.Type == "PulseVoltage" ||
375 p.Type == "PulseCurrent")
376                                 {
377                                     if (p.prevItemPin == "+")
378                                         negatives.Add("i_" + p.Name);
379                                     else
380                                         positives.Add("i_" + p.Name);
381                                 }
382                                 else
383                                 {
384                                     if (p.prevItemPin == "+")
385                                         positives.Add("i_" + p.Name);
386                                     else
387                                         negatives.Add("i_" + p.Name);
388                                 }
389                             }
390                         }
391 
392                         positives.Sort();
393                         negatives.Sort();
394                         if (positives.Count == 0) eq = "0";
395                         foreach (string i in positives)
396                         {
397                             eq += i;
398                             if (positives.Last() != i)
399                                 eq += " + ";
400                         }
401                         eq += " = ";
402                         if (negatives.Count == 0) eq += "0";
403                         foreach (string i in negatives)
404                         {
405                             eq += i;
406                             if (negatives.Last() != i)
407                                 eq += " + ";
408                         }
409 
410                         if (!nodeEquations.Contains("  " + eq))
411                             nodeEquations.Add("  " + eq);
412                     }
413 
414                     else if (item.Type != "Node" && item.Type != "Ground" && item.Type !=
415 "Connector")
416                     {
417                         if (item.NextItems.FirstOrDefault(x => x.Type == "Ground" || x.Type ==
418 "Node") == null)
419                         {
420                             List<string> positives = new List<string>();
421                             List<string> negatives = new List<string>();
422                             if (item.nextItemPin == "-")
423                                 negatives.Add("i_" + item.Name);
424                             else
```

```
425                             positives.Add("i_" + item.Name);
426
427                         string eq = string.Empty;
428                         foreach (ModelItem p in item.NextItems)
429                         {
430                             if (p.NextItems.FirstOrDefault(x => x.Name == item.Name) !=
431 null)
432                             {
433                                 if (p.Type == "SineVoltage" || p.Type == "PulseVoltage" ||
434 p.Type == "PulseCurrent")
435                                 {
436                                     if (p.nextItemPin == "+")
437                                         negatives.Add("i_" + p.Name);
438                                     else
439                                         positives.Add("i_" + p.Name);
440                                 }
441                                 else
442                                 {
443                                     if (p.nextItemPin == "+")
444                                         positives.Add("i_" + p.Name);
445                                     else
446                                         negatives.Add("i_" + p.Name);
447                                 }
448                             }
449                             else
450                             {
451                                 if (p.Type == "SineVoltage" || p.Type == "PulseVoltage" ||
452 p.Type == "PulseCurrent")
453                                 {
454                                     if (p.prevItemPin == "+")
455                                         negatives.Add("i_" + p.Name);
456                                     else
457                                         positives.Add("i_" + p.Name);
458                                 }
459                                 else
460                                 {
461                                     if (p.prevItemPin == "+")
462                                         positives.Add("i_" + p.Name);
463                                     else
464                                         negatives.Add("i_" + p.Name);
465                                 }
466                             }
467                         }
468
469                         if (positives.Count == 0) eq = "0";
470                         positives.Sort();
471                         negatives.Sort();
472                         foreach (string i in positives)
473                         {
