Universidad Nacional de Educación a Distancia
(UNED)

Escuela Técnica Superior de Ingeniería Informática

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

Editor de Modelos Hidráulicos en Lenguaje Modelica

Autor:

Jackson Fabian Reyes Bermeo

Directores:

Alfonso Urquía Moraleda

Carla Martín Villalba

Curso: 2022-2023 - Convocatoria: Septiembre

Dedicado a mi familia Reyes Bermeo.

AGRADECIMIENTOS

En primer lugar, quiero agradecer a mis directores de proyecto: Alfonso Urquía Moraleda y Carla Martín Villalba, por darme la oportunidad, el apoyo e inspiración para desarrollar el presente Trabajo Final de Máster (TFM), proporcionándome las herramientas necesarias, aclarando las dudas y ofreciéndome posibles soluciones a los problemas que se presentarón durante el desarrollo de de este trabajo.

También me gustaría agradecer a todos los profesores del presente máster, que de una o de otra manera me han impartido sus conocimientos, experiencias y recomendaciones que me han permitido obtener una formación integral; a mis amigos y compañeros de clase que aun sin conocemos presencialmente, siempre hemos estado en contacto telemáticamente, compartiendo ideas, dudas, conocimientos y sobre todo la experiencia de trabajar en equipo a pesar de la distancia.

Por último, me gustaría expresar mi más grande agradecimiento a mi familia Reyes Bermeo y a mi pareja Gabriela, por estar en cada momento, acompañándome, pendiente de mí y testigos del desarrollo de este trabajo, que junto a mi esfuerzo ha sido posible llegar a su etapa final.

Jackson Fabian Reyes Bermeo.

RESUMEN

La inmersión en el mundo del modelado y la simulación puede resultar intimidante debido a la necesidad de comprender diversos campos del conocimiento, como el modelado, las matemáticas, la programación y los dominios físicos relacionados. En este contexto, este proyecto se enfoca en el desarrollo de una aplicación o herramienta diseñada para simplificar la creación de modelos en lenguaje Modelica a través de una interfaz gráfica intuitiva y fácil de usar. Está diseñada para usuarios de distintos niveles de experiencia, especialmente aquellos que se están introduciendo en el mundo del modelado y la simulación.

Esta aplicación se basa en la técnica "Drag and Drop"(arrastrar y soltar) para componer modelos. Permite a los usuarios arrastrar componentes desde una paleta de componentes Modelica y colocarlos en un lienzo de diseño que facilita la construcción de modelos compuestos mediante conexiones entre ellos. Esta forma de diseñar modelos reduce la necesidad de comprender la sintaxis de Modelica. Además, la herramienta permite seleccionar, mover y eliminar componentes, así como realizar conexiones entre ellos y gestionar dichas conexiones. La aplicación no solo facilita la visualización, edición y almacenamiento de modelos, sino que también elimina la necesidad de tener un compilador Modelica instalado, lo que aumenta la portabilidad de la aplicación.

Esta herramienta, denominada **FluidEditor v0.1**, ha sido desarrollada en el lenguaje de programación Java y se ha diseñado específicamente para la creación de modelos hidráulicos utilizando los componentes descritos en la librería Fluid, que forma parte de las librerías estándar de Modelica (MSL). La aplicación tiene la capacidad de extraer los componentes de los archivos propios de la librería Fluid y representarlos visualmente como iconos en una paleta de componentes. Estos iconos mejoran la identificación y comprensión en comparación con enfoques tradicionales que se basan únicamente en etiquetas de texto o cajas de difícil comprensión visual. Además, cada componente del diseño permite la edición de sus parámetros correspondientes mediante una ventana de configuración a la que se accede con un doble clic en el componente seleccionado. La herramienta también proporciona una zona en donde se puede visualizar automáticamente el código Modelica generado que corresponde al diseño realizado. Este código incluye instancias de los componentes de la librería estándar de Modelica (MSL) y descripciones de las conexiones entre componentes. Los modelos generados pueden guardarse en un archivo Modelica y

recuperarse para futuras ediciones. Además, son compatibles con entornos de simulación como OpenModelica, Dymola o Wolfram System Modeler.

El desarrollo de la aplicación siguió una metodología iterativa incremental, empleando el patrón de diseño Modelo-Vista-Controlador (MVC) y aplicando principios de programación orientada a objetos (POO). El proceso se dividió en varias etapas, que incluyeron el desarrollo de la capacidad para extraer modelos de los archivos de la librería Fluid, analizar y extraer información del código de cada modelo, crear una librería que permitiera generar iconos a partir de la información de las anotaciones Modelica y desarrollar la capacidad de la aplicación para generar código Modelica a partir del modelo diseñado. Para validar la aplicación, se reprodujeron ejemplos de la propia librería Fluid y se llevaron a cabo pruebas en entornos de modelado y simulación, como OpenModelica y Wolfram System Modeler, obteniendo resultados satisfactorios. Además, se incluye un manual rápido en los anexos como parte de la documentación de la misma.

Palabras clave

Modelado y Simulación, Lenguaje Modelica, Java, JavaFx, Interfaz Gráfica de Usuario, Programación Orientada a Objetos, Modelado Matemático, Hidráulica.

ABSTRACT

Immersing oneself in the world of modeling and simulation can be intimidating due to the need to comprehend various fields of knowledge, such as modeling, mathematics, programming, and related physical domains. In this context, this project focuses on the development of an application or tool designed to simplify the creation of models in the Modelica language through an intuitive and user-friendly graphical interface. It is designed for users of different experience levels, especially those who are venturing into the realm of modeling and simulation.

This application is based on the "Drag and Drop"technique to compose models. It allows users to drag components from a palette of Modelica components and place them on a design canvas, facilitating the construction of composite models through connections between them. This approach to modeling reduces the need to understand Modelica syntax. Furthermore, the tool enables users to select, move, and delete components, as well as establish connections between them and manage those connections. The application not only facilitates the visualization, editing, and storage of models but also eliminates the need for an installed Modelica compiler, enhancing its portability.

This tool, named **FluidEditor v0.1**, has been developed in the Java programming language and is specifically designed for creating hydraulic models using components described in the Fluid library, which is part of the Modelica Standard Library (MSL). The application has the capability to extract components from the files of the Fluid library and visually represent them as icons in a component palette. These icons enhance identification and comprehension compared to traditional approaches relying solely on text labels or visually complex boxes. Additionally, each component in the design allows for editing its corresponding parameters through a configuration window accessible by double-clicking the selected component. The tool also provides an area where the automatically generated Modelica code corresponding to the design can be viewed. This code includes instances of Modelica Standard Library (MSL) components and descriptions of connections between components. Generated models can be saved in a Modelica file and retrieved for future editing. Furthermore, they are compatible with simulation environments such as OpenModelica, Dymola, or Wolfram System Modeler.

The development of the application followed an iterative incremental methodology, utilizing the Model-View-Controller (MVC) design pattern and applying object-oriented programming (OOP) principles. The process was divided into several stages, including developing the capability to extract models from Fluid library files, analyzing and extracting information from the code of each model, creating a library that allowed for generating icons from Modelica annotation information, and developing the application's ability to generate Modelica code from the designed model. To validate the application, examples from the Fluid library were reproduced and tested in modeling and simulation environments like OpenModelica and Wolfram System Modeler, yielding satisfactory results. Additionally, a quick guide manual is included in the appendices as part of the application's documentation.

Keywords

Modeling and Simulation, Modelica Language, Java, JavaFX, Graphic User Interface, Object-Oriented Programming, Mathematical Modeling, Hydraulics.

Índice General

Lista de Figuras

Lista de Tablas

Lista de Códigos

1 INTRODUCCIÓN, OBJETIVOS Y ESTRUCTURA

1.1. Introducción

El presente trabajo, titulado **Editor de Modelos Hidráulicos en Lenguaje Modelica** se enfoca en el desarrollo e implementación de una aplicación en Java diseñada para simplificar la creación, modificación y almacenamiento de modelos hidráulicos. Esta aplicación utiliza componentes visuales en forma de iconos para representar modelos en el lenguaje Modelica, centrándose específicamente en componentes del paquete **Fluid** de la librería estándar Modelica (MSL). La aplicación recibe el nombre de **FluidEditor v0.1** o simplemente **FluidEditor**.

FluidEditor tendrá la capacidad de leer, extraer y generar una jerarquía de componentes en forma de un árbol de iconos a partir de los archivos Modelica que conforman el paquete **Fluid**. Esta estructura de componentes permite a los usuarios seleccionar, arrastrar, soltar y conectar elementos en el área de diseño de la aplicación y crear modelos compuestos de manera intuitiva. Además, la aplicación podrá generar automáticamente el código Modelica correspondiente al modelo compuesto diseñado. Este código Modelica será portable y se puede utilizar en diversos entornos de modelado y simulación.

Un editor gráfico de modelos Modelica se ha convertido en una herramienta esencial en campos como la ingeniería, la ciencia y la investigación, especialmente en la simulación de sistemas complejos. Su utilidad radica en simplificar el proceso de modelado y simulación al proporcionar a los usuarios una interfaz visual e intuitiva para diseñar, construir y validar modelos Modelica. Esto permite a los usuarios centrarse en la representación conceptual y estructural del sistema en lugar de preocuparse por la sintaxis de Modelica o los aspectos técnicos de la programación. Esto es especialmente beneficioso para quienes se inician en la simulación y pueden sentirse abrumados por la necesidad de dominar múltiples áreas de conocimiento, como matemáticas aplicadas, modelado y ciencias de la computación.

Actualmente, existen diversos editores gráficos de modelos Modelica en el mercado que ofrecen herramientas para edición, análisis, validación y simulación. Estos entornos están disponibles en versiones de código abierto, como OpenModelica [OpenModelica, 2023], y en versiones comerciales, como Dymola [Dassault-Systèmes, 2023] y Wolfram System Modeler [Wolfram, 2023]. Para obtener información adicional sobre estos y otros entornos de modelado y simulación, se puede visitar el sitio web oficial de la Modelica Association [Modelica Association, 2023b], que proporciona una amplia documentación.

Es importante destacar que el desarrollo de FluidEditor no busca competir con estos entornos existentes. Mas bien, intenta contribuir como una ayuda a los usuarios a familiarizarse con el proceso de diseño de modelos en Modelica de manera gráfica, ofreciendo una solución sencilla e intuitiva para crear, editar y guardar sus propios modelos sin la necesidad de utilizar herramientas complejas que puedan resultar confusas, especialmente para quienes se inician en este campo.

1.2. Objetivos

El objetivo principal de este trabajo consiste en diseñar, desarrollar e implementar una aplicación o herramienta de software que permita a los usuarios crear modelos hidráulicos compuestos de manera sencilla y visual. Esto se logrará mediante la capacidad de arrastrar y conectar componentes disponibles en una paleta o árbol de componentes. La aplicación, que la hemos denominado **FluidEditor**, también tendrá la capacidad de generar automáticamente el código Modelica asociado al modelo compuesto, eliminando la necesidad de depender del compilador Modelica. Este código será funcional, permitiendo la ejecución en entornos de modelado como OpenModelica, Dymola, Wolfram System Modeler, entre otros.

Para alcanzar este objetivo principal, se han establecido los siguientes subobjetivos:

- **Edición del modelo:** La aplicación permitirá a los usuarios seleccionar, mover y eliminar componentes, así como añadir, seleccionar y eliminar conexiones.
- **Configuración de parámetros:** Los usuarios podrán asignar y modificar los valores de los parámetros de cada componente de forma flexible.
- **Generación de código Modelica:** La aplicación estará capacitada para generar automáticamente el código Modelica correspondiente al modelo compuesto realizado, asegurando su utilidad y funcionalidad.

- **Funcionalidad del código:** Los modelos generados por el editor serán simulables en entornos de modelado y simulación como OpenModelica, Dymola o Wolfram System Modeler. Para ello es necesario que el código generado sea funcional y que se permita guardar en ficheros Modelica.
- **Capacidad de guardado y recuperación:** La herramienta permitirá a los usuarios guardar y recuperar los modelos previamente creados, brindando flexibilidad y posibilidades de realizar ediciones futuras de manera conveniente.

Para cumplir con estos subobjetivos, se realizará un análisis de los requisitos de la aplicación, se descompondrá en capas utilizando el patrón Modelo-Vista-Controlador para reducir la complejidad, se planificarán tareas siguiendo una metodología iterativa incremental basada en análisis, diseño, implementación y pruebas, y se seleccionarán tecnologías modernas como el lenguaje Java y la biblioteca JavaFX para el desarrollo. Así mismo se comprobará la funcionalidad del código Modelica generado mediante ejemplos de la propia librería **Fluid** que serán simulados y comparados en OpenModelica y Wolfram System Modeler.

FluidEditor permite a los usuarios concentrarse en la representación conceptual y estructural del sistema, liberándolos de la necesidad de preocuparse por la sintaxis de Modelica o la programación. Los usuarios tienen la capacidad de seleccionar componentes desde un árbol de componentes y colocarlos en el área de diseño, lo que facilita la creación de modelos tanto simples como complejos. Cada componente se representa mediante un gráfico o icono, lo que mejora la identificación visual en comparación con los enfoques tradicionales que utilizan cajas y etiquetas.

En resumen, **FluidEditor** es una herramienta o aplicación con interfaz gráfica desarrollada en Java que simplifica la creación de modelos hidráulicos sin la necesidad de un compilador Modelica. Esta aplicación ayuda a usuarios, alumnos, ingenieros y profesionales que están ingresando al ámbito de la simulación, proporcionando una interfaz intuitiva para la creación de modelos Modelica. El diseño se realiza arrastrando componentes desde un árbol de componentes hasta un lienzo de diseño, donde se establecen las conexiones entre ellos, y finalmente la aplicación genera el código Modelica que puede emplearse en diversos entornos de modelado y simulación que utilizan el lenguaje Modelica

1.3. Estructura de la memoria

La memoria de este proyecto está organizada en los siguientes capítulos:

- **Capítulo 1: Introducción.** En este capítulo se proporciona una visión general del proyecto, incluyendo sus objetivos y la estructura de la memoria.

- **Capítulo 2: Marco teórico.** Este capítulo ofrece un repaso de los conceptos fundamentales de modelado y simulación, así como una introducción al lenguaje Modelica. El propósito es establecer el contexto necesario para comprender el desarrollo del trabajo. Además, se detallan las tecnologías, metodologías, patrones y lenguajes de programación considerados durante la creación de la aplicación FluidEditor v0.1.

- **Capítulo 3: Análisis y planificación.** Este capítulo se enfoca en las etapas iniciales de la creación de la aplicación. Se identifican los requisitos, se realiza un análisis detallado y se elabora un plan para llevar a cabo el desarrollo en un marco de tiempo específico.

- **Capítulo 4: Arquitectura de la aplicación.** En este capítulo se describe de manera global cómo se ha dividido la aplicación para reducir su complejidad, trabajando con bloques independientes y manejables. Se destaca el uso del patrón de diseño Modelo-Vista-Controlador (MVC).

- **Capítulo 5: Implementación.** Este capítulo proporciona una descripción detallada del proceso de implementación de la aplicación, siguiendo la estructura definida en la arquitectura del patrón de diseño MVC. Se abordan los aspectos relacionados con la arquitectura y los requisitos establecidos.

- **Capítulo 6: Pruebas.** En este capítulo se lleva a cabo un conjunto de pruebas destinadas a verificar el correcto funcionamiento de la aplicación. Se hace especial hincapié en la validación del código generado para los modelos propuestos. Para ello, se crean ejemplos de modelos de la biblioteca Fluid y se verifica su operabilidad mediante simulaciones en otros entornos de modelado y simulación, como OpenModelica y Wolfram System Modeler.

- **Capítulo 7: Conclusiones y trabajo futuro.** En este capítulo se presentan las conclusiones extraídas del desarrollo de la aplicación. Se enumeran también posibles mejoras para futuras iteraciones y se proponen líneas de trabajo futuro que podrían enriquecer la funcionalidad de la aplicación.

- **Bibliografía, Anexos.** Por último, se añade la bibliografía correspondiente, así como los anexos en el que muestra un manual de usuario y el código fuente de la aplicación.

En los siguientes capítulos, se describirá en detalle la metodología utilizada en el desarrollo de la aplicación, incluyendo la arquitectura y las tecnologías empleadas. Además, se presentará el proceso de diseño de la interfaz gráfica y la lógica subyacente para la generación de código Modelica. A través de ejemplos de prueba, se demostrará la eficacia y la utilidad de la herramienta en la creación de modelos y su posterior generación de código Modelica útil y funcional. Finalmente, se discutirán las conclusiones obtenidas y se señalarán posibles direcciones futuras para mejorar y expandir esta aplicación.

2 MARCO TEÓRICO

2.1. Introducción

En este capítulo nos centraremos en la revisión teórica de algunos conceptos fundamentales de modelado y simulación, de las herramientas disponibles en el mercado para la simulación y finalmente, exploraremos algunas tecnologías que nos permitan desarrollar la herramienta de edición de modelos hidráulicos y así cumplir con los objetivos planteados en el capitulo anterior, sobre todo nos centraremos en dar un descripción de las tecnologías seleccionadas para el desarrollo de este proyecto.

2.2. Modelado de Sistemas

El avance de la ciencia y la tecnología que podemos presenciar hoy en día, se ha desarrollado gracias a la constante curiosidad del ser humano por descubrir la verdad y así dar respuesta a sus inquietudes, tanto propias como las que suceden alrededor del él (lo que se conoce como entorno). En este camino de búsqueda insaciable, el ser humano se han encontrado con varios descubrimientos, los mismos que han sido los propulsores de nuevos desarrollos, conforme avanzan estos descubrimientos, surgen nuevas necesidades, necesidades como: dar una descripción o explicación formal de las leyes que describen el comportamiento de dicho descubrimiento, ahí es donde entran en juego ramas de la ciencia como: la física, la química, que junto a la matemática intentan analizar, experimentar y extraer información para describir de manera simplificada estas leyes. La extracción de información requiere de un planteamiento correcto de lo que se quiere analizar, describir, explicar o simular, esto ha dado origen a lo que se conoce como sistema, que consiste en una representación abstracta, conceptual, gráfica, física de fenómenos, sistemas o procesos. Un sistema es un objeto o colección de objetos cuyas propiedades queremos estudiar [Fritzson, 2011].

Definir lo que constituye un sistema es algo subjetivo y su planteamiento debe guiarse por el uso que se le dará a este. Las razones por las que se requiere estudiar un sistema pueden estar motivadas por intentar explorar su comportamiento, comprender más sobre su naturaleza con el fin de extraer información, incluso experimentar antes de su construcción. Frecuentemente la experimentación directa sobre un sistema no esta disponible, ya sea por motivos económicos, bioéticas, riesgos de la vida, complejidad, etc. Por lo que en la mayoría de casos se recurre a modelos formales, matemáticos que expresen los comportamientos de su estructura, de su física, de su química, de su dinámica mediante ecuaciones lógico-matemáticas que describen la evolución del sistema a lo largo del tiempo de simulación (periodo de observación). Esta forma de expresar el sistema se conoce como modelo. Un modelo tiene varias definiciones en la literatura del modelado. Se define como: Un modelo de un sistema es cualquier cosa a la que se puede aplicar un "experimento"para responder preguntas sobre ese sistema [Fritzson, 2011].

Un modelo matemático esta descrito por expresiones lógico-matemáticas que nos brindan una alta y sofisticada ventaja de cara a la experimentación, nos permiten ensayar condiciones que serian difíciles de llevar a cabo en un sistema real, tales como: estudiar la evolución temporal de largos periodos de tiempo sin tener que esperar el periodo real, el tiempo se limita a la velocidad de procesamiento de la máquina de calculo; variar parámetros en rangos extremos que serian inviables en un caso real, etc. En otras palabras estos modelos nos ofrece un una gran flexibilidad para realizar diferentes tipo de variaciones que nos permitan observar el comportamiento y sacar las conclusiones pertinentes sin tener que esperar un largo periodo de tiempo, en comparación con el tiempo que se necesitaría si lo experimentáramos en un caso real, entre otras ventajas [Urquía and Martín, 2016].

En la Figura 2.1 se puede observar un resumen de las distintas formas que se puede estudiar un sistema. De esta clasificación nos centraremos en la experimentación con el modelo del sistema. El mismo que se clasifica en: Modelo Mental, Modelo Verbal, Modelo Físico y Modelo Matemático. Lo que nos atañe en nuestro caso es el modelo matemático en especifico en la parte de simulación.

Para llevar a cabo un modelo se necesita plantear un serie de hipótesis, en dicha hipótesis se debe tener en cuenta siempre las características más importantes que definan la realidad de su representación (o las variables de interés en el estudio del caso) y a la vez se desea que sea lo bastante sencillo para entenderlo, manipularlo y extraer conclusiones de lo que estamos interesados observar. Esto se consigue eliminando los detalles que no aportan al objetivo del modelo, y tratando de identificar las características más importantes. Por ello, es de gran interés tener un conocimiento previo de lo que pretendemos modelar.