474                             eq += i;
475                             if (positives.Last() != i)
476                                 eq += " + ";
477                         }
478                         eq += " = ";
479                         if (negatives.Count == 0) eq += "0";
480                         foreach (string i in negatives)
481                         {
482                             eq += i;
483                             if (negatives.Last() != i)
484                                 eq += " + ";
485                         }
486
487                         if (!nodeEquations.Contains("  " + eq))
```

```
488                             nodeEquations.Add("  " + eq);
489                     }
490                 }
491                 // Los condensadores usan el nodo de referencia mas cercano para declarar
492 el voltaje equivalente
493                 else if (item.Type == "Capacitor") //elementos en serie
494                 {
495 
496                     if (item.NextItems.Count == 1 && item.NextItems[0].Type == "Ground" ||
497 item.NextItems[0].Type == "Node")
498                     {
499                         continue;
500                     }
501                     if (item.NextItems.Count == 1)
502                     {
503                         string eq = string.Empty;
504                         eq += "i_" + item.Name + " = " + "i_" + item.NextItems[0].Name;
505                         if (!nodeEquations.Contains("  " + eq))
506                             nodeEquations.Add("  " + eq);
507                     }
508                 }
509 
510             }
511 
512             foreach (string eq in nodeEquations)
513             {
514                 result += eq + ";\n";
515             }
516 
517             return result;
518         }
519 
520         public string ConstitutiveRelations(List<ModelItem> objects, bool SpanishComments,
521 string result)
522         {
523             List<string> nodeEquations = new List<string>();
524             //            // Relaciones constitutivas
525             result += "// Relaciones constitutivas\n";
526             nodeEquations = new List<string>();
527             foreach (ModelItem item in objects)
528             {
529                 // Busco los nodos mas cercanos a cada elemento
530                 if (item.Type == "Resistor" || item.Type == "Inductor" || item.Type ==
531 "Capacitor")
532                 {
533                     string eq = string.Empty;
534                     if (item.Type == "Capacitor")
535                     {
536                         if (item.Type == "Capacitor")
537                         {
538                             eq = string.Empty;
539                             eq = "i_" + item.Name;
540                             eq += " = " + item.Equation;
541                             nodeEquations.Add("  " + eq);
542                             eq = string.Empty;
543                         }
544 
545                     }
546                     eq = GetVoltage(item);
547 
548                     if (item.Type != "Capacitor")
549                     {
550                         eq += " = " + item.Equation;
```

```
551                     }
552                     else
553                     {
554                         eq += " = u_" + item.Name;
555                     }
556                     nodeEquations.Add("  " + eq);
557                 }
558 
559                 else if (item.Type == "Node")
560                 {
561                     string eq = string.Empty;
562                     foreach (ModelItem p in item.PrevItems)
563                     {
564                         if (p.Type == "Node")
565                         {
566                             eq += p.Name;
567                             eq += " = " + item.Name;
568                             nodeEquations.Add("  " + eq);
569                         }
570                     }
571                 }
572 
573                 else if (item.Type == "SineVoltage")
574                 {
575                     string eq = GetVoltage(item);
576                     eq += " = " + item.Equation;
577 
578                     nodeEquations.Add("  " + eq);
579                 }
580 
581                 else if (item.Type == "PulseVoltage")
582                 {
583                     string eqTime = string.Empty;
584                     eqTime = @"  when sample(0,T" + item.Name + @") then
585   up_" + item.Name + @" = not pre(up_" + item.Name + @");
586 end when;
587 u_" + item.Name + " = if up_" + item.Name + " then U" + item.Name + @" else 0";
588 
589 
590                     nodeEquations.Add(eqTime);
591 
592                     string eq = GetVoltage(item);
593                     eq += " = u_" + item.Name;
594 
595                     nodeEquations.Add("  " + eq);
596                 }