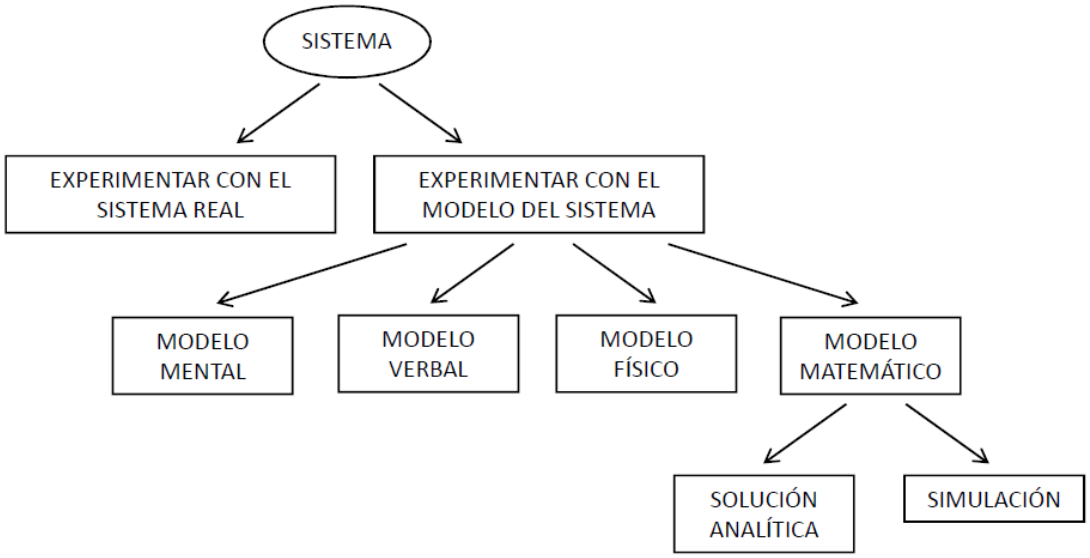

Figura 2.1: Formas de estudiar un sistema [Urquía and Martín, 2016]

2.2.1. Tipos de modelos matemáticos

Los modelos matemáticos pueden clasificarse atendiendo a diferentes criterios, entre los criterios más utilizados por la literatura, se encuentran los siguientes, aquellos modelos en el que la variable del tiempo juegan un papel importante, y aquellos modelos en el que no se toma en cuenta el tiempo o dicha variable no es importante en su análisis, dando como resultado a dos clasificaciones:

- **Modelos estáticos**: No tienen dependencia con el tiempo.
- **Modelos dinámicos**: Tienen una fuerte dependencia con la variable tiempo, es decir, sus variables de estudio están en función de la evolución temporal.

Otra clasificación muy típica es en aquellos modelos que atienden a la existencia de variables aleatorias o la no existencia de las mismas, dando como resultado la clasificación siguiente:

- **Modelos deterministas**: Aquellos modelos que no contienen variables aleatorias.
- **Modelos estocásticos**: Aquellos modelos en el que alguna o todas sus variables son aleatorias.

Las dos clasificaciones descritas previamente, se pueden combinar y dar como resultado las cuatro combinaciones.

- **Modelos estáticos deterministas**.
- **Modelos estáticos estocásticos**.
- **Modelos dinámicos deterministas**.
- **Modelos dinámicos estocásticos**.

Los modelos dinámicos pueden clasificarse atendiendo al instante de tiempo en el que el valor de sus variables pueden cambiar, y se clasifican en:

- **Modelos de tiempo discreto**: El valor de las variables sólo puede cambiar en instantes específicos, permaneciendo constante el resto del tiempo, los eventos se producen en instantes de tiempo predefinidos y a intervalos constantes de tiempo.
- **Modelos de eventos discretos**: Los eventos no se producen en un tiempo predefinido, sino que pueden ocurrir en cualquier instante, es decir, no contienen un intervalo constante o regular.
- **Modelos de tiempo continuo**: Se caracterizan por el hecho de que el valor de sus variables puede cambiar de manera continua a lo largo del tiempo. A este tipo de variables se las denomina variables de tiempo continuo.
- **Modelos híbridos**: Son aquellos modelos que tienen una parte de tiempo continuo, y una parte de tiempo discreto o eventos discretos.

Los modelos de tiempo continuo pueden a su vez clasificarse atendiendo a si contienen o no derivadas respecto a las coordenadas espaciales y se clasifican en:

- **Modelos de parámetros concentrados**: Están descritos mediante ecuaciones algebraicas y ecuaciones diferenciales ordinarias (ODE, del ingles Ordinary Differential Equations) en las cuales la derivada es únicamente respecto al tiempo.
- **Modelos de parámetros distribuidos**: En estos modelos existen ecuaciones en las que tienen derivadas respecto a las coordenadas espaciales, con la posibilidad que aparezcan derivadas respecto al tiempo.

Los modelos de parámetros concentrados a su vez admiten una clasificación, dicha clasificación esta determinada en función de los tipos de ecuaciones que intervienen en su definición.

- **Modelos algebraicos**: Modelos compuestos únicamente por ecuaciones algebraicas.
- **Modelos dinámicos en ecuaciones diferenciales ordinarias (ODE)**: Conocidos también como modelos dinámicos ODE. Son modelos compuesto por ecuaciones diferénciales ordinarias, es decir, la ecuación diferencial admite únicamente derivada respecto del tiempo. Los modelos ODE se clasifican a su vez en ODE explícito y ODE implícitos, dependiendo de si es posible o no despejar a un lado de la igualdad las derivadas respecto al tiempo.
- **Modelos de ecuaciones algebraico-diferenciales (DAE)**: Conocidos como modelos DAE (del inglés, Differential-Algebraic Equations), en estos modelos intervienen ecuaciones algebraicas y ecuaciones diferenciales ordinarias con derivadas únicamente respecto al tiempo. Los modelos DAE pueden ser DAE semi-explícito, cuando es posible despejar las derivadas, o bien DAE implícito, cuando no es posible despejar estas. En los modelos DAE no aparecen derivadas respecto a las coordenadas espaciales.

En la Figura 2.2, se resume la clasificación de los modelos matemáticos comentados anteriormente. En el desarrollo de este trabajo los atañe los **modelos de tiempo continuo**.

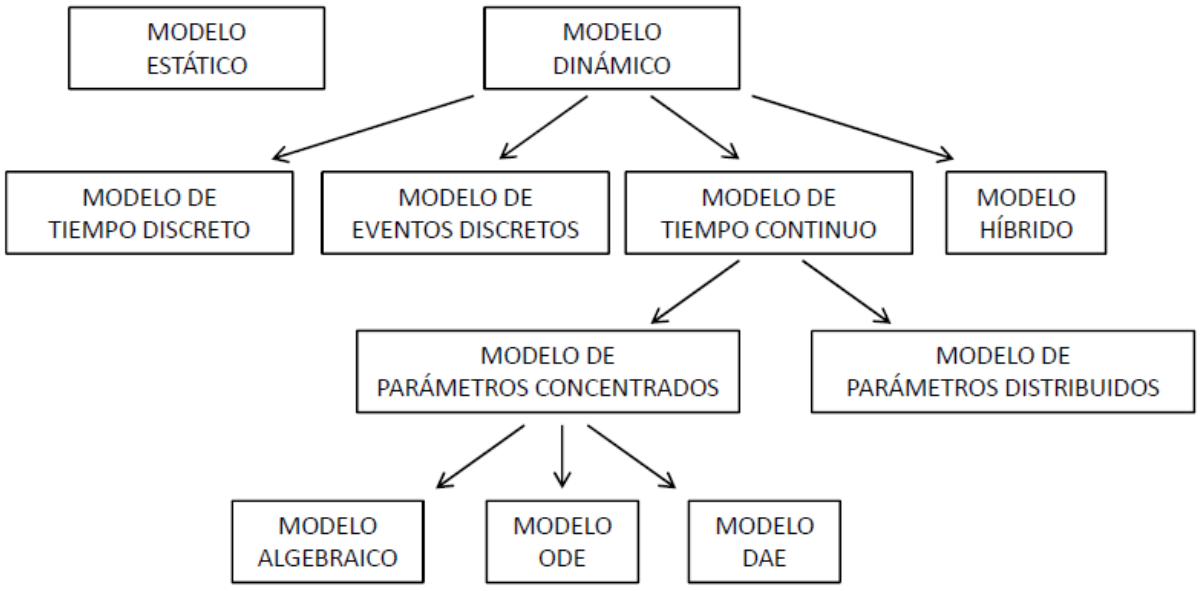

Figura 2.2: Clasificación de los modelos matemáticos [Urquía and Martín, 2016]

2.3. Modelado y simulación de tiempo continuo

En esta sección, nos enfocaremos en revisar y comprender en qué consiste la simulación en tiempo continuo. Para lograrlo, haremos un breve recorrido histórico con el objetivo de contextualizar y tener una visión más clara de esta.

El modelado y la simulación tienen sus raíces en la década de 1920, pero su evolución desde entonces ha sido sorprendente e inimaginable. En sus inicios, estas técnicas se limitaban al ámbito de los laboratorios y centros de investigación. Hoy en día, se han democratizado y están al alcance de cualquier estudiante, ingeniero o incluso de usuarios curiosos que deseen explorar este campo. La tecnología ha desempeñado un papel fundamental en este desarrollo [Elmqvist et al., 1998].

Inicialmente, se utilizaron técnicas analógicas, predominantes entre 1920 y 1950. Sin embargo, un hito importante se produjo con la llegada de las computadoras, marcando un cambio de paradigma de la simulación analógica a la simulación digital. Este cambio abrió la puerta al desarrollo de herramientas de software que continúan evolucionando día a día [Elmqvist et al., 1998].

Hoy en día, gracias a la capacidad de procesamiento gráfico de las computadoras, estas herramientas permiten pasar de la compleja definición de ecuaciones en texto para la construcción de modelos a una definición más simple e intuitiva mediante representaciones gráficas. Esto se logra a través de diagramas de bloques u objetos que representan entidades. En cuanto a los resultados, también han experimentado una transformación significativa. Hemos pasado de extensas tablas de números difíciles de interpretar y extraer información, a gráficos dinámicos que permiten una visualización instantánea y una comprensión más clara de la información.

En sus inicios, las técnicas analógicas se fundamentaban en ecuaciones diferenciales ordinarias (ODE, por sus siglas en inglés, Ordinary Differential Equation), que se describen matemáticamente como se muestra en la Ecuación (2.1). La idea principal consistía en definir un modelo en términos de ODEs y luego desarrollar un dispositivo físico que obedeciera estas ecuaciones. El sistema se inicializaba con valores iniciales apropiados, conocidos como condiciones iniciales, y se observaba cómo evolucionaba a lo largo del tiempo, lo que se denomina tiempo de simulación [Elmqvist et al., 1998].

$$\frac{dx}{dt} = f(t, x) \tag{2.1}$$

En la simulación analógica, una Ecuación Diferencial Ordinaria (ODE), como la que se muestra en la Ecuación (2.1), debía expresarse en términos de operaciones fundamentales como integración, adición y multiplicación. Existían diversos métodos para obtener soluciones numéricas aproximadas para una ODE. Estos métodos implicaban la sustitución de las ODE por ecuaciones algebraicas y su resolución mediante métodos numéricos. Por ejemplo, el método de Euler se basa en la aproximación de la primera derivada mediante una ecuación de diferencias. También existen técnicas más eficientes, como los métodos de la familia Runge-Kutta de paso fijo y métodos de paso múltiple, entre otros. Un avance significativo en la resolución de problemas mediante métodos numéricos fue la introducción de la adaptación del paso de integración, gracias a las contribuciones de Fehlberg [Fehlberg, 1969].

El constante avance de los métodos numéricos, el aumento de la capacidad de cómputo y el desarrollo de lenguajes de programación permitieron el surgimiento del paradigma de modelado físico. Este paradigma destaca por su capacidad para analizar sistemas altamente complejos, identificables por la presencia de restricciones impuestas por las variables del sistema y la existencia de múltiples ecuaciones acopladas. El enfoque típico del modelado físico implica descomponer el sistema en subsistemas, teniendo en cuenta las interfaces de cada uno de ellos para aplicar balances de masa, energía y momentos. El modelo del sistema completo resulta de la combinación de la información de todos los subsistemas.

La aparición de los lenguajes de modelado orientados a objetos en la década de 1990 dio lugar al desarrollo de herramientas de software para el modelado y la simulación. Estas herramientas facilitaron la descripción de modelos híbridos de sistemas físicos, donde fenómenos de diferentes dominios (eléctrico, mecánico, hidráulico, térmico, etc.) se interrelacionan. Estos lenguajes de modelado son de propósito general y no están limitados a un dominio específico.

Sin embargo, la existencia de numerosos lenguajes de modelado llevó a una dispersión en el esfuerzo de desarrollo de herramientas y bibliotecas. Estas se desarrollaban en diferentes lugares y, en algunos casos, no se podían reutilizar debido a estar fuertemente acopladas a problemas específicos o escritas en diferentes lenguajes de programación. Para abordar esta fragmentación y proponer un lenguaje de modelado estándar que permitiera la interoperabilidad de modelos y la compatibilidad entre entornos de modelado, se creó un grupo de diseño en 1996. Este grupo estaba formado por personas con experiencia en el desarrollo de lenguajes de modelado y en la aplicación de modelos en el ámbito académico e industrial. El lenguaje resultante, llamado Modelica, incorporó características de lenguajes de modelado previamente existentes, como ALLAN, Dymola, NMF, ObjectMath, Omola, SIDOPS+ y Smile. Desde 1997, se han publicado diversas versiones de Modelica [Urquía and Martín, 2016].

Un hito importante fue la fundación de la Modelica Association en 2000, que publica las especificaciones del lenguaje, así como artículos científicos, manuales y bibliotecas de modelos, y proporciona enlaces para la descarga de herramientas de modelado y simulación. Entre las bibliotecas de modelos gratuitas más destacadas se encuentra la Modelica Standard Library (MSL), desarrollada y mantenida por la Modelica Association. Modelica se ha convertido en un lenguaje ampliamente utilizado en los ámbitos académico e industrial, con numerosas bibliotecas gratuitas y comerciales disponibles para su uso. Información extraída de [Urquía and Martín, 2016].

En cuanto a los entornos y lenguajes desarrollados para el modelado de sistemas, en general, pueden clasificarse en dos tipos principales [Urquía and Martín, 2016]:

- Entornos de Simulación: Estos entornos facilitan el modelado basado en diagramas de bloques, lo que permite una descripción modular y jerárquica de la representación matemática del modelo. En este paradigma, el desarrollador del modelo debe manipular y expresar explícitamente qué variable debe evaluarse en cada una de las ecuaciones del modelo. Dos ejemplos de entornos de simulación para modelos híbridos son Matlab/Simulink y Scilab/Scicos. El más conocido es Simulink (originalmente llamado SIMULAB), que se integra con Matlab. Este lenguaje surgió en 1991 y está diseñado especialmente para trabajar con diagramas de bloques, utilizando MATLAB para el análisis dinámico del sistema [Grace, 1991].

- Lenguajes de Modelado Orientado a Objetos: Estos lenguajes simplifican la descripción del modelo al permitir que el usuario escriba directamente las ecuaciones del mismo. Las herramientas de software que soportan estos lenguajes, denominadas entornos de modelado, se encargan de determinar qué variables se evalúan en cada ecuación, además de organizar y manipular el modelo de manera que pueda resolverse numéricamente. La ventaja principal de estos lenguajes es que liberan al desarrollador del modelo de tareas adicionales, lo que agiliza significativamente el proceso de desarrollo y modificación de los modelos, así como su reutilización. Dos ejemplos de lenguajes de modelado orientado a objetos son Modelica y EcosimPro.

2.4. El lenguaje Modelica

El lenguaje Modelica es un lenguaje de modelado orientado a objetos que facilita el paradigma del modelado físico. Soporta una descripción no causal del modelo basada en ecuaciones que facilita su reutilización. Modelica está concebido para describir modelos compuestos por ecuaciones algebraico diferenciales y eventos.

Modelica es gratuita, esta diseñada y respaldada por la Asociación Modelica. Es idónea para sistemas que abarcan diversos dominios, como modelos utilizados en aplicaciones de robótica, automoción y aeroespaciales que incluyen subsistemas eléctricos e hidráulicos, además de la producción y distribución de energía eléctrica [Modelica Association, 2001].

2.4.1. La Asociación Modelica

La Asociación Modelica es una organización no gubernamental sin fines de lucro con miembros en Europa, Estados Unidos, Canadá y Asia. Su misión principal es desarrollar y promover el lenguaje de modelado Modelica para su aplicación en la modelización, simulación y programación de sistemas y procesos físicos y técnicos. La Asociación Modelica es la propietaria y administradora de los derechos intelectuales asociados con Modelica, que incluyen marcas comerciales, la especificación del lenguaje Modelica, bibliotecas estándar de Modelica, entre otros [Modelica Association, 2023b].

Los recursos están disponibles de manera generalizada para fomentar el desarrollo industrial y la investigación en este campo. Desde 1996, la Asociación Modelica ha liderado la creación de estándares coordinados de acceso abierto y ha promovido el desarrollo de software de código abierto en el ámbito de los sistemas ciber-físicos. Los estándares actuales de la Asociación Modelica incluyen [Modelica Association, 2023b]:

- Lenguage Modelica.
- Interfaz de simulación funcional (FMI, del inglés Functional Mock-up Interface)
- Estructura y parametrización del sitema (SSP, del inglés System Structure and Parameterization)
- Protocolo de co-simulación distribuida (DCP, del inglés Distributed Co-Simulation Protocol)
- Interfaz de maqueta funcional para sistemas integrados (eFMI, del inglés Functional Mock-up Interface for embedded Systems)

Cada uno de estos estándares se encuentra respaldado por software de código abierto que facilita su implementación y uso, como la biblioteca estándar de Modelica que contiene aproximadamente 1600 componentes de modelos Modelica en diversos dominios. También existen otras bibliotecas de Modelica de código abierto, como el verificador de cumplimiento de FMI, que se utiliza para verificar si un modelo cumple con los requisitos del estándar FMI [Modelica Association, 2001]. Más información en la Web de la Asociación Modelica [Modelica Association, 2023b].

2.4.2. La Librería Estándar Modelica (MSL)

La Librería Estándar Modelica (MSL, por sus siglas en ingles Modelica Standard Library), también conocida como Biblioteca Estándar Modelica, es una biblioteca gratuita, incluye una amplia gama de componentes y modelos predefinidos de diversos campos de la ingeniería. Permite la modelización de máquinas mecánicas (1D/3D), sistemas eléctricos (analógicos, digitales, máquinas), sistemas magnéticos, térmicos, fluidos, sistemas de control y sistemas jerárquicos. También incorpora funciones numéricas y funciones para manipular cadenas de texto, archivos y secuencias [Modelica Association, 2001].

La librería estándar se distribuye en entornos de modelado tales como OpenModelica [OpenModelica, 2023], Dymola [Dassault-Systèmes, 2023], etc. y puede descargarse gratuitamente de la página web de la Asociación Modelica [Modelica Association, 2023b]. La librería estándar se presenta como un paquete llamado **Modelica**, el cual, a su vez, contiene sub-paquetes como **Blocks**, **Constants**, **Electrical**, entre otros. En la Figura 2.3, se muestra el árbol de paquetes que conforman la Librería Estándar de Modelica. Se ha destacado y señalado con una flecha el paquete que se utilizará en el desarrollo de la aplicación de este proyecto, como se describirá más adelante en esta memoria.

Para utilizar la Librería Estándar Modelica, se requiere un entorno de modelado y simulación. En el mercado existen opciones tanto comerciales como gratuitas de estos entornos, en el apartado siguiente se hablará de algunos de ellos. A lo largo de los años, se han lanzado varias versiones de la Librería Estándar Modelica, siendo la última la versión 4.0 lanzada en 2020, la cual se utilizará en este proyecto.

2.4.3. La librería Fluid

La librería **Fluid** es un paquete gratuito de Modelica que proporciona componentes para el diseño de sistemas termo-fluidos unidimensionales. Incluye recipientes, tuberías, máquinas de fluidos, válvulas y accesorios. Una característica única de esta librería es que las ecuaciones de los componentes, los modelos de medios, así como las correlaciones de pérdida de presión y transferencia de calor, están desacoplados entre sí. Todos los componentes están implementados de manera que puedan usar los medios disponibles en la biblioteca **Modelica.Media**. Esto significa que se pueden utilizar medios incompresibles y comprimibles, así como medios de una o varias sustancias con una o más fases.

En la Figura 2.4 se muestran los diferentes paquetes que constituyen la librería **Fluid**. Mientras que en la Tabla 2.1 se muestra la descripción para cada uno de estos componentes.

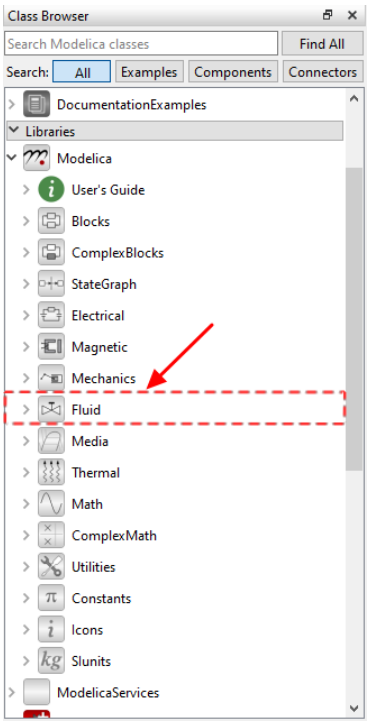

Figura 2.3: Paquetes que conforman la Librería Estándar Modelica (MSL).

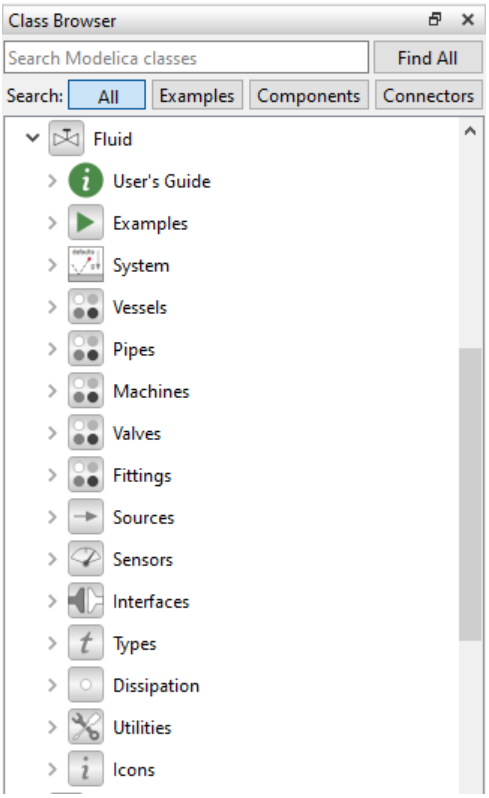

Figura 2.4: Principales paquetes que conforman la librería Fluid.

Nombre	Descripción
User's Guide	Guía de usuario.
Examples	Demostración del uso de la librería.
System	Propiedades del sistema y valores predeterminados (ambiente, dirección del flujo, inicialización)
Vessels	Dispositivos para almacenar fluido.
Pipes	Dispositivos para transportar fluido.
Machines	Dispositivos para convertir entre energía contenida en un fluido y energía mecánica.
Valves	Componentes para la regulación y control del flujo de fluidos.
Fittings	Adaptadores para conexiones de componentes fluidos y regulación del flujo de fluidos.
Sources	Definir condiciones de contorno fijas o prescritas.
Sensors	Sensores ideales para extraer señales de un conector de fluido.
Interfaces	Interfaces para flujo en estado estacionario y no estacionario, de fase mixta, de múltiples sustancias, incompresible y compresible.
Types	Tipos comunes de modelos de fluidos.
Dissipation	Funciones para la transferencia de calor por convección y las características de pérdida de presión.
Utilities	Modelos de utilidad para construir componentes fluidos (no deben usarse directamente)

Tabla 2.1: Descripciones de los componentes que forman parte del paquete Fluid.

2.4.4. Las clases Modelica

Modelica es un lenguaje orientado a objetos y, por lo tanto, comparte características comunes con los lenguajes de programación orientados a objetos, como el encapsulamiento, la herencia, el polimorfismo, entre otros. En la práctica, esto implica que es posible definir clases y luego crear diferentes instancias de la misma clase, cada una con propiedades específicas (atributos). También significa que es factible crear una jerarquía de clases.

Un aspecto distintivo de este lenguaje es la definición de clases especializadas que contienen especificaciones adicionales, que facilitan el modelado de componentes específicos. La definición de modelos, librerías, magnitudes físicas, conjuntos de datos y funciones se lleva a cabo mediante las siete clases presentadas en la Tabla 2.2, que conforman el núcleo esencial del lenguaje Modelica.

Nombre	Aplicación
`type`	Permite extender los tipos de variables predefinidos en el lenguaje.
`connector`	Define conectores (esto es, grupos de variables de la interfaz de los componentes) con el fin de facilitar la descripción de la conexión entre componentes. No puede contener ecuaciones.
`model`	Se emplea para definir las clases de modelos.
`block`	Es igual que la clase model, pero en la clase block las variables de la interfaz deben tener la causalidad computacional explícitamente definida.
`record`	Permite definir conjuntos de variables y de parámetros. No puede tener ecuaciones. Su finalidad es facilitar la parametrización de los modelos.
`function`	Permite la definición de funciones, es decir, el encapsulado de código algorítmico que puede ser reutilizado en la definición del modelo y también del experimento.
`package`	Facilita la definición de librerías de clases. Un package es una clase que únicamente puede contener otras clases.

Tabla 2.2: Clases del lenguaje Modelica. Extraída de [Urquía and Martín, 2016].

2.4.5. Las anotaciones Modelica

Una anotación en Modelica es información adicional que se asocia a un modelo Modelica. Esta información adicional es utilizada por los entornos de simulación para, por ejemplo, respaldar la documentación del modelo o facilitar su edición y representación gráfica. La mayoría de las anotaciones no afectan a la ejecución de la simulación, es decir, se obtiene el mismo resultado si se elimina la anotación, aunque existen excepciones a esta regla. La sintaxis de una anotación es la siguiente:

```
annotation(elementos_de_anotacion)
```

donde `elementos_de_anotacion` es una lista de elementos de anotación, separados por comas, los cuales pueden ser cualquier tipo de expresión compatible con la sintaxis de Modelica [Modelica Association, 2023a].

La capacidad de Modelica para admitir anotaciones en los modelos desempeña un papel fundamental en el desarrollo de este proyecto. Estas anotaciones son esenciales para definir la representación gráfica del icono y el diagrama de cada modelo. Esto se debe a que a partir de estas anotaciones, que están presentes en los archivos Modelica, se extraerá la información gráfica de cada uno de los componentes Modelica que forman parte de la librería Fluid. Esta información se utilizará para generar una representación visual de estos componentes en una paleta desde la cual podrán ser arrastrados al área de diseño de la aplicación para componer nuevos modelos de manera gráfica.

Anotaciones para objetos gráficos

La representación gráfica de una clase se compone de dos niveles de abstracción: la capa de iconos y la capa de diagrama, en las cuales se presentan objetos gráficos, iconos de componentes, conectores y líneas de conexión. La representación del icono, en general, visualiza el componente ocultando los detalles jerárquicos. Mientras que la descomposición jerárquica se describe en la capa de diagrama, mostrando iconos de sub-componentes y las conexiones entre ellos.

Los gráficos se especifican como una secuencia ordenada de primitivas gráficas que componen el icono, estas primitivas gráficas son las siguientes:

- **Line**
- **Rectangle**
- **Text**
- **Polygon**
- **Ellipse**
- **Bitmap**

En el siguiente ejemplo se presenta una anotación que incorpora las primitivas mencionadas previamente. La correspondencia entre el código Modelica y su representación gráfica se puede apreciar en la Figura 2.5. Cada una de estas primitivas se identifica por su nombre específico, como Rectangle, Ellipse, Line, Text, y entre paréntesis se detallan las propiedades particulares de cada primitiva, como su origen (origin), dimensiones (extent), color (color), entre otras. Estas primitivas están encapsuladas dentro de la palabra clave `Icon(conjunto_primitivas)`.

```
1    annotation(
2        Icon( graphics = {
3                    Rectangle(origin = {-43, 56}, lineColor = {255, 170,
                          127}, fillColor = {170, 0, 255}, pattern =
                         LinePattern.Dash, fillPattern = FillPattern.
                         Cross, lineThickness = 1, extent = {{-25, 20},
                         {25,-20}}),
4                    Ellipse(origin = {45, 54}, lineColor = {0, 255,
                         255}, fillColor = {170, 0, 127}, fillPattern =
                         FillPattern.HorizontalCylinder, extent = {{-29,
                         24}, {29, -24}}), Polygon(origin = {-43, -44},
                         lineColor = {0, 255, 127}, fillColor = {170,
                         170, 255}, fillPattern = FillPattern.CrossDiag,
                         points = {{-27, 32}, {17, 28}, {31, -10}, {-13,
                         -34}, {-37, -6}, {-27, 32}, {-27, 32}}),
5                    Line(origin = {51.1085, -38.3625}, points =
                         {{-28.9874, -26.7224}, {13.0126, 27.2776}},
                         color = {170, 0, 255}, pattern = LinePattern.
                         DashDot, thickness = 1.2),  Text(origin = {2,
                         8}, textColor = {255, 0, 255}, extent = {{-42,
                         0}, {42, 0}}, textString = "Texto de prueba",
                         fontSize = 16) }
6            )
7        );
```

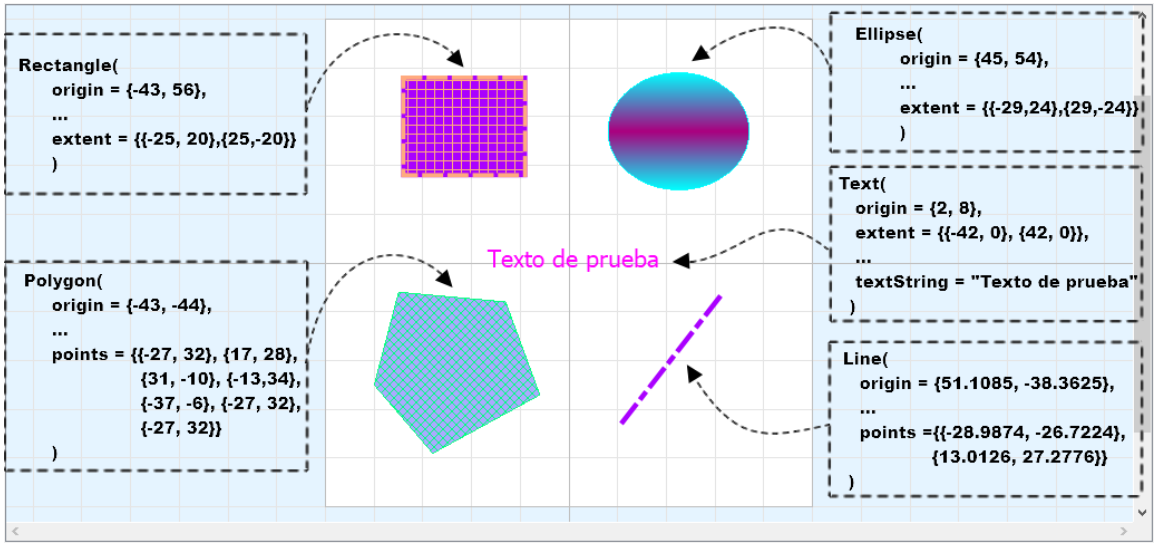

Figura 2.5: Ejemplo de correspondencia entre el código de las primitivas en las anotaciones y su representación gráfica.

Otra de las anotaciones con las que cuenta Modelica y de las que se han utilizado en este trabajo, son aquellas anotaciones que permiten la transformación de los iconos dentro del área de diseño. Estas anotaciones posibilitan la definición de la ubicación, las dimensiones, la rotación y otros aspectos estéticos tanto del icono como del diagrama de diseño. A continuación, se presenta un ejemplo de estas anotaciones, en el que se define un componente llamado **pump** al que se le ha aplicado una transformación. La transformación se define como argumento de `transformation(transformation_component)`. Para obtener información más detallada sobre este tipo de anotaciones, y de las distintas transformaciones se puede consultar la especificación del lenguaje Modelica [Modelica Association, 2023a].

```
Modelica.Fluid.Machines.PrescribedPump pump
    annotation(
        Placement(
            visible = true,
            transformation(
                origin = {-73.414, 15},
                extent = {{-10, -10}, {10, 10}},
                rotation = 90
                )
            )
        );
```

2.5. Entornos de modelado y simulación Modelica

En esta sección, exploraremos algunos de los entornos de modelado y simulación más relevantes que hacen uso del lenguaje Modelica, como Dymola, Wolfram System Modeler y OpenModelica.

2.5.1. Dymola

Dymola es un entorno de modelado y simulación para modelos descritos en el lenguaje Modelica. Su desarrollo se originó en el trabajo de Hilding Elmqvist, quien lo ideó como parte de su tesis doctoral [Hilding, 1978]. La primera versión se basó en el "Lenguaje de Modelado Dinámico"(conocido como Dymola) y posteriormente se re-implementó utilizando los lenguajes de programación Pascal y C++.

En 1992, Elmqvist fundó la empresa sueca Dynasim AB, la cual fue posteriormente adquirida por Dassault Systèmes para integrar Dymola en la plataforma CATIA.

Dymola traduce el modelo escrito en Modelica a un programa en lenguaje de programación C llamado `dsmodel.c`. Cuando se ejecuta la simulación, Dymola compila este programa, generando un archivo ejecutable llamado dymosim.exe. Para llevar a cabo este proceso, Dymola requiere un compilador de C instalado en el sistema del usuario, ya que este compilador no se distribuye junto con el software. Hay varias opciones de compiladores de C disponibles, como Microsoft Visual Studio/Visual C++ Express Edition y MinGW GCC, entre otros.

Dymola se distribuye bajo una licencia propietaria, aunque existe una versión de evaluación con funciones limitadas. En la actualidad, es uno de los entornos más ampliamente utilizados en la industria del automóvil, la aeroespacial y el equipamiento industrial para el modelado y la simulación de sistemas [Dassault-Systèmes, 2023].

En las Figuras 2.6, 2.7 y 2.8, se muestra la interfaz de Dymola en su versión 2020. En la vista de diseño, se encuentra el árbol de componentes que pueden ser arrastrados al área de diseño para crear modelos compuestos. En la sección de código (Texto), se presenta el código Modelica tanto del modelo compuesto como de cada componente individual. En la pestaña de simulación (Simulation), se pueden seleccionar o desactivar las variables y parámetros que se desean analizar de forma gráfica. Como se observará más adelante, otros entornos de simulación comparten una estructura similar en su interfaz gráfica de usuario (GUI, por sus siglas en inglés, Graphic User Interface).

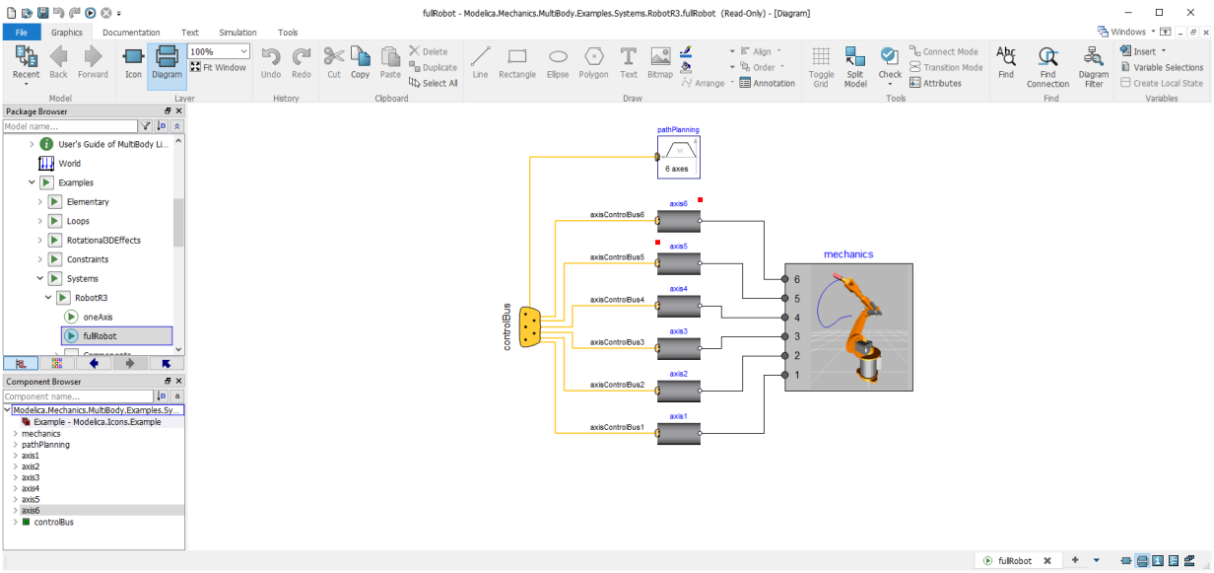

Figura 2.6: Interfaz Dymola 2020: Vista de diseño [Dassault-Systèmes, 2020]

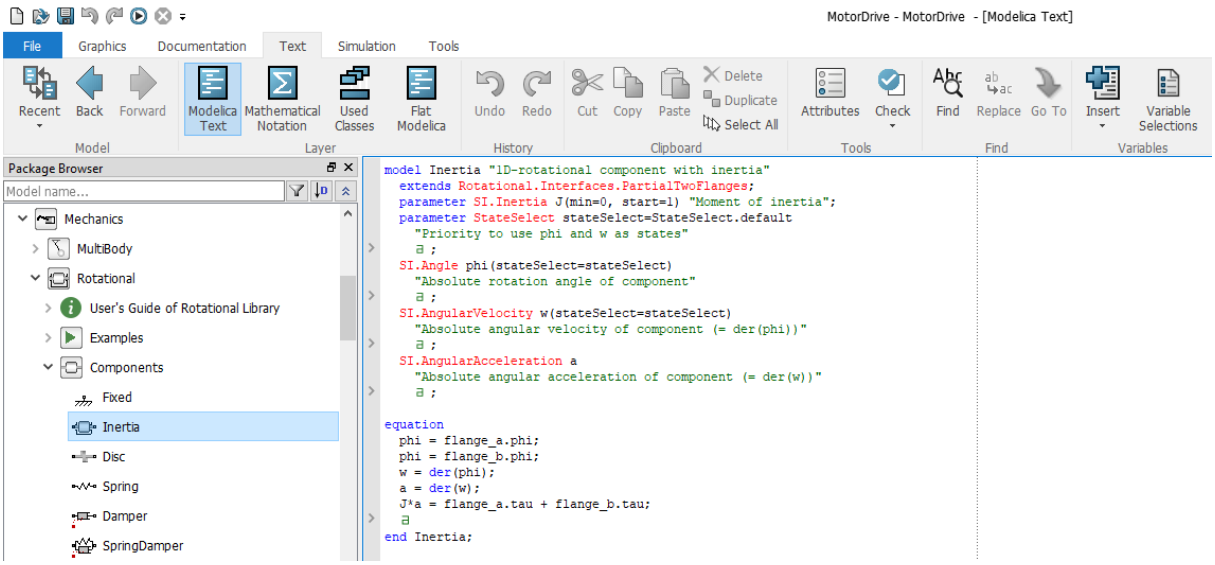

Figura 2.7: Interfaz Dymola 2020: Vista de código [Dassault-Systèmes, 2020]

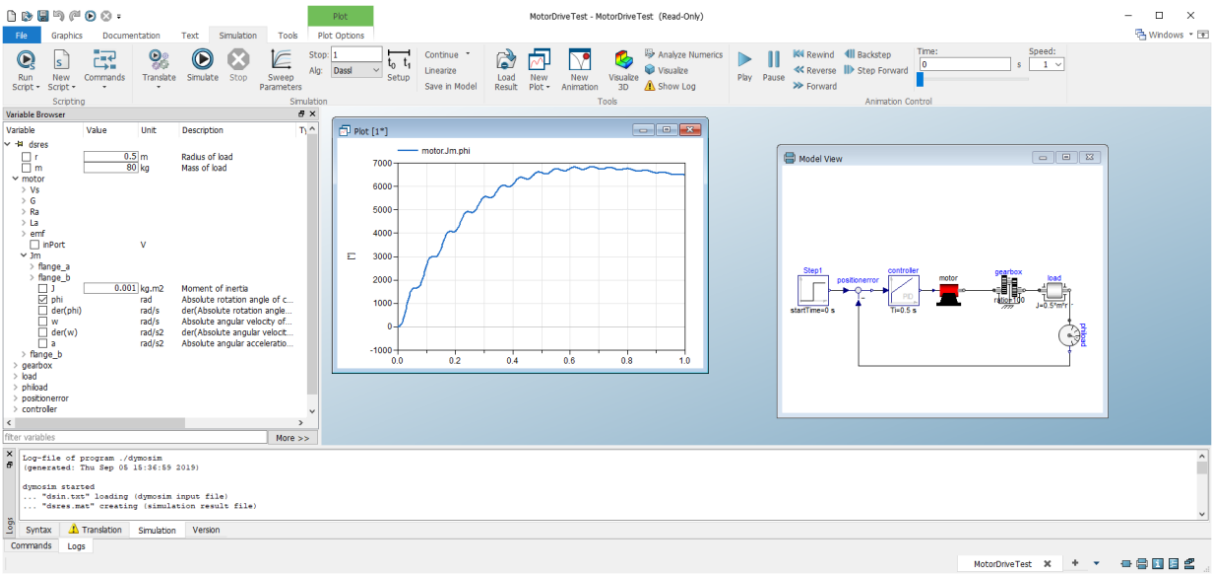

Figura 2.8: Interfaz Dymola 2020: Vista de simulación [Dassault-Systèmes, 2020]

2.5.2. Wolfram System Modeler

En su página oficial, **Wolfram System Modeler** se define como:

"Wolfram System Modeler es un entorno de simulación y modelado de última generación, fácil de usar, para sistemas ciberfísicos. Al usar la función de arrastrar y soltar desde la amplia selección de bibliotecas de modelado incorporadas y expandibles, podrá crear modelos multidominio de nivel industrial de su sistema completo. Agregar la potencia de Wolfram Language le brindará un entorno completamente integrado para analizar, comprender e iterar rápidamente diseños de sistemas, impulsando el conocimiento, la innovación y los resultados" [Wolfram, 2023].