597 
598                 else if (item.Type == "PulseCurrent")
599                 {
600                     string eqTime = string.Empty;
601                     eqTime = @"  when sample(0,T" + item.Name + @") then
602   up_" + item.Name + @" = not pre(up_" + item.Name + @");
603 end when;
604 i_" + item.Name + " = if up_" + item.Name + " then I" + item.Name + @" else 0";
605 
606 
607                     nodeEquations.Add(eqTime);
608 
609                     string eq = GetVoltage(item);
610                     eq += " = u_" + item.Name;
611 
612                     nodeEquations.Add("  " + eq);
613                 }
```

```
614
615                 else if (item.Type == "Diode")
616                 {
617                     string u = GetVoltage(item);
618
619                     string eq = string.Empty;
620                     eq += "i_" + item.Name + "  = " + item.Equation.Replace("u", "(" + u +
621 ")");
622                     nodeEquations.Add("  " + eq);
623                 }
624
625             }
626
627
628             foreach (ModelItem item in objects)
629             {
630                 if (item.Type == "Switch")
631                 {
632                     string eq = string.Empty;
633                     eq += item.Equation + "";
634                     foreach (ModelItem p in item.PrevItems)
635                     {
636                         if (p.Type == "Node" || p.Type == "SineVoltage" || p.Type ==
637 "PulseVoltage")
638                         {
639                             eq += p.Name;
640                         }
641                     }
642                     eq += " - ";
643                     foreach (ModelItem p in item.NextItems)
644                     {
645                         if (p.Type == "Node" || p.Type == "SineVoltage" || p.Type ==
646 "PulseVoltage" || p.Type == "Ground")
647                         {
648                             eq += p.Name;
649                         }
650                     }
651                     nodeEquations.Add("  " + eq);
652
653                     string eqTime = string.Empty;
654                     eqTime = @"  when sample(0,T" + item.Name + @") then
655     off_" + item.Name + @" = not pre(off_" + item.Name + @");
656   end when;
657   i_";
658                     eqTime += item.Name;
659                     eqTime += @" = (s*unitVoltage)*(if off_" + item.Name + @" then Goff
660 else 1);";
661                     eqTime += "\n" + "  v_" + item.Name + " = (s*unitCurrent)*(if off_" +
662 item.Name + @" then 1 else Ron)";
663                     nodeEquations.Add(eqTime);
664                 }
665             }
666
667             foreach (string eq in nodeEquations)
668             {
669                 result += eq + ";\n";
670             }
671             return result;
672         }
673
674         public string GetVoltage(ModelItem item)
675         {
676             string u1 = string.Empty;
```

```
677             string u2 = string.Empty;
678             string eq = string.Empty;
679             List<string> positives = new List<string>();
680             List<string> negatives = new List<string>();
681 
682             ///TODO
683             if (FindPrevNode(item, item, 0) != string.Empty && FindPrevNode(item, item, 0)
684 != "0")
685             {
686                 if (item.prevItemPin == "+")
687                     positives.Add(FindPrevNode(item, item, 0));
688                 else
689                     negatives.Add(FindPrevNode(item, item, 0));
690             }
691             if (FindNextNode(item, item, 0) != string.Empty && FindNextNode(item, item, 0)
692 != "0")
693             {
694                 if (item.nextItemPin == "+")
695                     positives.Add(FindNextNode(item, item, 0));
696                 else
697                     negatives.Add(FindNextNode(item, item, 0));
698             }
699 
700             if (positives.Count == 0) eq = "";
701             foreach (string i in positives)
702             {
703                 eq += i;
704                 if (positives.Last() != i)
705                     eq += " + ";
706             }
707             if (negatives.Count != 0)
708             {
709                 eq += " - ";
710                 if (negatives.Count > 1)
711                     eq += "(";
712                 foreach (string i in negatives)
713                 {
714                     eq += i;
715                     if (negatives.Last() != i)
716                         eq += " + ";
717                 }
718                 if (negatives.Count > 1)
719                     eq += ")";
720             }
721             return eq;
722         }
723 
724         //Busca el nodo inmediatamente anterior
725         public string FindPrevNode(ModelItem item, ModelItem root, int iteration)
726         {
727             string u = string.Empty;