Este entorno de simulación fue desarrollado por MathCore Engineering AB y luego adquirido por Wolfram Research, el 30 de marzo de 2011. Wolfram System Modeler proporciona una interfaz gráfica interactiva que simplifica el modelado y la simulación de sistemas de ingeniería y ciencias de la vida. Los usuarios pueden arrastrar componentes desde un panel ubicado a la izquierda hacia un área de diseño central, donde pueden realizar conexiones para construir sistemas completos. Lo que distingue a este entorno es su integración con Mathematica, lo que permite a los usuarios aprovechar la capacidad de modelado del lenguaje Modelica y la potencia computacional de Mathematica para desarrollar, simular, documentar, analizar, optimizar y realizar pre-procesamiento de manera eficiente.

Entre las características más destacables de Wolfram System Modeler incluyen:

- Interfaz intuitiva de tipo “arrastrar y soltar”.
- Utiliza en el lenguaje Modelica (lenguaje de uso libre), orientado a objetos y con descripciones de comportamiento mediante ecuaciones.
- Modelado no causal (basado en componentes) y causal (basado en bloques).
- Capacidades de modelado multidominio, que abarcan áreas como mecánica, eléctrica, hidráulica, térmica, control y biología, entre otras.
- Integración con Mathematica para el análisis y documentación de simulaciones.

Wolfram System Modeler se lanzó por primera vez en marzo de 2011 y, a la fecha de este documento, se encuentra en la versión 13.3. En las Figuras 2.9, 2.10 y 2.11 se pueden apreciar las interfaces de Wolfram System Modeler en los modos de Diseño, Código y Simulación, respectivamente. Para obtener más información, se puede visitar la documentación oficial en [Wolfram, 2023].

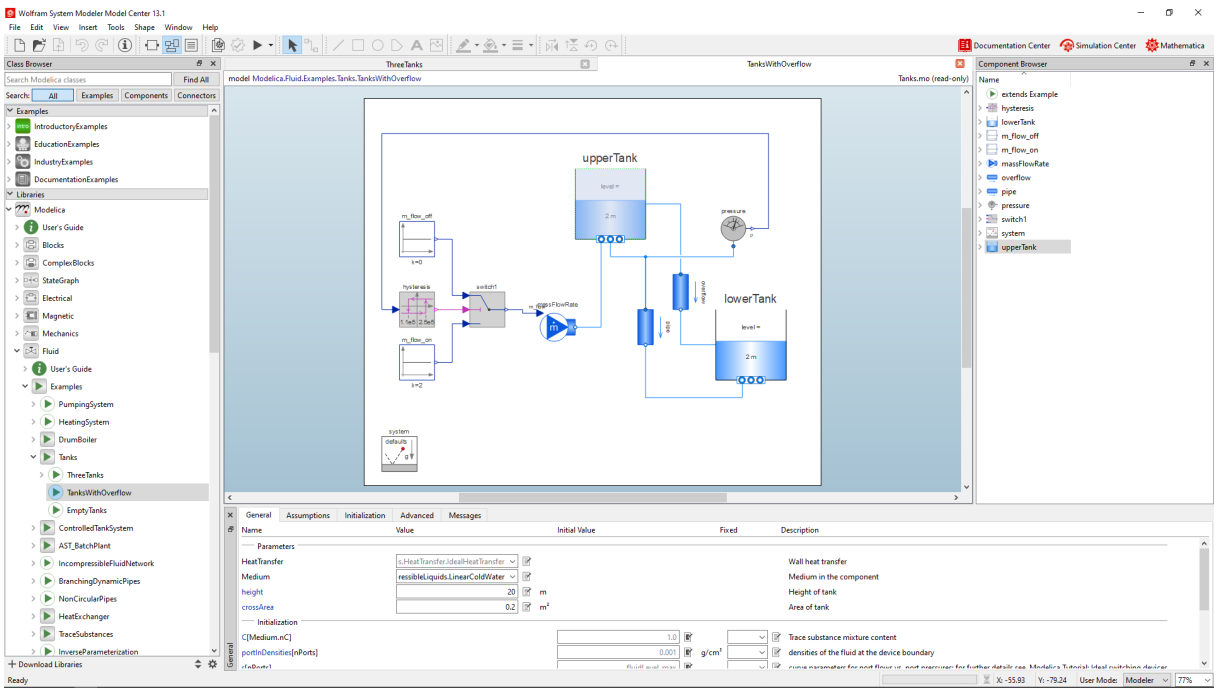

Figura 2.9: Interfaz Wolfram System Modeler 13.1: Vista de diseño

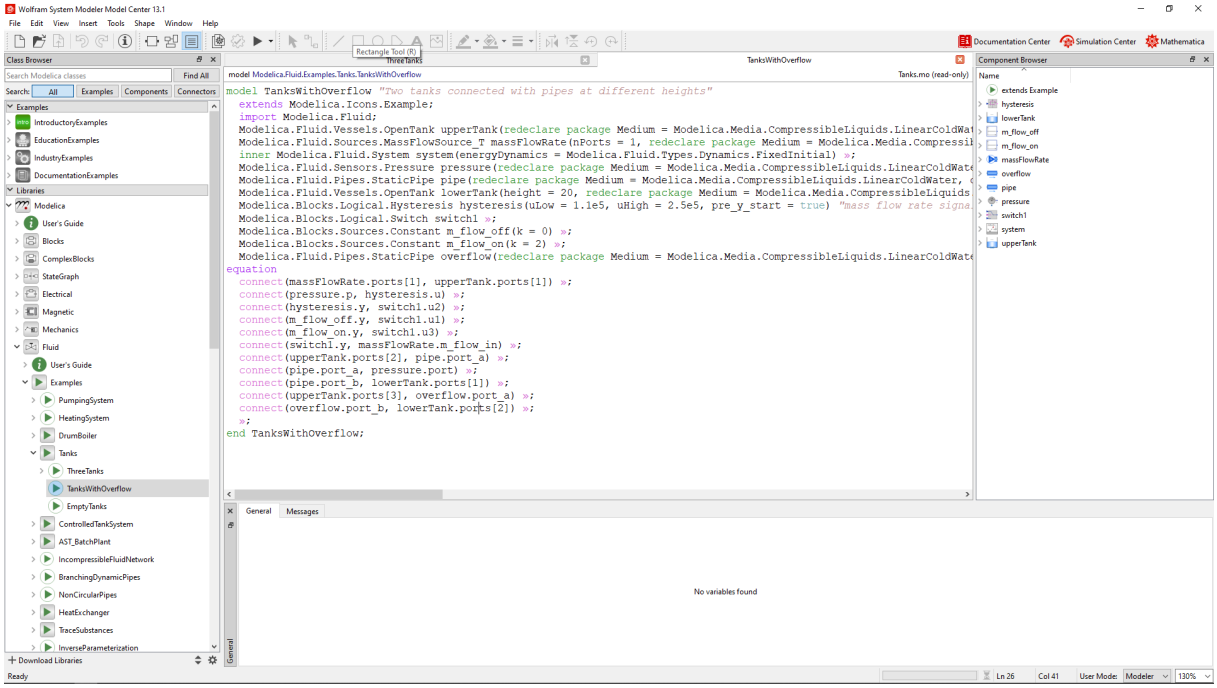

Figura 2.10: Interfaz Wolfram System Modeler 13.1: Vista de código

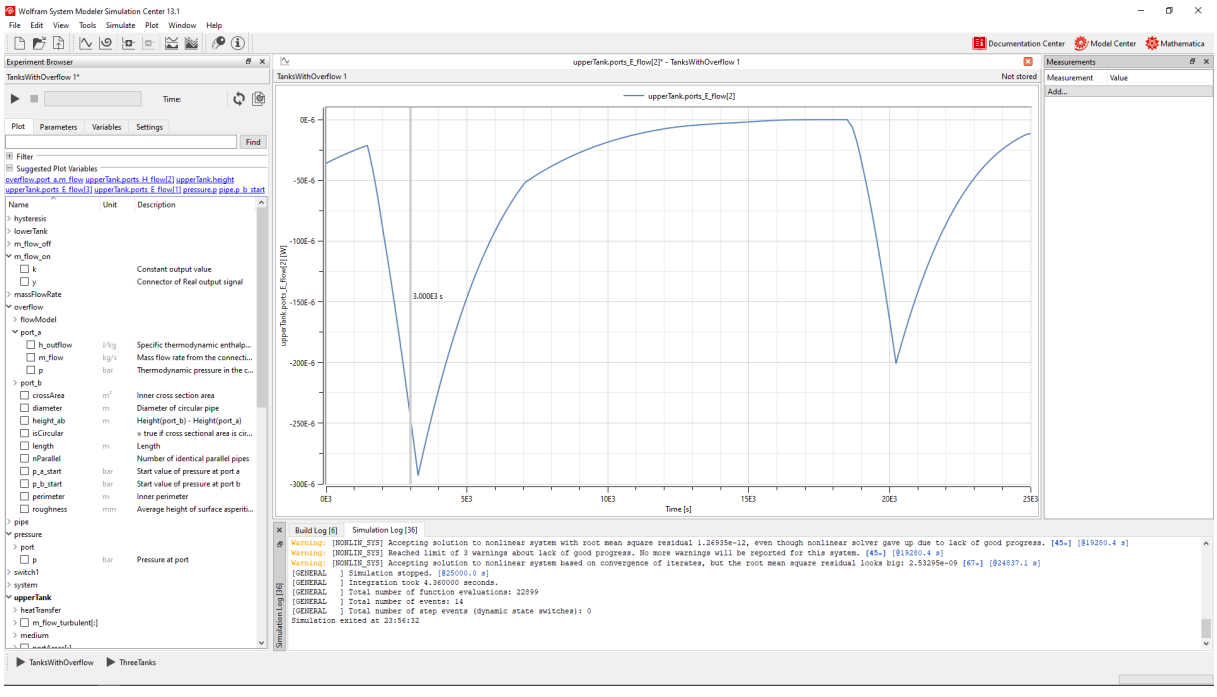

Figura 2.11: Interfaz Wolfram System Modeler 13.1: Vista de simulación

2.5.3. OpenModelica

OpenModelica es uno de los entornos de modelado y simulación de código abierto más ampliamente utilizados, tanto en el ámbito industrial como en el académico. Está respaldado por la organización sin fines de lucro Open Source Modelica Consortium (OSMC).

El objetivo fundamental de OpenModelica es proporcionar un entorno completo de modelado, compilación y simulación que sea de código abierto, lo que permite su distribución gratuita para fines de investigación, educación y uso industrial [OpenModelica, 2023].

Este entorno de desarrollo permite la ejecución interactiva de la mayoría de expresiones, algoritmos y funciones de Modelica. También ofrece una eficiente capacidad de compilación de modelos basados en ecuaciones y funciones Modelica en código C. El código C generado se combina con una biblioteca de funciones auxiliares, una biblioteca para el uso del tiempo de ejecución y un solucionador numérico DAE.

OpenModelica ofrece varias herramientas, muchas de las cuales son de código abierto. A continuación, se mencionan algunas de ellas:

Compilador OpenModelica Avanzado (OMC)

- Compilación del código a lenguaje C para la simulación.
- Proporciona una API para consultar el código cargado.
- Se puede utilizar desde la línea de comandos o de forma interactiva como un objeto Corba.

Shell interactivo de OpenModelica (OMShell)

- Controlador de sesión interactivo que analiza e interpreta los comandos y expresiones de Modelica, para la evaluación, simulación, trazado, etc.
- Permite trabajar con OMC.
- Contiene historial simple.

Editor de conexión de OpenModelica (OMEdit)

- Editor gráfico donde se diseñan los modelos y sus conexiones.
- Esta diseñado con C++ y bibliotecas de Qt.
- Representa el front-end, mientras que OMC es el back-end.

Cuaderno OpenModelica (OMNotebook)

- Cuaderno simple estilo Mathematica para Modelica.
- El objetivo es proporcionar una enseñanza avanzada de Modelica.
- Manejo del tutorial DrModelica.
- Está compuesto por celdas en las cuales se pueden incluir texto ordinario, modelos y expresiones de Modelica que pueden ser evaluados y simulados.

OpenModelica ofrece un conjunto amplio de herramientas, tanto de código abierto como propietarias. La Figura 2.12 ilustra cómo se relacionan las herramientas más importantes de código abierto.

2.6. Metodología, arquitectura y tecnologías

En esta sección, se detallará la metodología, la arquitectura y las tecnologías empleadas en el desarrollo de la aplicación, con el objetivo de cumplir los objetivos previamente descritos en el primer capítulo de esta memoria.

Para abordar la complejidad de la aplicación de manera eficiente, es necesario utilizar una metodología adecuada que permita dividir el problema en partes más manejables, siguiendo el principio de "divide y vencerás". La ingeniería de software y el desarrollo de software se basan en metodologías maduras desarrolladas en otros campos, como la arquitectura. Estas metodologías suelen incluir etapas como planteamiento, análisis, diseño, implementación, pruebas y puesta en marcha [Larman and Valle, 2003]. La forma en que se organiza y aborda cada una de estas etapas da lugar a diferentes tipos de metodologías. Se pueden distinguir metodologías tradicionales, que siguen un enfoque secuencial donde no se puede avanzar a la siguiente etapa sin completar la etapa anterior, y metodologías ágiles, que descomponen el proyecto en funcionalidades y priorizan la funcionalidad más crítica para obtener rápidamente una versión preliminar del producto y evaluar su viabili-

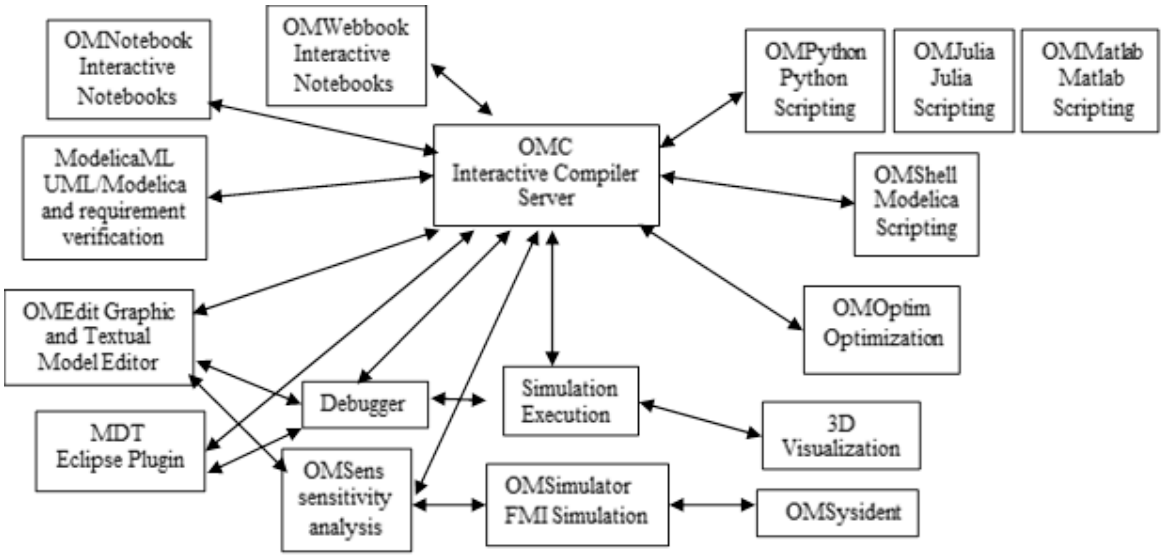

Figura 2.12: Estructura e interacción de las herramientas que conforman OpenModelica [OpenModelica, 2023].

dad. Estas metodologías tienen la particularidad que en caso necesario, se pueden realizar correcciones y agregar funcionalidades a medida que se obtienen versiones preliminares hasta finalmente obtener el diseño final completo con todos los requisitos.

En el desarrollo de esta aplicación se ha intentado utilizar una combinación de metodologías, motivados por experimentación, el aprendizaje y la mejora. En resumen, se puede decir que se ha realizado una metodología iterativa incremental, esto se verá con mayor detalle en los siguientes capítulos.

Respecto a la arquitectura de la aplicación, se ha optado por el patrón de diseño Modelo-Vista-Controlador (MVC), ampliamente utilizado en numerosas aplicaciones. Este patrón busca separar la interfaz de usuario (Vistas) de los datos de la aplicación (Modelos) mediante un Controlador que actúa como intermediario entre ambas partes.

En cuanto al lenguaje de programación, se ha elegido Java, un lenguaje de alto nivel ampliamente conocido y utilizado en la industria y la educación. Java es especialmente adecuado para aplicaciones complejas y ofrece una amplia gama de bibliotecas y herramientas. Para desarrollar una interfaz de usuario moderna y amigable, se ha utilizado JavaFX, una biblioteca que permite el diseño de interfaces de usuario dentro de Java.

2.6.1. Patrón Modelo-Vista-Controlador (MVC)

El patrón Modelo-Vista-Controlador (MVC) ha sido utilizado desde la década de los años 70, cuando se introdujo por primera vez en Smalltalk-76. A lo largo de los años, ha demostrado ser un enfoque válido para el desarrollo de aplicaciones en diversas plataformas, especialmente en aquellas orientadas a objetos. La principal ventaja de este patrón radica en su capacidad para dividir la aplicación en tres componentes principales:

- **El Modelo:** Contiene la representación de los datos manejados por la aplicación, es decir, el modelo de negocio.
- **La Vista:** Es la interfaz de usuario, encargada de la interacción con el usuario y la presentación de datos.
- **El Controlador:** Actúa como intermediario entre la Vista y los Modelos, gestionando el flujo de información entre ambos.

Esta separación brinda una gran versatilidad a la aplicación y reduce la complejidad en su diseño y construcción.

En la Figura 2.13 se muestra un diagrama genérico del patrón Modelo-Vista-Controlador (MVC). Las vistas interactúan con los usuarios, capturan sus acciones (eventos) y las transmiten al controlador, que se encarga de gestionar estos eventos. El controlador, a su vez, modifica o actualiza los modelos. Cuando los modelos detectan cambios, notifican a la vista para que esta refleje los cambios en la interfaz de usuario. Las vistas también pueden ser actualizadas directamente por el controlador, por ejemplo, cuando el usuario interactúa con la vista sin necesidad de manipular los modelos, como al maximizar o mover una ventana.

Este enfoque de diseño promueve la separación de de capas que facilitan el mantenimiento y la escalabilidad de las aplicaciones.

2.6.2. Lenguaje de programación Java

Java es un lenguaje de programación de alto nivel orientado a objetos. Fue originalmente desarrollado por James Gosling, en los laboratorios Sun Microsystem, fue comercializado por primera vez en el año 1995. Su sintaxis deriva en gran medida del lenguaje de programación C y C++, a diferencia de estos lenguajes su código no se traduce directamente a código máquina, sino que, se compila a un lenguaje intermedio denominado

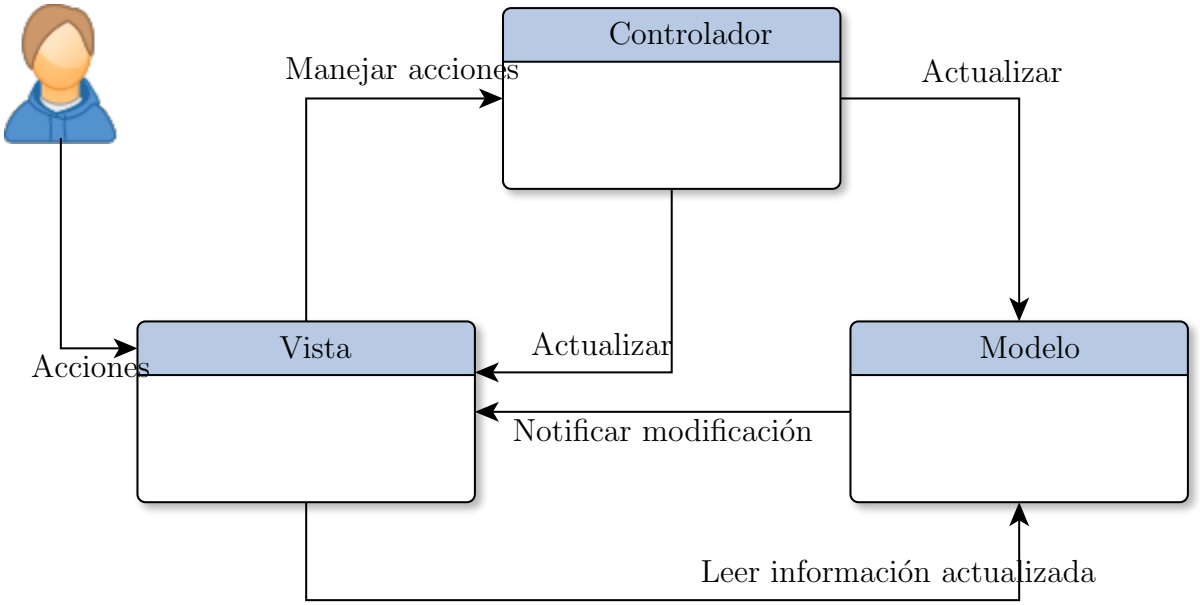

Figura 2.13: Diagrama del patrón Modelo-Vista-Controlador (MVC)

bytecode que puede ejecutarse en cualquier máquina virtual Java (JVM, del inglés Java Virtual Machine). El objetivo de usar un lenguaje intermedio es desacoplar el código de alto nivel (código de programación) de la máquina en la que se ejecuta el código (lenguaje máquina), no requiere compilarse para cada nueva máquina, simplemente con los byteco-de pueden ejecutarse en cualquier máquina con cualquier sistema operativo que contenga la máquina virtual Java (JVM), de hecho la promesa de Gosling era "Write Once, Run Anywhere".

Java ha experimentado numerosos cambios, fruto de ello son las diferentes versiones del entorno de desarrollo (JDK, del inglés Java Development Kit), partiendo de la versión JDK 1.0 hasta la actual a fecha de escritura de esta memoria que nos encontramos en la versión JDK 20. Su desarrollo empezó en Sun Microsystems pero posteriormente fue adquirida por Oracle, el 27 de enero de 2010. Destacar que en la versión JDK 8, lanzada en marzo de 2014 se incorpora completamente la librería JavaFx, librería que detallaremos más adelante por que se ha utilizado para el desarrollo de la aplicación de este proyecto. Además, en esa misma versión se añade la funcionalidad para programar programación funcional mediante expresiones Lambda, para más información consultar la documentación [Oracle, 2023].

El diseño robusto, la portabilidad y el respaldo de la industria y la educación han convertido a Java en un lenguaje de mayor crecimiento y amplitud en el ámbito de la programación, desarrollo software en diferentes plataformas.

Las plataformas o entornos en las que se utiliza Java se puede mencionar las siguientes:

- Dispositivos móviles.
- Sistema embebidos.
- Navegador Web.
- Servidores.
- Aplicaciones de escritorio.

En sus primeros días, Java contaba con la biblioteca AWT (Abstract Windows Toolkit) para componentes gráficos. Sin embargo, AWT tenía limitaciones, ya que dependía de los componentes nativos del sistema operativo en el que se ejecutaba la aplicación, lo que a veces resultaba en interfaces gráficas limitadas y poco atractivas. Para superar estas limitaciones, se introdujo la API Swing como parte de las Java Foundation Classes (JFC). Swing permitió a los desarrolladores crear interfaces de usuario más flexibles y atractivas, independientes del sistema operativo subyacente.

En la década de 2010, JavaFX se lanzó como una tecnología moderna para crear interfaces gráficas, compitiendo con otras tecnologías como Adobe Flash y Microsoft Silverlight. JavaFX proporciona una plataforma para crear interfaces web con las capacidades y características de las aplicaciones de escritorio, lo que resulta en una experiencia de usuario más enriquecedora y dinámica.

2.6.3. JavaFx

JavaFX es una plataforma de desarrollo de código abierto que se utiliza para crear aplicaciones de cliente enriquecidas en sistemas integrados, móviles y de escritorio utilizando el lenguaje de programación Java. Esta plataforma es el resultado de un esfuerzo colaborativo de numerosas personas y empresas con el objetivo de proporcionar un conjunto de herramientas moderno, eficiente y completo para la creación de aplicaciones de cliente [OpenJFX, 2023].

JavaFX ofrece un conjunto de herramientas y bibliotecas que permiten a los desarrolladores crear aplicaciones con interfaces gráficas de usuario (GUI) modernas, ricas en funciones y altamente personalizables. Fue desarrollado por Oracle, anteriormente conocido como Sun Microsystems, como una alternativa más avanzada y funcional a la biblioteca gráfica Swing, que a su vez fue el sucesor de AWT.

Algunas de las características y capacidades notables de JavaFX incluyen:

- **Gráficos avanzados:** JavaFX ofrece soporte para gráficos vectoriales y bitmap, lo que permite crear interfaces de usuario atractivas y escalables. También tiene capacidades para crear efectos visuales y animaciones.
- **Componentes personalizable:** Permite la creación de componentes de interfaz gráfica personalizados y altamente flexibles.
- **Controles avanzados:** Ofrece una amplia variedad de controles y elementos visuales predefinidos, como botones, listas, tablas, gráficos, etc.
- **Animaciones:** JavaFX facilita la creación de animaciones y transiciones fluidas en la interfaz de usuario.
- **Multimedia:** Proporciona soporte para la reproducción de audio y vídeo, así como para la creación de aplicaciones multimedia interactivas.
- **Diseño estándar:** Permite la aplicación de hojas de estilo CSS para definir la apariencia y el estilo de la interfaz de usuario.

JavaFX utiliza un lenguaje de marcado llamado FXML, basado en XML, para describir la estructura y la apariencia de las interfaces de usuario. Esto brinda una mayor flexibilidad en la creación y personalización de interfaces. Además, para facilitar el diseño de interfaces, existe una herramienta llamada **JavaFX Scene Builder** que permite construir interfaces de manera intuitiva mediante operaciones de arrastrar y soltar, siguiendo el principio "lo que ves es lo que obtienes"(WYSIWYG).

En resumen, JavaFX es una plataforma versátil que brinda a los desarrolladores las herramientas necesarias para crear aplicaciones de cliente modernas y atractivas con facilidad y eficiencia.

2.6.4. JavaFX Scene Builder

JavaFX Scene Builder es una herramienta gráfica que te permite diseñar y construir interfaces gráficas de usuario (GUI) para aplicaciones JavaFX de una manera visual y sin la necesidad de escribir código manualmente.

JavaFX Scene Builder emerge como una herramienta crucial en el campo del desarrollo de aplicaciones con interfaces gráficas de usuario (GUI) en el entorno Java. Con el auge de aplicaciones modernas y ricas en contenido visual, la creación de interfaces atractivas

y funcionales se ha convertido en un componente esencial para la experiencia del usuario. En este contexto, JavaFX Scene Builder desempeña un papel fundamental al proporcionar una plataforma visual para diseñar interfaces de usuario de manera eficiente y efectiva.

JavaFX Scene Builder ofrece una solución integral para diseñar interfaces gráficas de usuario sin la necesidad de involucrarse en la codificación manual de componentes visuales y su posicionamiento. Su enfoque WYSIWYG (What You See Is What You Get) permite a los desarrolladores y diseñadores crear y visualizar la apariencia de la interfaz en tiempo real, lo que fomenta una colaboración más fluida y una iteración eficaz en la etapa de diseño.

Este enfoque visual no solo acelera el proceso de desarrollo, sino que también mejora la calidad del diseño de interfaces gráficas. JavaFX Scene Builder permite arrastrar y soltar elementos visuales en una representación virtual del diseño, lo que simplifica la organización de los componentes, la estructura de la interfaz y la implementación de interacciones. Esta funcionalidad se convierte en una herramienta poderosa para los diseñadores, ya que pueden expresar sus ideas creativas de manera precisa sin estar limitados por el código.

Además de la facilidad de diseño, JavaFX Scene Builder promueve la separación de componentes y contenedores, que permiten la definición de la estructura de la interfaz en archivos FXML. Estos archivos XML describen la jerarquía de los componentes visuales, sus propiedades y los eventos asociados. Esta separación entre la estructura de la interfaz y la lógica subyacente favorece la modularidad, el mantenimiento y la escalabilidad del código, al tiempo que optimiza la colaboración entre diseñadores y desarrolladores. Información extraída de [Gluon, 2023].

En la Figura 2.14 podemos observar la herramienta de diseño de interfaces JavaFx. En la parte superior tenemos el menú típico de cualquier aplicación de Windows. En la parte izquierda tenemos un árbol con los diferentes componentes: botones, contenedores, paneles, layouts, etc. Estos elementos se puede arrastrar y soltar en el centro de la ventana (área de diseño). En la parte derecha tenemos la propiedades de cada elemento seleccionado, esta nos permitirá definir la posición, el tamaño, el color e incluso, incrustar código CSS para cambiar la apariencia utilizando la hoja de estilos en cascada.

Una vez que terminemos de diseñar, JavaFX Scene Builder nos construirá un fichero FXML que contendrá la estructura, diseño y apariencia en formato XML, similar al que se muestra en la 2.15.

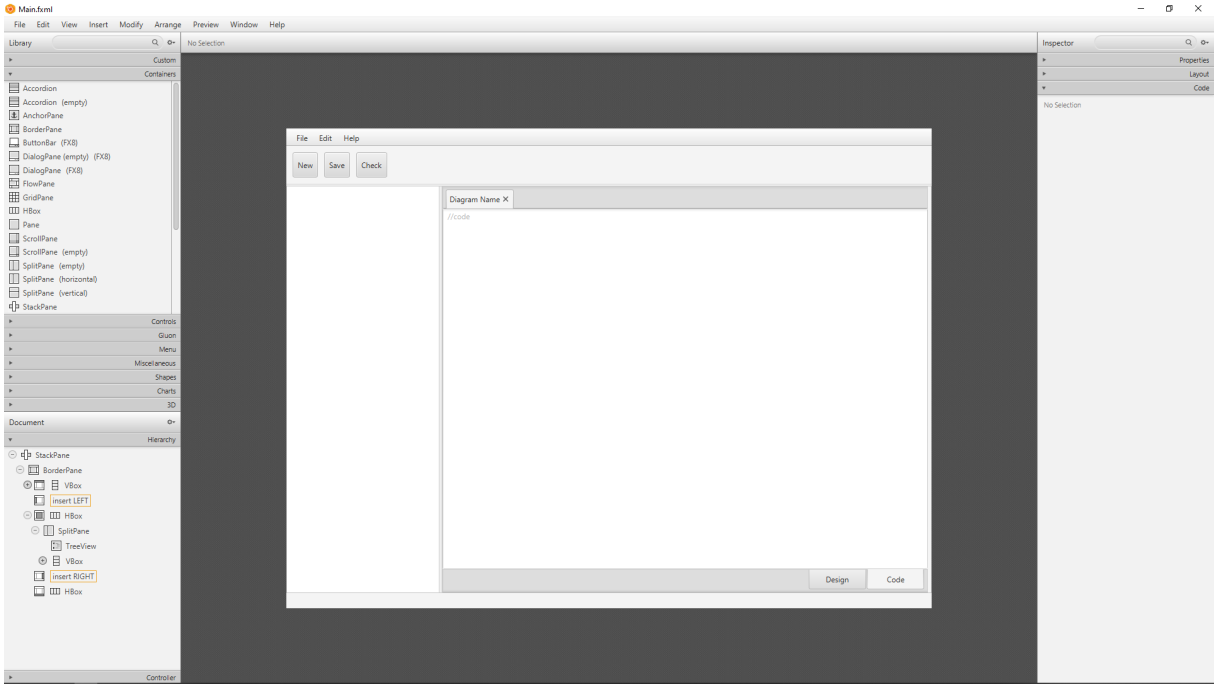

Figura 2.14: Interfaz gráfica de JavaFx Scene Builder.

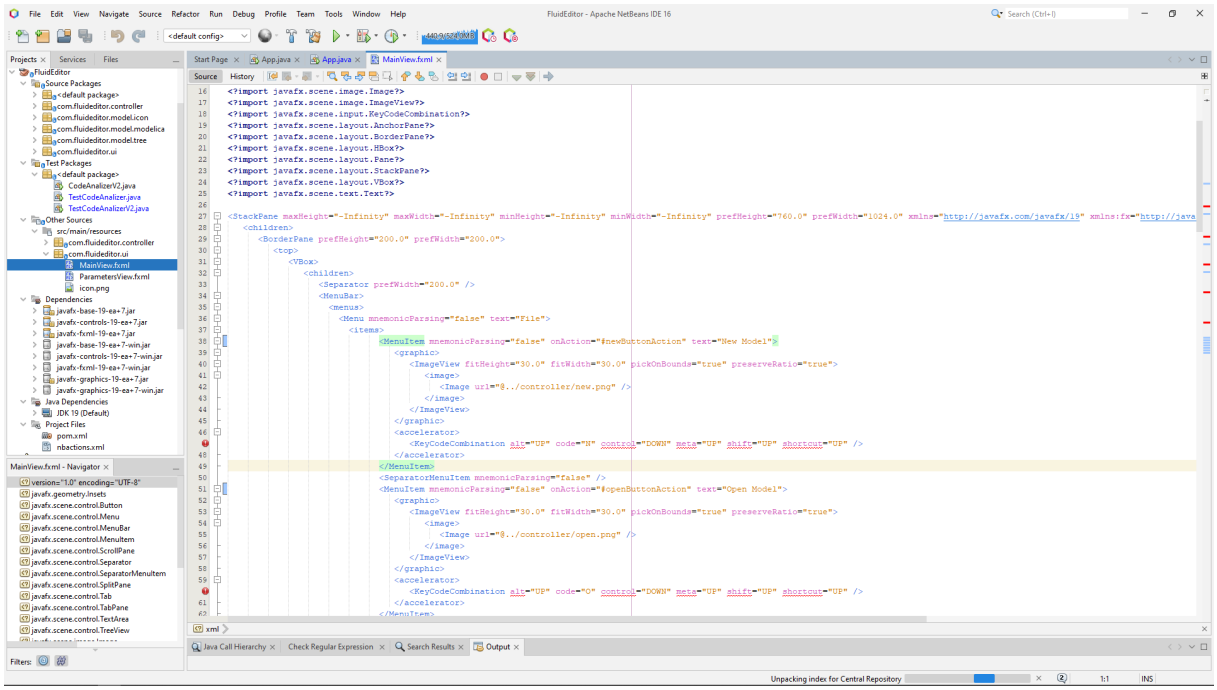

Figura 2.15: Código FXML generado utilizando JavaFX Scene Builder.

2.6.5. Expresiones regulares en Java

Las expresiones regulares, comúnmente conocidas como regex o regexp, son secuencias de caracteres que forman patrones de búsqueda. Estos patrones se utilizan para buscar, validar y manipular texto de una manera precisa y eficiente. En Java, las expresiones regulares se pueden utilizar gracias a las clases proporcionadas por el paquete **java.util.regex.** Este paquete incluye las siguientes clases fundamentales:

- **Pattern:** Esta clase define el patrón que se utilizará para la búsqueda de coincidencias en el texto.
- **Matcher:** Permite buscar en el texto el patrón definido en la clase Pattern.

Una expresión regular puede ser un solo carácter o un patrón más complejo. Se utilizan para cualquier tipo de búsqueda de patrones de texto, ya sea para extraer, sustituir o actualizar información en el texto. Para más información consultar en [Friedl, 2006].

En el Código 2.1, se presenta un ejemplo sencillo del uso de expresiones regulares en Java. En este ejemplo, se intenta buscar la palabra "UNED"dentro del siguiente texto: "Este es el texto en el que debe contener las coincidencias de búsqueda, en este caso UNED". Para lograrlo, se utiliza el método `compile()` de la clase `Pattern`, al cual se le pasa como argumento el patrón de búsqueda, que en este caso es la palabra "UNED". También se incluye en el argumento `Pattern.CASE_INSENSITIVE` para permitir la búsqueda tanto en minúsculas como en mayúsculas.

El método `compile()` genera un objeto del tipo `Pattern` que se le ha llamado `pattern`. En este objeto, mediante el método `matcher()` y proporcionándole como argumento el texto en el que deseamos buscar, se crea un objeto del tipo `Matcher` llamado `matcher`, que contendrá todas las coincidencias. En este objeto utilizando el método `find()`, podemos recorrer cada una de las coincidencias, lo que se denomina hacer un "match". En este ejemplo, solo queremos saber si existe o no la palabra buscada en el texto de búsqueda, por lo que simplemente incluimos una sentencia condicional que nos permita mostrar un mensaje. Si existe una coincidencia, se muestra el mensaje "Palabra encontrada"; en caso contrario, se muestra el mensaje "Palabra no encontrada".

```java
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Main {
  public static void main(String[] args) {
    Pattern pattern = Pattern.compile("UNED", Pattern.CASE_INSENSITIVE);
    Matcher matcher = pattern.matcher("Este es el texto en el que debe
        contener las coincidencia de búsquedas, en este caso UNED.");
    boolean matchFound = matcher.find();
    if(matchFound) {
      System.out.println("Palabra encontrada");
    } else {
      System.out.println("Palabra no encontrada");
    }
  }
}
```

Código 2.1: Ejemplo del uso de expresiones regulares en Java.

Para crear una expresión regular, se utilizan cuantificadores y metacaracteres. En la Tabla 2.3 y 2.4 se describen algunos de los cuantificadores y metacaracteres más utilizados en expresiones regulares.

Cuantificador	Descripción
`n+`	Encuentra cualquier cadena con al menos un carácter `n`.
`n*`	Encuentra cero o más ocurrencias de `n` en la cadena.
`n?`	Encuentra la aparición de `n` cero o una vez en la cadena.
`n{x}`	Encuentra la secuencia de `n` exactamente `x` veces.
`n{x,}`	Encuentra una secuencia de `n` al menos `x` veces en la cadena.

Tabla 2.3: Cuantificadores en expresiones regulares.

Estos cuantificadores y metacaracteres permiten crear patrones de búsqueda flexibles y poderosos que pueden adaptarse a una amplia variedad de necesidades en el procesamiento de texto con Java. En este trabajo han sido clave para extraer la información de los ficheros Modelica.

2.6.6. Entorno de Desarrollo Integrado (IDE)

Un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés, Integrated Development Environment) es una aplicación que proporciona un conjunto de herramientas destinadas a facilitar y agilizar el proceso de desarrollo de software. Estos entornos suelen incluir un editor de código fuente que resalta las palabras clave del lenguaje de programación, permite la auto-completado inteligente del código (IntelliSense), así como herramientas de depuración y compilación.

Metacaracter	Descripción
|	Símbolo para indicar un OR. Es decir, para indicar una alternativa.
.	Encuentra cualquier carácter.
^	Sirve para hacer match al principio del string.
$	Hace match al final de un String.
\d	Coincide con un dígito: [0-9]
\D	Coincide con cualquier cosa excepto un dígito: [^0-9]
\s	Coincide con un carácter de espacio en blanco (espacio, tabulación, nueva línea): [\t\n\r\f\v]
\S	Coincide con cualquier cosa excepto un carácter de espacio en blanco: [^ \t\n\r\f\v]
\b	Hace match con una palabra completa
\w	Coincide con un carácter de palabra (letras, dígitos o guiones bajos): [a-zA-Z0-9_]
\W	Coincide con cualquier cosa excepto un carácter de palabra: [^a-zA-Z0-9_]
\uxxxx	Encuentra el carácter Unicode especificado por el número hexadecimal xxxx.

Tabla 2.4: Metacaracteres en expresiones regulares.

Para cada lenguaje de programación, existen varios IDE disponibles. Por ejemplo, para Java, se pueden mencionar Eclipse, NetBeans, IntelliJ IDEA y Visual Studio Code, entre otros. Muchos de estos IDE son multiplataforma, lo que significa que funcionan en diferentes sistemas operativos, y también son compatibles con múltiples lenguajes de programación.