728             iteration += 1;
729             if (iteration >= 4) return u;
730             if (item.PrevItems != null && item.PrevItems.FirstOrDefault(x => x.Key ==
731 root.Key) == null)
732             {
733                 foreach (ModelItem p in item.PrevItems)
734                 {
735                     if (p.Type == "Node")
736                     {
737                         u = p.Name;
738                     }
739                     else if (p.Type == "Ground")
```

```
740                     {
741                         u = "0";
742                     }
743                     else
744                     {
745                         if (string.IsNullOrEmpty(u))
746                         {
747                             u = FindPrevNode(p, root, iteration);
748                         }
749                     }
750                 }
751             }
752             else if (item.NextItems != null && item.NextItems.FirstOrDefault(x => x.Key ==
753 root.Key) == null)
754             {
755                 foreach (ModelItem p in item.NextItems)
756                 {
757                     if (p.Type == "Node")
758                     {
759                         u = p.Name;
760                     }
761                     else if (p.Type == "Ground")
762                     {
763                         u = "0";
764                     }
765                     else
766                     {
767                         if (string.IsNullOrEmpty(u))
768                         {
769                             u = FindPrevNode(p, root, iteration);
770                         }
771                     }
772                 }
773             }
774 
775             return u;
776         }
777 
778         //Busca el nodo inmediatamente posterior
779         public string FindNextNode(ModelItem item, ModelItem root, int iteration)
780         {
781             string u = string.Empty;
782             iteration += 1;
783             if (iteration >= 4) return u;
784             if (item.NextItems != null && item.NextItems.FirstOrDefault(x => x.Key ==
785 root.Key) == null)
786             {
787                 foreach (ModelItem p in item.NextItems)
788                 {
789                     if (p.Type == "Node")
790                     {
791                         u = p.Name;
792                     }
793                     else if (p.Type == "Ground")
794                     {
795                         u = "0";
796                     }
797                     else
798                     {
799                         if (string.IsNullOrEmpty(u))
800                         {
801                             u = FindNextNode(p, root, iteration);
802                         }
```

```
803                     }
804                 }
805             }
806             else if (item.PrevItems != null && item.PrevItems.FirstOrDefault(x => x.Key ==
807 root.Key) == null)
808             {
809                 foreach (ModelItem p in item.PrevItems)
810                 {
811                     if (p.Type == "Node")
812                     {
813                         u = p.Name;
814                     }
815                     else if (p.Type == "Ground")
816                     {
817                         u = "0";
818                     }
819                     else
820                     {
821                         if (string.IsNullOrEmpty(u))
822                         {
823                             u = FindNextNode(p, root, iteration);
824                         }
825                     }
826                 }
827 
828             }
829             return u;
830         }
831 
832         //Suma todos los componentes en serie
833         public void FindNextSerie(ModelItem item, ref string eq)
834         {
835             if (item.NextItems.Count == 1)
836             {
837                 foreach (ModelItem p2 in item.NextItems)
838                 {
839                     if (p2.Type != "Node" && p2.Type != "Ground" && p2.Type != "Capacitor")
840                     {
841                         if (p2.NextItems.FirstOrDefault(x => x.Type == "Ground") != null)
842                         {
843                             eq += " + (" + p2.Equation + ")";
844                             FindNextSerie(item.NextItems[0], ref eq);
845                         }
846                     }
847                     else if (p2.Type == "Capacitor")
848                     {
849                         eq += " + u_" + p2.Name;
850                         FindNextSerie(item.NextItems[0], ref eq);
851                     }
852                 }
853             }
854         }
855 
856     }
857 }
```

B-3 CompositeService.cs

El fichero CompositeService.css desarrolla el código de traducción al modelo compuesto