La elección de un IDE depende en gran medida de las preferencias personales del desarrollador, de la experiencia con los IDEs y en pocas ocasiones del lenguaje de programación. Para el desarrollo de la aplicación de este proyecto, se optó por utilizar NetBeans, un IDE de código abierto ampliamente popular con una sólida integración con el lenguaje Java.

En la Figura 2.16, se muestra la interfaz de NetBeans, donde se puede observar el árbol de directorios del proyecto en la parte izquierda, el editor de código fuente en el centro y la salida por consola en la parte inferior. El IDE brinda múltiples configuraciones de personalización, así como la integración con gestores de paquetes como Maven y gestor de versiones como Git. Para más información sobre NetBeans visite [Apache NetBeans, 2023].

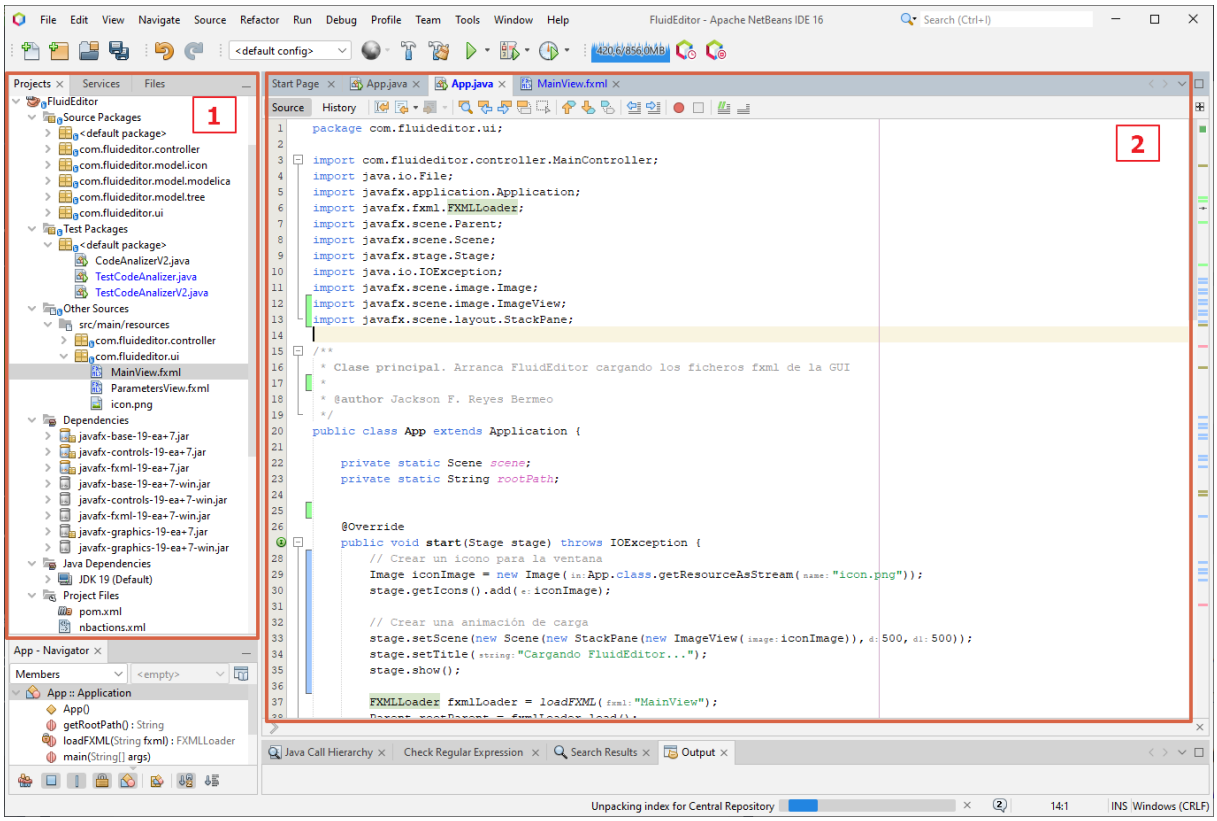

Figura 2.16: Interfaz del Entorno de Desarrollo Integrado (IDE) NetBeans: 1) Árbol de directorios del proyecto, 2) Editor de código fuente.

2.7. Conclusiones

En este capítulo, se ha realizado una revisión de varios conceptos fundamentales relacionados con el modelado y la simulación. También se ha abordado el tema del lenguaje Modelica, los entornos de modelado y simulación. Se ha llevado a cabo una revisión de algunas de las tecnologías que serán empleadas en el desarrollo de la aplicación. Estas tecnologías incluyen el lenguaje de programación Java, la biblioteca JavaFX y la tecnología de expresiones regulares, las cuales desempeñan un papel crítico en la extracción de información de los archivos Modelica. Además, se ha enfatizado en el patrón MVC, que constituye el enfoque principal para el desarrollo de esta aplicación.

3 ANÁLISIS Y PLANIFICACIÓN

3.1. Introducción

En este capítulo, realizaremos un análisis preliminar antes de abordar el diseño y desarrollo de la aplicación. Estableceremos los requisitos, discutiremos la interacción del usuario en la aplicación y, finalmente, presentaremos un diagrama de Gantt en el que se muestra la planificación del proyecto.

Cada aplicación tiene su origen en la necesidad de abordar problemas, que a menudo son complejos. Para reducir esta complejidad, descomponemos los problemas en partes más pequeñas. La solución de estos problemas parciales se traduce en un conjunto de requisitos que la aplicación debe satisfacer. Es importante distinguir entre dos tipos de requisitos: los funcionales y los no funcionales. Los requisitos funcionales describen las especificaciones que el sistema debe cumplir, mientras que los requisitos no funcionales se refieren a criterios de calidad, restricciones y características que afectan al rendimiento, la seguridad y otros aspectos del sistema, como la usabilidad y la mantenibilidad, entre otros.

3.2. Catálogo de requisitos

En un entorno de desarrollo profesional y comercial, los requisitos de la aplicación desempeñan un papel fundamental. Es esencial garantizar que estas funcionalidades sean comprendidas adecuadamente por todas las partes involucradas en el desarrollo. A menudo, el cliente y el equipo de desarrollo provienen de contextos muy diferentes, lo que puede dar lugar a malentendidos. Por esta razón, se han desarrollado diversas metodologías para reducir esta brecha de comprensión. En última instancia, es responsabilidad del jefe de proyecto establecer mecanismos que permitan una especificación clara de los requisitos y garantizar que el cliente tenga una comprensión precisa de ellos [Wiegers, 2003].

Cada aplicación tiene su origen en la necesidad de abordar problemas, que a menudo son complejos. Para reducir esta complejidad, descomponemos los problemas en partes más pequeñas. La solución de estos problemas parciales se traduce en un conjunto de requisitos que la aplicación debe satisfacer.

Es importante distinguir entre dos tipos de requisitos: los funcionales y los no funcionales. Los requisitos funcionales describen las especificaciones que el sistema debe cumplir, mientras que los requisitos no funcionales se refieren a criterios de calidad, restricciones y características que afectan al rendimiento, la seguridad y otros aspectos del sistema, como la usabilidad y la mantenibilidad, entre otros [Larman and Valle, 2003].

En el caso de esta aplicación, dado que el cliente y el desarrollador son la misma persona, es probable que los requisitos estén lo suficientemente claros. No obstante, es una buena práctica mantener un registro de los requisitos, especialmente en lo que respecta a los requisitos funcionales. A medida que el proyecto aumenta en complejidad, es fácil perder el rumbo y terminar con funcionalidades que inicialmente no se habían contemplado o que resultan innecesarias. Por tanto, se llevará a cabo una especificación de los principales requisitos que debe cumplir la aplicación. La forma en que se realiza la especificación puede variar según la metodología utilizada. En este caso, se emplearán tablas que definirán los requisitos y sus respectivas subtareas, acompañados de una breve descripción. Lo más importante es que los requisitos estén detallados de manera clara y concisa, de modo que puedan ser comprendidos tanto por los desarrolladores como por el cliente. Esto servirá como un registro contractual de los requisitos acordados. En la Tabla 3.1, se presentan los requisitos funcionales que la aplicación debe cumplir.

Tabla 3.1: Requisitos funcionales del Editor de Modelos Hidráulicos.

Requisitos funcionales (RF)		Descripción
RF1 - Funcionalidad del Editor		Principales funcionalidades del editor de modelos hidráulicos.
	RF1.1 Paleta de componentes	El editor debe tener una paleta de componentes de donde se selecciona y arrastra al área de diseño, los componentes estarán agrupados en forma de árbol.
	RF1.2 Área de diseño	Área de diseño en la cual los componentes pueden: ▪ moverse, ▪ seleccionarse, ▪ eliminarse.
	RF1.3 Área de código	Debe existir una área en la que se muestre el código del modelo compuesto.

Continúa en la siguiente página

Tabla 3.1 – Continuación de la tabla en la página anterior

Requisitos funcionales (RF)		Descripción
RF2 - Gestión de modelos		Posibilidad de crear modelos desde cero o desde modelos previamente guardados.
	RF2.1 Eliminar modelos	Debe existir la posibilidad de eliminar completamente el modelo.
	RF2.2 Guardar modelos	La aplicación debe tener la capacidad de guardar los modelos para su posterior edición.
	RF2.3 Abrir modelos	Los modelos pueden cargarse (abrir) desde un fichero .mo previamente guardado.
RF3 - Diseño del modelo		Posibilidades durante el diseño del modelo.
	RF3.1 Propiedades componente	Cada componente debe tener sus propiedades que se puedan visualizar y editar (parámetros).
	RF3.2 Eliminar componente	Debe existir la posibilidad de eliminar componentes individuales.
	RF3.3 Conexiones entre componentes	Posibilidad de conectar dos componentes desde cada uno de sus conectores del propio componente.
	RF3.4 Eliminar conexión	Posibilidad de eliminar conexiones individuales.
	RF3.5 Eliminación de componentes con conexión	Cuando se seleccione y se elimine un componente, las conexiones asociadas a este componente, también deberán eliminarse.
RF4 - Generación de código		
	RF4.1 Código en formato compuesto	Capacidad de generar código Modelica del modelo compuesto, instanciando las clases de la librería estándar de Modelica (MSL).
	RF4.2 Posicionamiento de los componentes	Los posicionamientos deben calcularse de manera correcta para que cualquier entorno de modelado y simulación los reconozca de manera coherente.

3.3. Análisis y especificaciones

Para diseñar una aplicación eficaz, resulta fundamental comprender cómo los usuarios interactuarán con ella. De esta forma, se pueden establecer y concretar de mejor manera los requisitos funcionales. Además de cumplir con los requisitos funcionales y no funcionales, es esencial mantener una visión clara del objetivo de la aplicación, tal como se detalló en el primer capítulo. Esta comprensión sólida sienta las bases para un desarrollo adecuado de la misma.

En esta fase de análisis, hemos identificado un flujo típico de uso de la aplicación. Este flujo consta de algunos procesos clave. En primer lugar, el usuario inicia la aplicación y selecciona componentes Modelica de la paleta de componentes, luego los arrastra al área de diseño para realizar las conexiones necesarias entre ellos y construir su modelo. Posteriormente, guarda el modelo en un archivo para transportarlo a otro entorno de modelado y simulación. De este proceso, destacamos tres elementos esenciales:

- **Selección de componentes:** La aplicación debe brindar a los usuarios la posibilidad de seleccionar componentes predefinidos en la interfaz, estos componentes pueden ser definidos mediante un icono estático o extraerlos de las anotaciones de la librería estándar Modelica (MSL). En este contexto, hemos optado por implementar una funcionalidad que pueda leer los archivos Modelica y, a partir de esta lectura, extraiga la información de las anotaciones y genere un árbol de componentes.
- **Iconos de los componentes:** Es crucial proporcionar a los usuarios una librería gráfica que permita representar cada uno de los componentes Modelica mediante un icono. Estos iconos se generarán a partir de gráficas primitivas extraídas de las anotaciones presentes en los archivos Modelica. Para cumplir con este requerimiento, hemos optado por desarrollar una librería propia que tenga dicha funcionalidad. Esta librería seguirá una estructura de clases similar a la de las anotaciones Modelica, pero se implementará en Java.
- **Análisis e interpretación del código Modelica:** La capacidad de analizar e interpretar el código Modelica de cada componente es esencial para garantizar la generación y ejecución adecuada de los modelos. En este caso, hemos desarrollado la funcionalidad requerida utilizando las expresiones regulares para extraer la información y de esta forma, construir objetos Java que representen dicha información.

Esta fase de análisis y especificaciones establece las bases para el desarrollo de la aplicación, asegurando que se cumplan los requisitos y se satisfagan las necesidades de los usuarios de manera efectiva y eficiente.

3.4. Planificación

Para llevar a cabo el desarrollo de este proyecto, se siguieron las fases típicas de la ingeniería del software: análisis, diseño, implementación y pruebas. Estas fases no se llevaron a cabo de forma estrictamente secuencial; más bien, se realizaron de manera iterativa e incremental. Esto significa que en cada iteración se llevaron a cabo todas las fases, sin restricciones para retroceder a una fase anterior si era necesario. Cada iteración permitía agregar nuevas funcionalidades o mejorar las ya implementadas. El resultado de cada iteración no consistía simplemente en un prototipo, sino en una parte funcional de la solución completa de la aplicación. Cada iteración ha sido guiada por las propias funcionalidades de la aplicación, es decir, los requisitos de la aplicación.

Durante cada iteración, el enfoque se centra en la implementación de una funcionalidad crítica o en la mejora de una ya existente. Esto tiene la ventaja de ayudar a identificar la complejidad global de la aplicación de manera temprana. Inicialmente, la aplicación carece de funcionalidad, lo que permite concentrarse exclusivamente en las funcionalidades críticas sin que el resto de las funcionalidades agregue complejidad a la funcionalidad en desarrollo. A medida que se agregan más funcionalidades, la aplicación se vuelve más compleja, pero las funcionalidades finales son menos críticas. Esto contribuye a mantener la complejidad global más o menos constante. Por otro lado, si se comenzara con funcionalidades simples y se dejaran las funcionalidades críticas para el final, sería difícil gestionar la complejidad y, en el peor de los casos, podríamos descubrir al final que la viabilidad de la aplicación no es factible en términos de tiempo y recursos, lo que conduciría al fracaso del desarrollo de toda la aplicación

De acuerdo con lo anterior, durante las fases de análisis, diseño y implementación, se hizo hincapié en la identificación de los requisitos críticos. Identificar estos requisitos puede ser un desafío, especialmente cuando se carece de experiencia o se desconoce el dominio al que pertenece la aplicación. En tales casos, es recomendable priorizar los requisitos en los que se tenga menos experiencia, ya sea porque se esté utilizando una tecnología nueva o porque falte el conocimiento necesario del dominio de la funcionalidad.

Para llevar a cabo estas funcionalidades, cada iteración se divide en tareas con el objetivo de descomponer la complejidad. La Tabla 3.2 describe las tareas relacionadas con cada iteración. El objetivo final de cada iteración es obtener una funcionalidad nueva o mejorar una ya existente.

Tabla 3.2: Detalle de las tareas de cada iteración relacionadas con los requisitos funcionales.

Iteración	Tareas	RF relacionado
1 - Pruebas técnicas en Java. En esta iteración se pretende adquirir conocimientos específicos que se utilizara en el lenguaje Java: arrastrar elementos, mover elementos, eventos clic del ratón, utilizar objetos TreeView, etc.	1.1 Prueba de concepto drag and drop. 2.2 Prueba de concepto de expresiones regulares. 1.3 Prueba de concepto sobre el árbol de componentes. 1.4 Prueba sobre el manejo de eventos. 1.5 Pruebas con JavaFX.	RF1-4 RF1.2 RF3.1, RF4 RF1 RF3,RF2
2 - Implementación lector de ficheros Modelica. En esta iteración se intenta implementar la funcionalidad que sea capaz de leer un fichero Modelica, extraer el código de los componentes y generar un árbol en el que cada nodo contenga su código correspondiente al componente.	2.1 Cargar el fichero en memoria. 2.2 Eliminar comentarios, líneas en blanco. 2.3 Eliminar saltos de línea en cada instrucción. 2.4 Crear un árbol de componentes con el código Modelica extraído de los ficheros.	RF4 RF4 RF4 RF1, RF4
3 - Implementar analizador de código Modelica. Se pretende crear la funcionalidad de analizar el código Modelica extraído de los ficheros y que se encuentra en cada nodo del árbol de componentes.	3.1 Extraer las declaraciones de un modelo. 3.2 Extraer las ecuaciones de cada modelo. 3.3 Extraer declaraciones, ecuaciones de la herencia. 3.4 Extraer declaraciones, ecuaciones de la composición.	RF3.1, RF4 RF3.1, RF4 RF3.1, RF4 RF3.1, RF4

Continúa en la siguiente página

Tabla 3.2 – Continuación de la tabla en la página anterior

Iteración	Tareas	Requisito relacionado
4 - Implementación librería gráfica. En esta iteración se intenta implementar un conjunto de clases que permitan crear una representación de los iconos de cada componente Modelica a partir de primitivas gráficas. La información se extrae de las anotaciones.	4.1 Implementar una clase encargada de crear un Rectángulo a partir de la anotación modélica. 4.2 Implementar una clase que represente a la Elipse. 4.3 Implementar una clase que represente una Línea. 4.4 Implementar una clase que represente el texto del icono de un componente Modelica. 4.4 Implementar una clase que represente un Bitmap.	RF3 ,RF4.2 RF3 ,RF4.2 RF3 ,RF4.2 RF3 ,RF4.2 RF3 ,RF4.2
5 - Implementación de la interfaz gráfica. En esta iteración se desarrolla la Interfaz gráfica de usuario (GUI) utilizando JavaFX Scene Builder.	5.1 Creación de los layouts de la interfaz. 5.2 Diseño de la barra de herramientas. 5.3 Diseño del visualizador del árbol de componentes. 5.4 Diseño del canvas donde se arrastrar los componentes y del visualizador de código.	RF1, RF3 RF2 RF1 RF1,RF3
6 - Implementación de los controladores. En esta iteración se implementa la lógica de la aplicación haciendo uso de las anteriores funcionalidades.	6.1 Implementación de los diferentes eventos que se producen en la interfaz con su correspondiente código manejador. 6.2 Implementación del flujo de operación de la aplicación. 6.3 Implementación del controlador auxiliar para manejar las propiedades de los componentes.	RF2, RF2 RF1-4 RF3.1
7 - Pruebas globales. Pruebas globales de las distintas funcionalidades implementadas.	7.1 Pruebas de guardado y carga de modelos. 7.2 Pruebas de generación de código Modelica funcional compatible con otros entornos de modelado. 7.3 Pruebas de integración de la aplicación.	RF2 RF4 RF1-4

Para obtener una visión más completa de la planificación del proyecto, se adjunta un diagrama de Gantt que detalla las diversas actividades realizadas durante el desarrollo de esta aplicación. El diagrama se presenta en la Figura 3.1. Es importante notar que algunas tareas se ejecutan en paralelo, mientras que otras se llevan a cabo una vez

finalizada la tarea anterior. Esto ha sido posible gracias a la aplicación del patrón Modelo-Vista-Controlador (MVC) y los principios de la programación orientada a objetos, que desacoplan la aplicación y permiten trabajar en paralelo en distintas funcionalidades.

Es relevante mencionar que algunas tareas específicas no se han incluido en el diagrama de Gantt con el fin de evitar una representación excesivamente detallada que pudiera saturar la visualización con numerosas actividades. Estas tareas no incluidas abarcan las pruebas individuales de clases, la refactorización de código, la implementación de varias alternativas para una misma funcionalidad, la elaboración de la memoria, entre otras.

Figura 3.1: Diagrama de Gannt de las actividades para el desarrollo del proyecto (desarrollado con la herramienta online https://www.onlinegantt.com/#/gantt).

3.5. Conclusiones

En este capítulo, hemos profundizado en el análisis inicial del desarrollo de la aplicación. Se han detallado los requisitos funcionales y las diversas tareas necesarias para llevar a cabo cada una de las iteraciones hasta obtener una aplicación con todas las funcionalidades especificadas. Además, hemos creado un diagrama de Gantt que refleja la planificación del proyecto, mostrando el progreso y la secuencia de actividades.

4 ARQUITECTURA DE LA APLICACIÓN

4.1. Introducción

Este capítulo detalla la arquitectura de la aplicación, resaltando sus principales componentes. Como se mencionó en capítulos anteriores, esta aplicación sigue el patrón arquitectónico Modelo-Vista-Controlador (MVC).

4.2. Componentes de la aplicación

La Figura 4.1 muestra los paquetes Java que conforman la arquitectura de la aplicación. Estos paquetes se han organizado de acuerdo a los tres capas del patrón MVC.

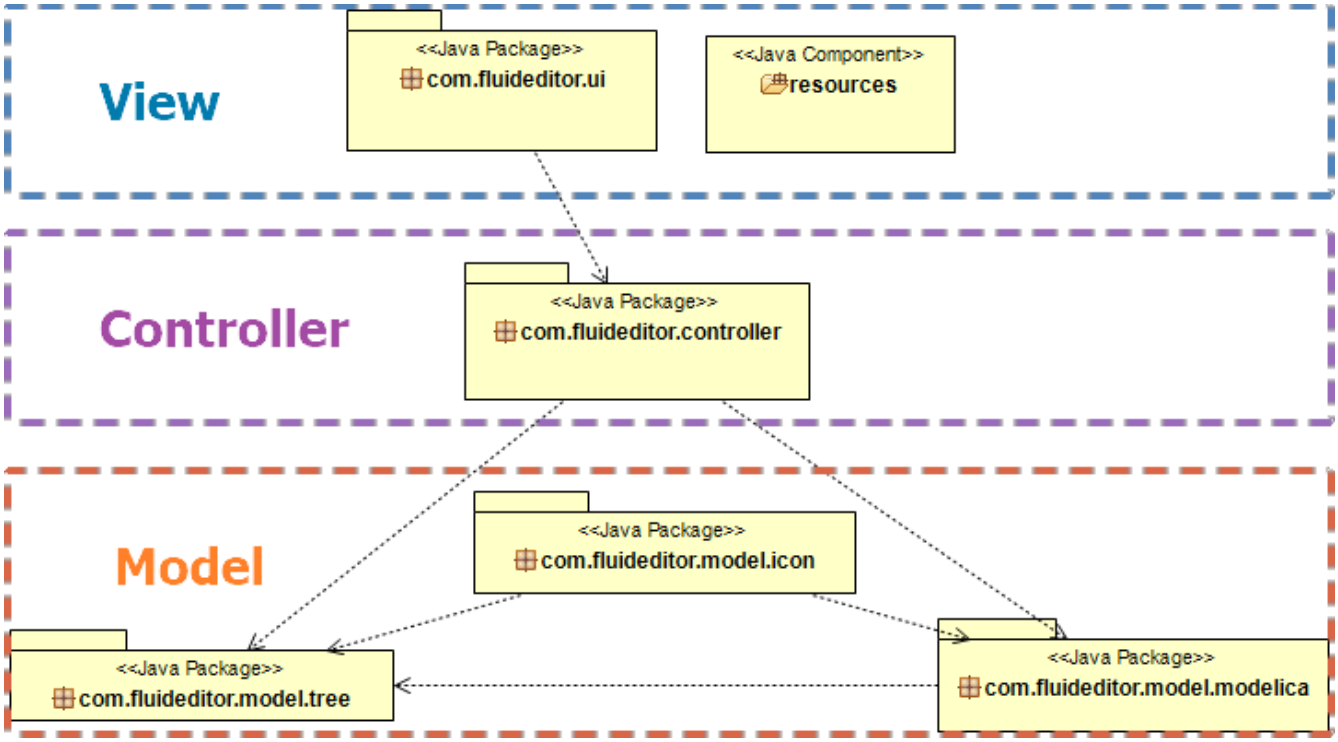

Figura 4.1: Diagrama Modelo-Vista-Controlador (MVC) de la aplicación.

4.2.1. Vista (View)

La vista se encarga de recibir y mostrar los resultados durante la interacción del usuario con la aplicación. Dado que esta es una aplicación gráfica, la interacción se lleva a cabo a través de la interfaz gráfica de usuario (GUI). Por lo tanto, en este bloque, todo está relacionado con la GUI. Los paquetes que conforman la vista son los siguientes:

- **resources:** Este paquete contiene los archivos FXML que describen el diseño de la interfaz gráfica de usuario (GUI). Estos archivos están escritos en código XML. La mayor parte de este código se ha generado utilizando JavaFX Scene Builder, una herramienta que permite crear interfaces gráficas mediante la función de arrastrar y soltar componentes en el área de diseño. Sin embargo, también existe la posibilidad de trabajar directamente con el archivo FXML si se tiene conocimiento en XML. En este paquete también se almacenan los diferentes iconos utilizados en la aplicación, en resumen, aquí se guardan todos los recursos necesarios para el correcto funcionamiento de la interfaz gráfica de usuario.
- **com.fluideditor.ui**: En este paquete se encuentran las clases responsables de cargar la interfaz gráfica, crear instancias del controlador y realizar configuraciones previas a la visualización de la GUI. Básicamente, este paquete se encarga de iniciar y mantener el flujo de interacción de la aplicación hasta que el usuario decida cerrarla.

4.2.2. Controlador (Controller)

El controlador será el manejador de los eventos que reciba la interfaz gráfica de usuario, además interactuará con los diferentes modelos. Es decir, el controlador se encargará de recibir los eventos del usuario, interpretarlos y tomar decisiones de como deben ser procesados y que datos o modelos va necesitar para dar repuesta a esas eventos (peticiones). Entre las principales responsabilidades se pueden destacar: recepción de entradas de la interfaz, interpretación de entradas, actualización de los modelos (lectura, escritura, actualización y eliminación), actualización de la vista (notificar a la vista para que actualice y muestre los cambios), gestión del flujo de control (el controlador puede llamar a otros controladores para mostrar otras vistas en función de las peticiones del usuario).

En el diseño de esta capa de abstracción se ha incluido un único paquete el mismo que contiene dos clases que implementaran los dos controladores de la aplicación, estos controladores son:

- **MainController:** Controlador principal que maneja las todas las interacciones de la interfaz gráfica de usuario de la aplicación.
- **PropertiesViewController:** Controlador que maneja la visualización/edición de los parámetros de cada componente Modelica que constituye el modelo que se esta diseñando.

4.2.3. Modelos (Model)

Los modelos contienen los datos y las reglas de la aplicación, es decir, nos proporcionan la representación subyacente de los datos y describen cómo se manipulan, procesan y almacenan. Los modelos actúan como una capa de abstracción entre los datos y la interfaz de usuario, se cominican mediante el controlador. Entre sus principales responsabilidades se incluyen: La gestión de datos, la definición de las reglas de manipulación y procesamiento de los datos (lógica del dominio), la notificación de los cambios en los datos a las partes interesadas (por ejemplo, utilizando el patrón de diseño Observer), entre otras.

Para el diseño de esta capa de abstracción se han incluido tres paquetes.

- **com.fluideditor.model.icon:** Este paquete contiene todas las clases necesarias para crear los iconos de los componentes Modelica. Los iconos se crean utilizando primitivas gráficas como rectángulos, elipses, líneas, etc. A este paquete se le denomina "librería gráfica". La información necesaria para generar los iconos de los componentes Modelica se extrae de las anotaciones presentes en el código Modelica.
- **com.fluideditor.model.tree:** Aquí se encuentran las clases que permiten generar el árbol de componentes Modelica. Este paquete incluye principalmente la clase encargada de leer los archivos Modelica y la clase que representa la información del código de cada componente, dicha información se extrae de los propios archivos de los modelos de la librería Fluid.
- **com.fluideditor.model.modelica:** En este paquete se encuentran las clases que representan a las clases Modelica descritas en Java que permiten crear objetos con la información de los modelos que sean manejables dentro de Java. Algunos ejemplos de objetos incluidos son el Modelo, el Componente, el Conector, la Conexión, los Parámetros, etc.

4.3. Lector de ficheros Modelica

Dentro de la capa de abstracción del modelo dentro del patrón MVC, podemos identificar una clase Java que se encarga de la lectura de los archivos Modelica. Esta clase adquiere una importancia crítica, ya que el funcionamiento integral de la aplicación se apoya en su capacidad para llevar a cabo una lectura precisa de cada uno de los ficheros. Un simple error en la lectura de cualquiera de estos archivos podría tener como consecuencia la falta de carga de otros componentes que no serían mostrados en la aplicación. En tal caso, la indisponibilidad de estos componentes podría resultar en la inoperancia total de la aplicación.

En esta sección, se pretende brindar un panorama general de cómo opera esta clase en particular. El primer paso en esta tarea es leer el archivo Modelica directamente mediante el uso de las clases predefinidas que Java proporciona para gestionar flujos de entrada y salida de datos. Cada línea del archivo se carga en una lista, que posteriormente será sometida a un proceso de transformación, al que llamaremos "lista plana"(listFlat). En este proceso, se eliminan los espacios en blanco, los comentarios y se unen las instrucciones que están divididas en múltiples líneas, de modo que cada línea contenga una instrucción completa (reconocible por un punto y coma al final de cada instrucción).

Luego se procede a extraer las principales propiedades del archivo, como el nombre, el tipo (si es un modelo, una clase, un paquete, etc.), la ruta, entre otras. Una vez que se obtiene la lista aplanada, el siguiente paso es construir un árbol que represente cada uno de los componentes descritos en el archivo. Esto implica iterar a través de la lista aplanada y extraer los componentes respetando la jerarquía en la que se encuentran. Por cada línea extraída de la lista aplanada, se lleva a cabo un análisis para determinar si contiene una palabra clave. Aquí, se consideran como palabras clave aquellas que están reservadas por el lenguaje Modelica para describir cada uno de los componentes, como "Model", "partial Model", "Class", "Block", "Package", entre otras.

Si se detecta una palabra clave, se invoca nuevamente la misma función para repetir el proceso de construcción del árbol (esto implica el uso de un método recursivo). Si no se identifica ninguna palabra clave, se asume que la línea se refiere a una instrucción dentro del componente actual. En tal caso, la instrucción se agrega a una lista de código específica para el componente en proceso de análisis. En la Figura 4.2 se observa un diagrama de flujo que intenta resumir lo comentado en este y en párrafos anteriores.

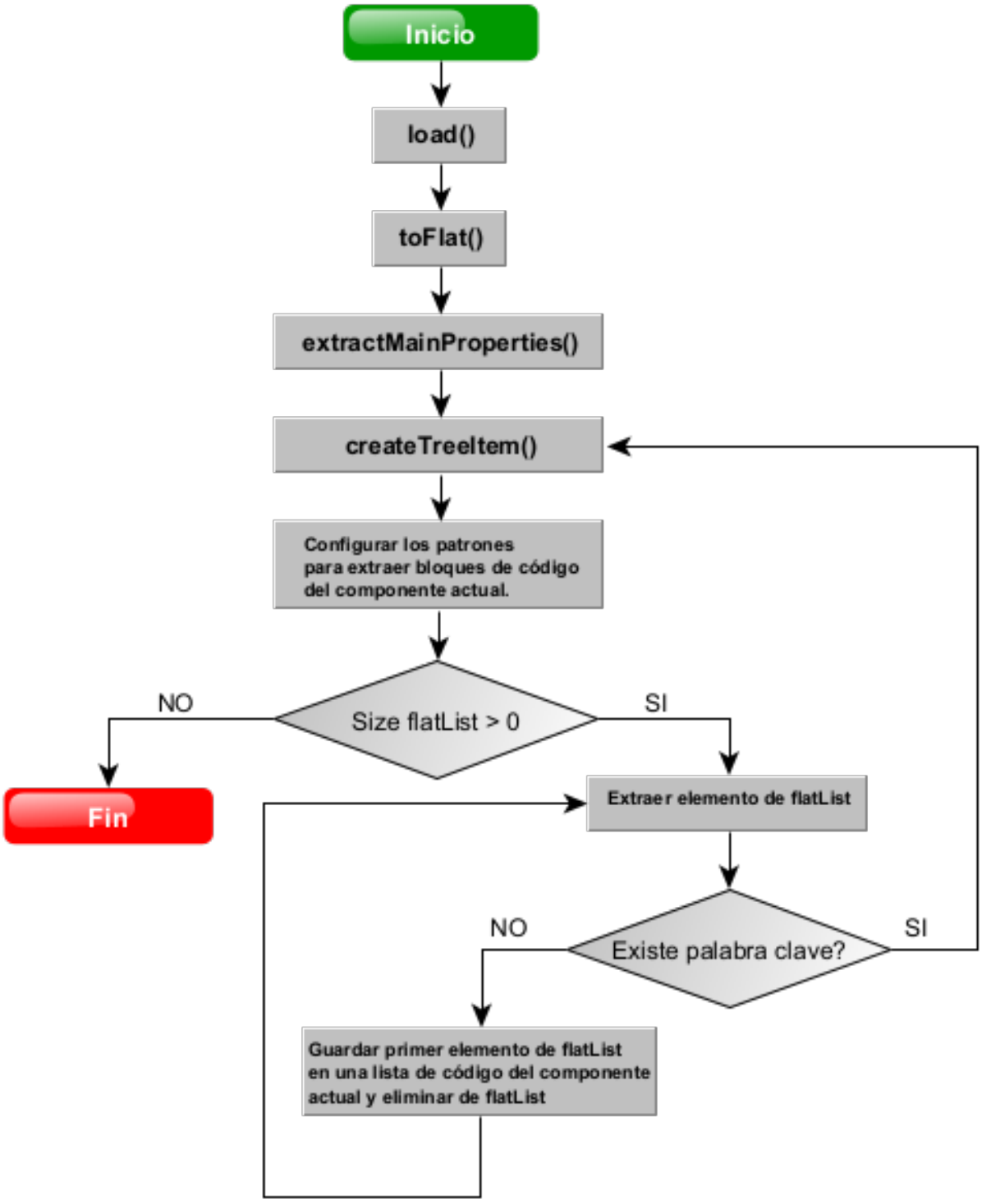

Figura 4.2: Diagrama de flujo del lector de ficheros Modelica.

4.4. Conclusiones

En este capítulo se proporciono una visión panorámica de la arquitectura de la aplicación, destacando especialmente la aplicación del patrón MVC en su estructura. Se han ofrecido detalles sobre los diversos paquetes que componen cada uno de estas capas de abstracción de esta arquitectura. La elección de segmentar en paquetes responde a la aspiración de promover la modularidad, lo cual, a su vez, fomenta tanto la reutilización de elementos como la gestión de la complejidad inherente.

Asimismo, se ha presentado un resumen del funcionamiento del lector de archivos Modelica. En el próximo capítulo, se procederá a profundizar en la descripción e implementación de las capas de abstracción del modelo MVC.

5 IMPLEMENTACIÓN

5.1. Introducción

En este capítulo se describen con detalle cada una de las implementaciones que conforman la arquitectura de la aplicación descrita en el capítulo anterior.

5.2. Configuración del proyecto

Antes de sumergirnos en la descripción de la implementación de la aplicación, vamos a abordar brevemente la configuración del entorno de desarrollo integrado (IDE) NetBeans para nuestro proyecto. Para este propósito, hemos utilizado una herramienta llamada Maven, que es un gestor de paquetes y dependencias de Java [Apache Maven, 2023]. Maven nos permite una gestión más eficiente de las dependencias de la aplicación, y para ello, debemos configurar el archivo **pom.xml**. En este archivo, agregamos cada una de las dependencias necesarias para nuestra aplicación, incluida la dependencia de JavaFX. En la Figura 5.1, se puede observar el código de configuración de este archivo. Además, en la misma figura se ha señalado cómo se ha organizado los diferentes paquetes dentro del directorio del proyecto para seguir el patrón MVC.

5.3. Implementación de los modelos

En esta sección nos centramos en la implementación de la capa de abstracción correspondiente al modelo dentro del patrón de diseño MVC. Dentro del patrón MVC, se encuentra la capa de abstracción conocida como el "Modelo", que es importante distinguir del **Modelo** en el contexto de Modelica. El primero constituye una abstracción de los datos gestionados por la aplicación, mientras que el segundo se refiere a un componente individual dentro del lenguaje Modelica. Este último está conformado por declaraciones, variables y ecuaciones. Este componente, en muchas ocasiones es representado gráficamente mediante un icono, el icono pretende describir su comportamiento a través de gráficos elementales y conectores que permiten la interacción con otros componentes (modelos).

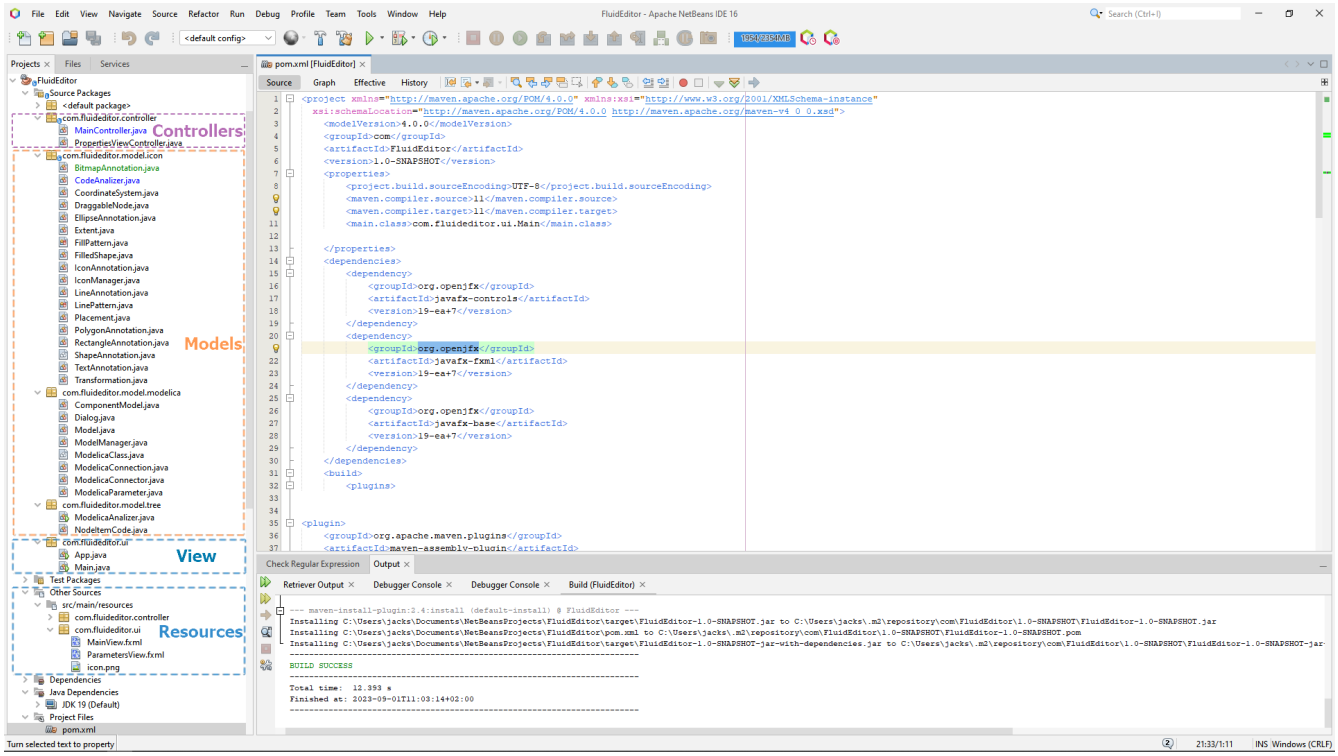

Figura 5.1: Configuración de los paquetes del patrón MVC en NetBeans para implementar FluidEditor v0.1.

5.3.1. Implementación de la librería gráfica

Los entornos gráficos de modelado que utilizan Modelica emplean iconos para representar los distintos componentes de los modelos. Siguiendo esta premisa, el objetivo consiste en desarrollar un conjunto de clases en Java que facilite la creación de estos iconos a partir de información extraída de las anotaciones del código en Modelica. Con la finalidad de lograr esta representación visual de los modelos, se ha concebido un grupo de clases que posibilitan el dibujo de iconos mediante la composición de elementos gráficos primitivos como rectángulos, elipses, líneas, textos, mapas de bits y polígonos. A este conjunto de clases que se relacionan e interaccionan entre sí para conseguir la funcionalidad comentada, se la ha denominado "librería gráfica", que su lógica dentro del patrón MVC representa a un Modelo.

Diagrama de clases

En la Figura 5.2, se presenta un diagrama de clases simple que ilustra la estructura y jerarquía de las clases que componen la librería gráfica. Se ha resaltado con un recuadro la jerarquía de las primitivas gráficas. Como clase principal o superclase, se encuentra la clase abstracta **ShapeAnnotation**, que define las propiedades y métodos comunes a las

primitivas gráficas concretas (figuras descendientes).