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using WebAnalogCircuits.Model;

namespace WebAnalogCircuits.Service
{
    public class CompositeService
    {
        public string TranslateModel(List<ModelItem> objects, string name)
        {
            bool SpanishComments = true;
            string result = string.Empty;
            result += "model " + name + "Composite\n";
            result += "import Analog = Modelica.Electrical.Analog;\n";

            result = DeclareComponents(objects, SpanishComments, result);

            result += "\nequation\n";
            result = DeclareConnections(objects, SpanishComments, result);

            result += "\nannotation(\n\tuses(Modelica(version = \"4.0.0\")),\n" +
                "\tDiagram(coordinateSystem(extent = { { 0, 0}, { 1400, 1400} })),\n" +
                "\tIcon(coordinateSystem(extent = {{0, 0}, {1400, 1400}})),\n" +
                "\tversion = \"\");\n\n";


            result += "end " + name + "Composite;\n";

            return result;
        }

        public string DeclareComponents(List<ModelItem> objects, bool SpanishComments,
string result)
        {
            foreach (ModelItem item in objects)
            {
                if (item.Type == "Connector") continue;
                if (item.Type == "Ground")
                {
                    result += "Analog.Basic.Ground " + item.Name + "\n annotation(";
                    decimal offset = 0;
                    if (item.Rotation == -90) { offset = -20; }
```

```
                    result += "Placement(visible = true, transformation(\n\torigin = { " +
(item.PositionX + offset).ToString().Replace(",", ".") + ", " +
item.PositionY.ToString().Replace(",", ".") + "},\n\textent = { { -20, -20}, { 20, 20}
},\n\trotation = " + item.Rotation.ToString().Replace(",", ".") + ")));\n\n";
                }
                else if (item.Type == "Resistor")
                {
                    result += "Analog.Basic.Resistor " + item.Name + "(R = " +
GetValue(item, "Resistance") + ") \n annotation(";
                }
                else if (item.Type == "Inductor")
                {
                    result += "Analog.Basic.Inductor " + item.Name + "(L = " +
GetValue(item, "Inductance") + ") \n annotation(";
                }
                else if (item.Type == "Capacitor")
                {
                    result += "Analog.Basic.Capacitor " + item.Name + "(C = " +
GetValue(item, "Capacitance") + ")\n annotation(";
                }
                else if (item.Type == "Diode")
                {
                    result += "Analog.Semiconductors.Diode  " + item.Name + "(Ids = " +
GetValue(item, "Current") + ", Vt = " + GetValue(item, "Voltage") + ")\n annotation(";
                }
                else if (item.Type == "SineVoltage")
                {
                    result += "Analog.Sources.SineVoltage " + item.Name +
                        "(V = " + GetValue(item, "Voltage") +
                        ", offset = 0 " +
                        ", f = " + GetValue(item, "Frequency") +
                        ")\n annotation(";
                }
                else if (item.Type == "PulseVoltage")
                {
                    result += "Analog.Sources.PulseVoltage " + item.Name +
                        "(V = " + GetValue(item, "Voltage") +
                        ", offset = 0 " +
                        ", period = " + GetValue(item, "Time") +
                        ")\n annotation(";
                }
                else if (item.Type == "PulseCurrent")
                {
                    result += "Analog.Sources.PulseCurrent " + item.Name +
                        "(I = " + GetValue(item, "Current") +
                        ", period = " + GetValue(item, "Time") + ")\n annotation(";
                }
                else if (item.Type == "Node")
                {
                    result += "Analog.Interfaces.NegativePin " + item.Name +

```

```
93                          "\n annotation(Placement(transformation(extent ={ { " +
94                          (item.PositionX - 10).ToString().Replace(",", ".") + "," +
95                          (item.PositionY - 10).ToString().Replace(",", ".") + "},{ " +
96                          (item.PositionX + 10).ToString().Replace(",", ".") + "," +
97                          (item.PositionY + 10).ToString().Replace(",", ".") + "} }})));\n\n";
98                  }
99                  else if (item.Type == "Switch")
100                 {
101                     result += "Modelica.Blocks.Sources.BooleanPulse booleanPulse(period= " +
item.Parameters.FirstOrDefault(x => x.Magnitude == "Time").Value + ");\n";
102
103                     result += "Analog.Ideal.IdealOpeningSwitch " + item.Name +
104                         "\n annotation(";
105                 }
106                 if (item.Type != "Node" && item.Type != "Ground")
107                 {
108                     result += "Placement(visible = true, transformation(\n\torigin = { " +
item.PositionX.ToString().Replace(",", ".") + ", " + item.PositionY.ToString().Replace(",",
109 ".") + "},\n\textent = { { -20, -20}, { 20, 20} },\n\trotation = " +
item.Rotation.ToString().Replace(",", ".") + "})));\n\n";
110                 }
111             }
112
113
114             return result;
115         }
116
117         public string DeclareConnections(List<ModelItem> objects, bool SpanishComments,
118 string result)
119         {
120             foreach (ModelItem item in objects)
121             {
122
123                 if (item.Type == "Switch")
124                 {
125                     result += "connect(booleanPulse.y, " + item.Name + ".control);\n";
126                 }
127                 if (item.Type == "Connector")
128                 {
129                     bool change = false;
130                     if (item.PrevItems != null && item.PrevItems.Count > 0 &&
131 item.PrevItems[0].PrevItems != null && item.NextItems != null && item.NextItems.Count > 0 &&
item.NextItems[0].NextItems != null)
132                     {
133                         if ((item.PrevItems[0].PrevItems.FirstOrDefault(x => x.Name ==
134 item.NextItems[0].Name) != null) &&
135                             (item.NextItems[0].NextItems.FirstOrDefault(x => x.Name ==
item.PrevItems[0].Name) != null))
136                         {
137                             change = true;
138                         }
139                     }
140
```

```
141                     if (item.LastConnection && item.PrevItems[0].Type == "Ground")
142                         change = true;
143                     if (change)
144                     {
145                         List<ModelItem> aux = new List<ModelItem>();
146                         aux = item.NextItems;
147                         item.NextItems = item.PrevItems;
148                         item.PrevItems = aux;
149                     }
150 
151                     string prevPin = ".p";
152                     string nextPin = ".n";
153 
154                     ModelItem item1 = item.NextItems[0];
155                     ModelItem item2 = item.PrevItems[0];
156 
157 
158                     if (item1.PrevItems != null && item1.PrevItems.FirstOrDefault(x => x.Key
    == item2.Key) != null)
159                     {
160                         if (item1.prevItemPin == "+")
161                             nextPin = ".p";
162                         else nextPin = ".n";
163                     }
164                     else if (item1.NextItems != null && item1.NextItems.FirstOrDefault(x =>
    x.Key == item2.Key) != null)
165                     {
166                         if (item1.nextItemPin == "+")
167                             nextPin = ".p";
168                         else nextPin = ".n";
169                     }
170 
171                     if (item2.PrevItems != null && item2.PrevItems.FirstOrDefault(x => x.Key
    == item1.Key) != null)
172                     {
173                         if (item2.prevItemPin == "+")
174                             prevPin = ".p";
175                         else prevPin = ".n";
176                     }
177                     else if (item2.NextItems != null && item2.NextItems.FirstOrDefault(x =>
    x.Key == item1.Key) != null)
178                     {
179                         if (item2.nextItemPin == "+")
180                             prevPin = ".p";
181                         else prevPin = ".n";
182                     }
183 
184 
185                     if (item.PrevItems[0].Type == "Ground")
186 
187 
188 
```

```
189                     {
190                         prevPin = ".p";
191                     }
192                     if (item.NextItems[0].Type == "Ground")
193                     {
194                         nextPin = ".p";
195                     }
196                     if (item.PrevItems[0].Type == "Node")
197                     {
198                         prevPin = "";
199                     }
200                     if (item.NextItems[0].Type == "Node")
201                     {
202                         nextPin = "";
203                     }
204 
205                     result += "connect(" + item.PrevItems[0].Name + prevPin + ", " +
    item.NextItems[0].Name + nextPin + ")";
206 
207                     result += " annotation (Line(points ={";
208 
209                     if (item.PrevItems[0].Rotation == -90)
210                     {
211                         result += "{" +
    (item.PrevItems[0].PositionX).ToString().Replace(",", ".") + "," +
212 (item.PrevItems[0].PositionY).ToString().Replace(",", ".") + "}";
213                     }
214                     else if (item.PrevItems[0].Rotation == 90)
215                     {
216                         result += "{" +
    (item.PrevItems[0].PositionX).ToString().Replace(",", ".") + "," +
217 (item.PrevItems[0].PositionY).ToString().Replace(",", ".") + "}";
218                     }
219                     else if (item.PrevItems[0].Rotation == 0)
220                     {
221                         result += "{" +
    (item.PrevItems[0].PositionX).ToString().Replace(",", ".") + "," +
222 (item.PrevItems[0].PositionY).ToString().Replace(",", ".") + "}";
223                     }
224                     else
225                     {
226                         result += "{" +
    (item.PrevItems[0].PositionX).ToString().Replace(",", ".") + "," +
227 (item.PrevItems[0].PositionY).ToString().Replace(",", ".") + "}";
228                     }
229 
230                     if (item.Points.Count > 0)
231                     {
232                         if (Math.Abs(item.Points[0].PositionX - item.PrevItems[0].PositionX)
    <= 20 || Math.Abs(item.Points[0].PositionY - item.NextItems[0].PositionY) <= 20)
233 
234                         {
235 
236 
```

```
237                             result += ",{" +
    (item.PrevItems[0].PositionX).ToString().Replace(",", ".") + "," +
238 (item.NextItems[0].PositionY).ToString().Replace(",", ".") + "}";
239                         }
240                         else if (Math.Abs(item.Points[0].PositionY -
241 item.PrevItems[0].PositionY) <= 20 || Math.Abs(item.Points[0].PositionX -
    item.NextItems[0].PositionX) <= 20)
242                         {
243                             result += ",{" +
244 (item.NextItems[0].PositionX).ToString().Replace(",", ".") + "," +
    (item.PrevItems[0].PositionY).ToString().Replace(",", ".") + "}";
245                         }
246                         else if (Math.Abs(item.Points[0].PositionX -
247 item.PrevItems[0].PositionX) < Math.Abs(item.Points[0].PositionX -
248 item.NextItems[0].PositionX))
249                         {
250                             result += ",{" +
    (item.PrevItems[0].PositionX).ToString().Replace(",", ".") + "," +
251 (item.NextItems[0].PositionY).ToString().Replace(",", ".") + "}";
252                         }
253                         else if (Math.Abs(item.Points[0].PositionY -
254 item.PrevItems[0].PositionY) < Math.Abs(item.Points[0].PositionY -
    item.PrevItems[0].PositionY))
255                         {
256                             result += ",{" +
257 (item.NextItems[0].PositionX).ToString().Replace(",", ".") + "," +
    (item.PrevItems[0].PositionY).ToString().Replace(",", ".") + "}";
258                         }
259                         else
260                         {
261                             foreach (ItemPoint p in item.Points)
262                             {
263                                 result += ",{" + p.PositionX.ToString().Replace(",", ".") +
264 "," + p.PositionY.ToString().Replace(",", ".") + "}";
265                             }
266                         }
267                     }
268                     else
269                     {
270                         if (item.PrevItems[0].PositionX != item.NextItems[0].PositionX &&
271 item.PrevItems[0].PositionY != item.NextItems[0].PositionY)
272                         {
273                             result += ",{" +
274 (item.NextItems[0].PositionX).ToString().Replace(",", ".") + "," +
275 (item.NextItems[0].PositionY).ToString().Replace(",", ".") + "}";
276                         }
277                     }
278                 }
279
280
281
282                 if (item.NextItems[0].Rotation == -90)
283                 {
284
285                     result += ",{" +
286 (item.NextItems[0].PositionX).ToString().Replace(",", ".") + "," +
287 (item.NextItems[0].PositionY).ToString().Replace(",", ".") + "}";
288                 }
289                 else if (item.NextItems[0].Rotation == 90)
290
291                 {
292
```

```
                        result += ",{" +
(item.NextItems[0].PositionX).ToString().Replace(",", ".") + "," +
(item.NextItems[0].PositionY).ToString().Replace(",", ".") + "}";

                    }
                    else if (item.NextItems[0].Rotation == 0)
                    {
                        result += ",{" +
(item.NextItems[0].PositionX).ToString().Replace(",", ".") + "," +
(item.NextItems[0].PositionY).ToString().Replace(",", ".") + "}";
                    }


                    result += "},";
                    result += " color ={ 0,0,255}))";
                    result += ";\n";
                }
            }

            return result;
        }

        private string GetValue(ModelItem item, string magnitude)
        {
            return item.Parameters.FirstOrDefault(x => x.Magnitude ==
magnitude).Value.ToString().Replace(",", ".");
        }

    }
}
```