Las clases que representan las primitivas gráficas concretas o descendientes son las siguientes:

- **LineAnnotation:** Permite dibujar una línea. Estas líneas pueden ser elementos del icono o servir como la representación de las conexiones entre componentes Modelica, trazando una línea a partir del conector de origen hasta el conector de destino.
- **BitmapAnnotation:** Carga y muestra imágenes almacenadas en un archivo en formato de mapa de bits. Cada píxel se representa mediante un valor de color de la imagen.
- **PolygonAnnotation:** Dibuja polígonos. Un polígono puede ser cerrado si el punto final coincide con el inicial, lo que permite el relleno. Si los puntos difieren, se crea una polilínea sin relleno.
- **RectangleAnnotation:** Encargada de dibujar rectángulos.
- **EllipseAnnotation:** Dibuja elipses; si los radios son iguales, se trata de un círculo.
- **TextAnnotation:** Agrega texto sobre elementos gráficos.

Las dos primeras figuras concretas (LineAnnotation y BitmapAnnotation) carecen de una instancia de la clase **FilledShape**, ya que no admiten relleno. Sin embargo, las demás figuras cuentan con una instancia de **FilledShape** para añadir relleno mediante **FilledPattern**, así como un tipo de borde mediante **LinePattern**.

La clase **IconAnnotation** puede contener cero o varias instancias de **ShapeAnnotation**, una por cada elemento gráfico primitivo que forman el icono completo. También incorpora una instancia de **CoordinateSystem** para establecer el sistema de coordenadas, y una instancia de **Placement** para permitir transformaciones al arrastrar el icono al área de diseño, ajustando su posición con respecto al sistema de coordenadas del diagrama global.

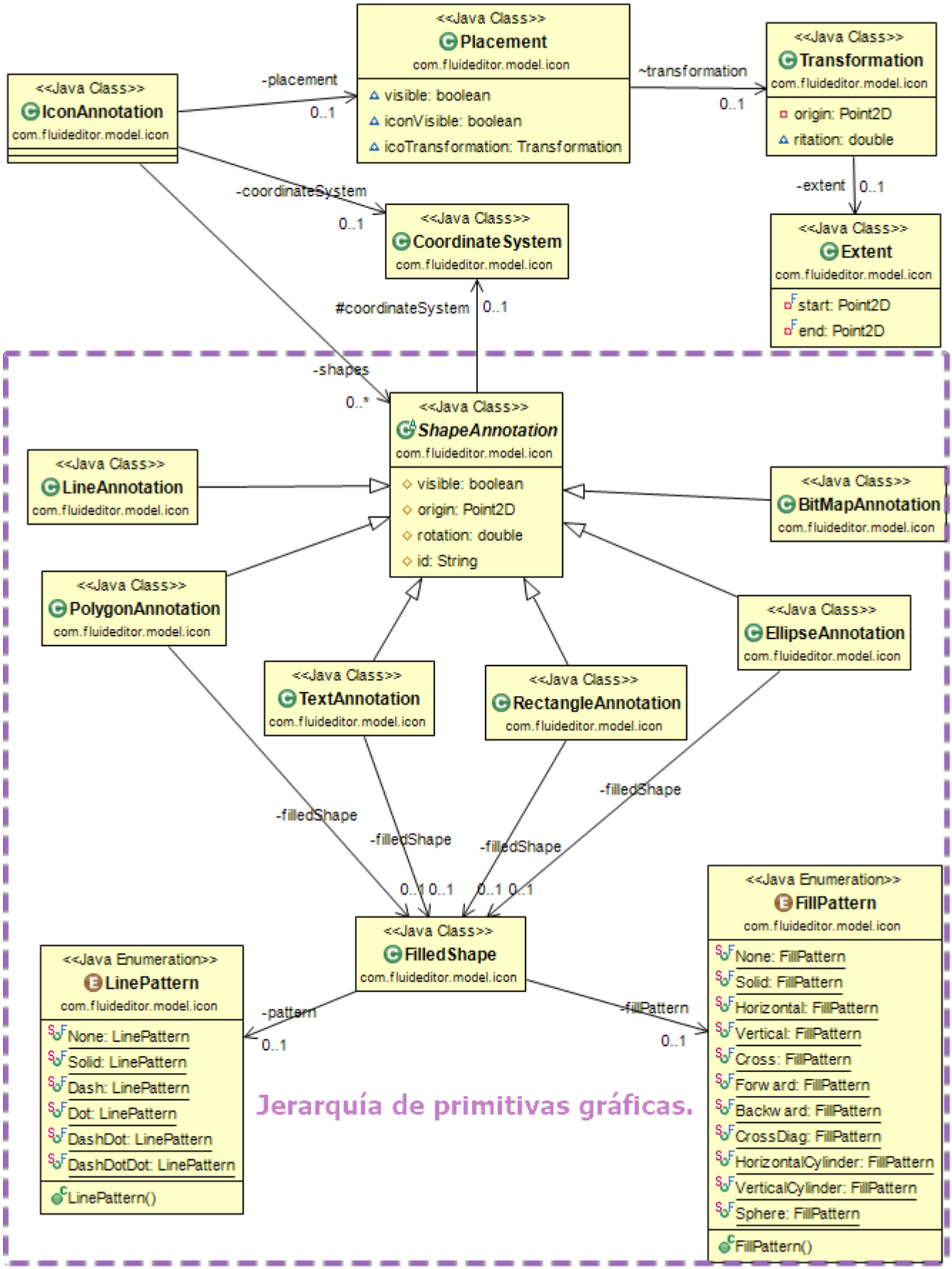

Figura 5.2: Diagrama de clases de la librería gráfica implementada en Java.

Anotaciones en Modelica

Las anotaciones en Modelica son etiquetas especiales utilizadas para proporcionar información adicional y contextos específicos a los componentes y modelos definidos en un código Modelica. Estas etiquetas no afectan directamente el comportamiento del modelo, pero enriquecen su documentación, representación gráfica y conectividad con herramientas externas. Las anotaciones se encargan de describir detalles que no pueden ser expresados únicamente a través de las declaraciones de variables y ecuaciones.

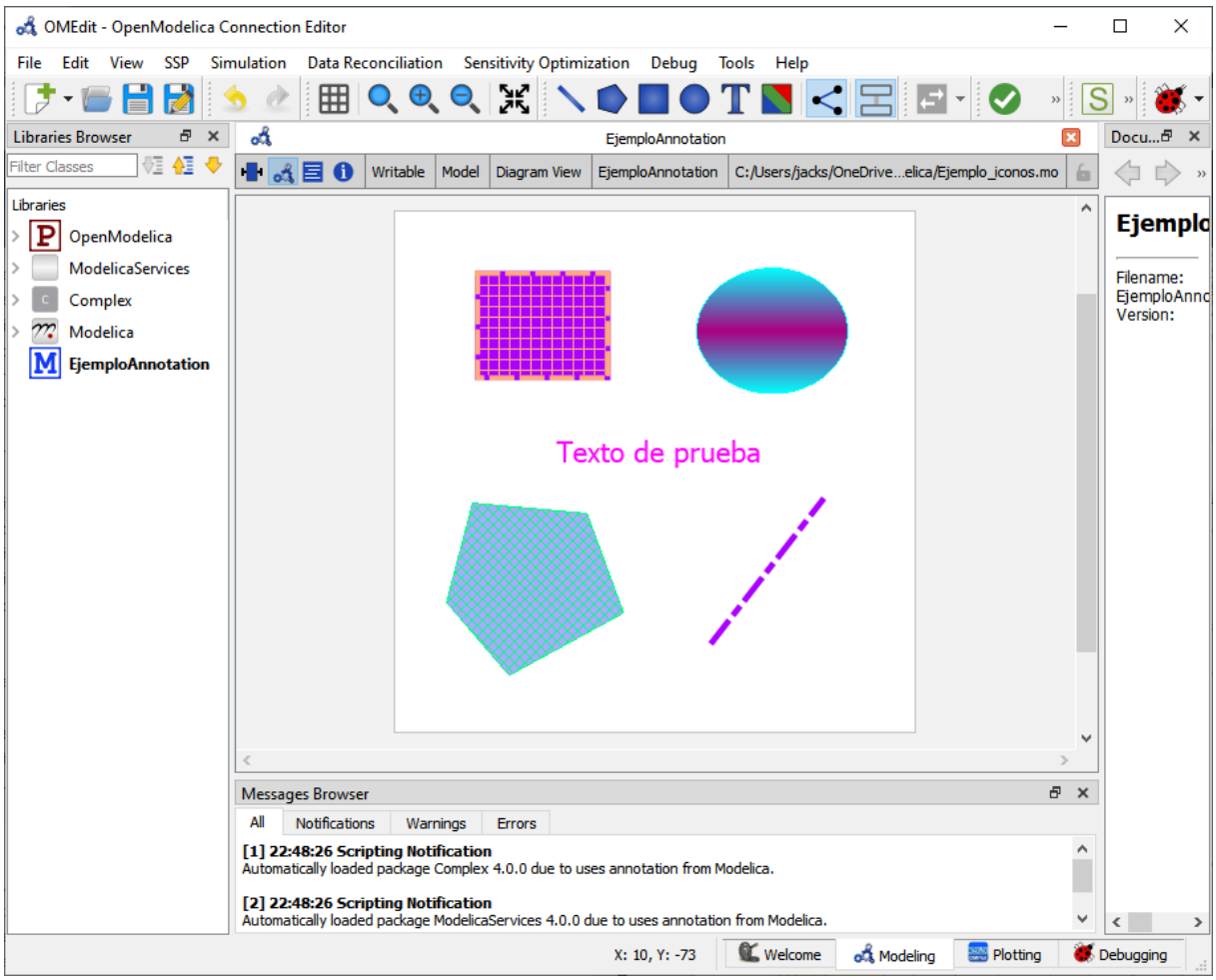

Figura 5.3: Ejemplo de gráficos primitivos en OpenModelica.

Las anotaciones pueden servir para varios propósitos, entre los que destacan:

- **Documentación:** Información descriptiva sobre el componente, su propósito, funcionamiento y cualquier otro detalle relevante para los usuarios.
- **Visualización gráfica:** Representan gráficamente los componentes en los entornos de modelado, incluye la ubicación, los conectores, la apariencia entre otros aspectos visuales.

- **Validación o comprobación:** Se pueden utilizar para realizar verificaciones y evitar errores.

En este proyecto nos interesan las anotaciones en las que se especifica el icono, cuya etiqueta es **Icon** así como las anotaciones de las conexiones. A continuación se muestran un ejemplo de cada una de estas anotaciones.

En la Figura 5.3 se muestran las principales primitivas gráficas utilizadas en los entornos de modelado, en este caso han sido generadas en el entorno OpenModelica. La anotación correspondiente a estas gráficas se muestra a continuación.

```
annotation(
    Icon(
    graphics = {
    Rectangle(origin = {-43, 56}, lineColor = {255, 170, 127}, fillColor
         = {170, 0, 255}, pattern = LinePattern.Dash, fillPattern =
        FillPattern.Cross, lineThickness = 1, extent = {{-25, 20}, {25,
        -20}}),
    Ellipse(origin = {45, 54}, lineColor = {0, 255, 255}, fillColor =
        {170, 0, 127}, fillPattern = FillPattern.HorizontalCylinder,
        extent = {{-29, 24}, {29, -24}}),
    Polygon(origin = {-43, -44}, lineColor = {0, 255, 127}, fillColor =
        {170, 170, 255}, fillPattern = FillPattern.CrossDiag, points =
        {{-27, 32}, {17, 28}, {31, -10}, {-13, -34}, {-37, -6}, {-27,
        32}, {-27, 32}}),
    Line(origin = {51.1085, -38.3625}, points = {{-28.9874, -26.7224},
        {13.0126, 27.2776}}, color = {170, 0, 255}, pattern =
        LinePattern.DashDot, thickness = 1.2),
    Text(origin = {2, 8}, textColor = {255, 0, 255}, extent = {{-42, 0},
         {42, 0}}, textString = "Texto de prueba", fontSize = 16)}));
```

Código 5.1: Ejemplo de anotaciónes Modelica para representar un icono.

En la Figura 5.4 se presenta una prueba de la funcionalidad de la librería gráfica implementada en Java. Las gráficas que se muestran han sido generadas a partir de las anotaciones mencionadas anteriormente. Como se puede observar, tanto la Figura 5.3 como la Figura 5.4 tienen una apariencia similar. La primera ha sido generada con OpenModelica y la segunda con la implementación en Java previamente comentada.

El otro tipo de anotación utilizada son las anotaciones para las conexiones entre componentes. En la Figura 5.5, se presenta un ejemplo en el que se muestran dos componentes Modelica interconectados, seguido de la anotación asociada a esas conexiones. Básicamente, esta anotación consiste en describir la línea que conecta a los dos componentes involucrados en la conexión. Es importante destacar que, en este caso, las anotaciones van junto a la etiqueta que indica la conexión, es decir, **connect**. El código Modelica correspodiente se muestra en el Código 5.2.

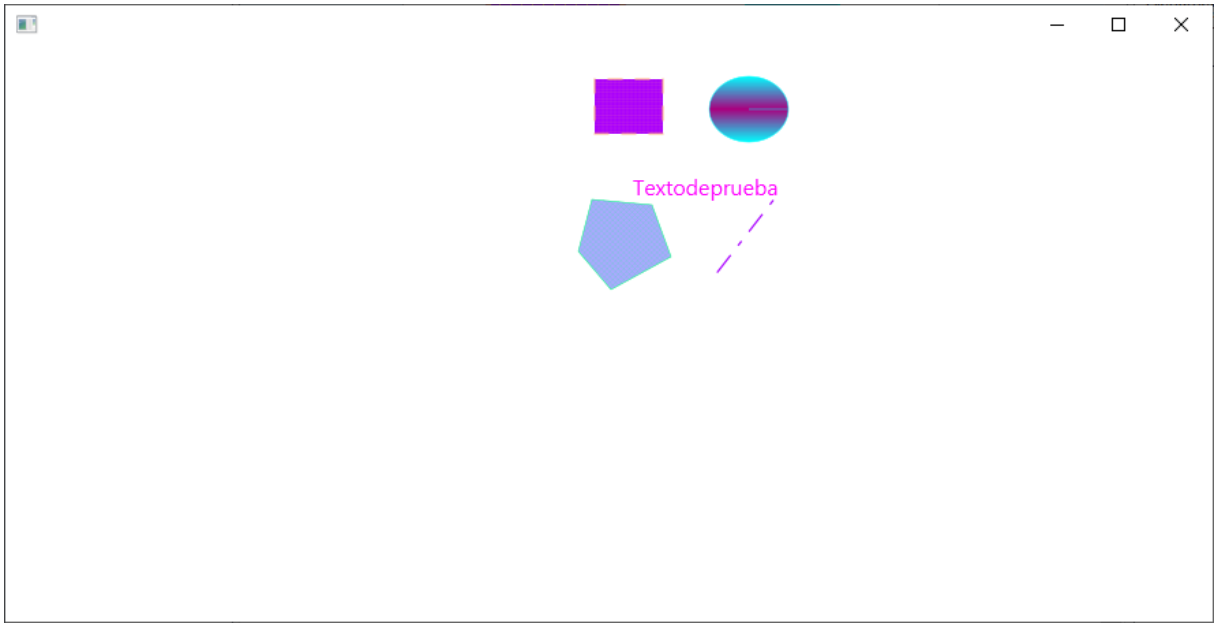

Figura 5.4: Prueba de la librería gráfica en Java utilizando anotaciones.

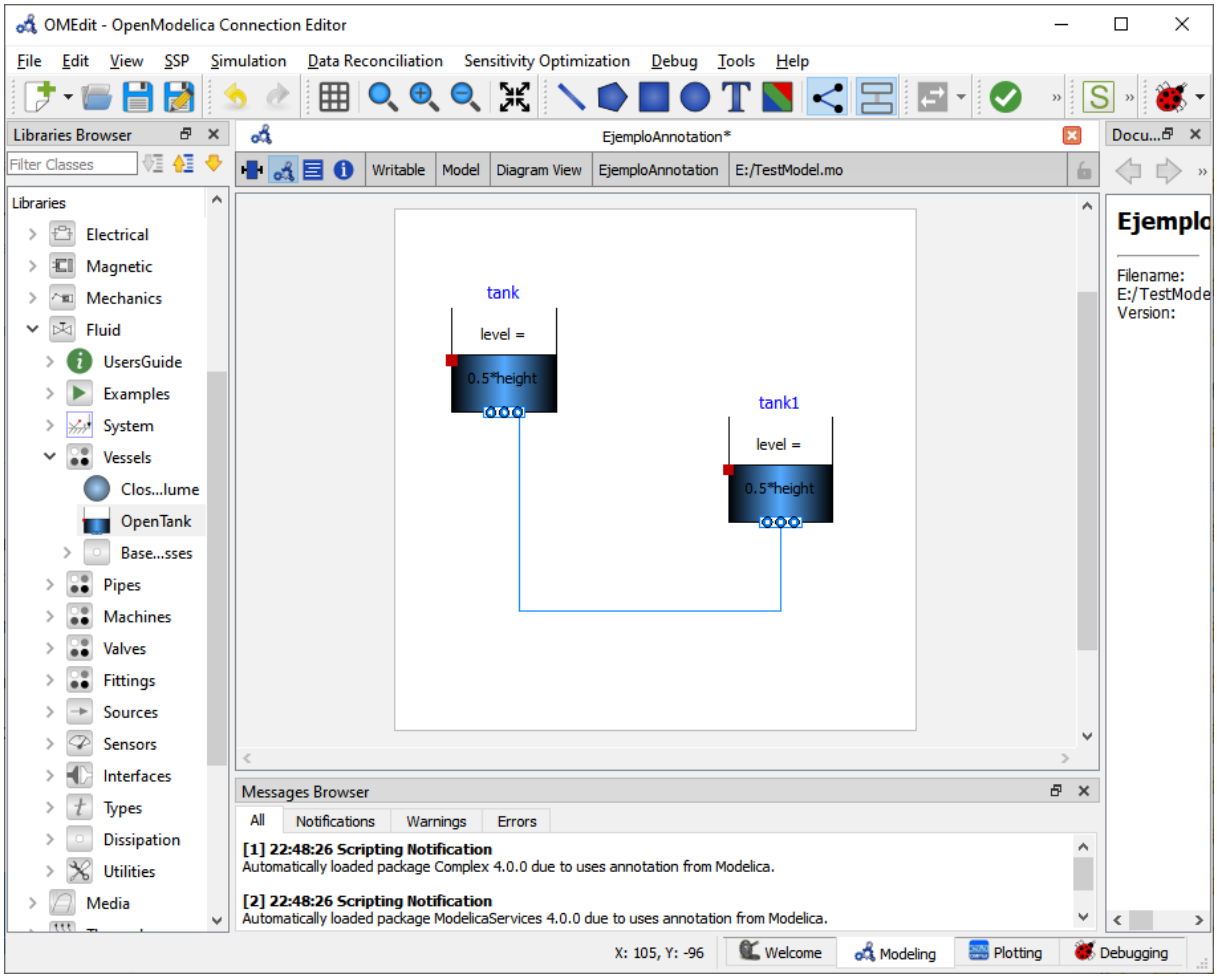

Figura 5.5: Ejemplo de conexión de dos componentes en OpenModelica.

```
connect(tank.ports[1], tank1.ports[1])
annotation(
    Line(points = {{-58, 22}, {-52, 22}, {-52, -54}, {48, -54}, {48,
        -20}}, color = {0, 127, 255}));
```

Código 5.2: Ejemplo de anotaciones Modelica para indicar una conexión entre dos componentes.

Una vez que la librería gráfica se ha implementado correctamente, el siguiente paso es crear un manejador denominado **ModelManager**. Este manejador se encargará de realizar operaciones y manipulaciones en la librería, como almacenar, ordenar, eliminar, entre otras. El propósito de este enfoque es evitar un acoplamiento directo de la librería gráfica con el controlador principal de la aplicación. En esencia, el **ModelManager** actúa como un pequeño controlador que brinda soporte al controlador principal. Puedes observar este manejador en la Figura 5.6, que corresponde al diagrama de clases completo del paquete relacionado con el Modelo del Icono.

5.3.2. Implementación del árbol de componentes

En la Sección 4.3, proporcionamos una breve descripción del proceso para extraer y obtener los modelos Modelica de los ficheros correspondientes y que se representarán en el árbol de componentes. Este procedimiento implica comenzar desde un directorio de referencia donde residen los archivos Modelica (este directorio irá junto al compilado de la aplicación y se llamará **lib**). A partir de este punto, cada archivo en el directorio se lee y sus componentes se extraen uno por uno. Para llevar a cabo este proceso, hemos implementado una clase específica encargada de esta tarea, siguiendo un algoritmo detallado en la Figura 4.2. Este diagrama de flujo ilustra de manera resumida el procedimiento paso a paso para alcanzar este objetivo de manera eficaz.

En la Figura 5.7, presentamos el diagrama de clases correspondiente a este paquete. La clase fundamental que desempeña la funcionalidad previamente mencionada es **ModelicaAnalyzer**. En esta clase se encapsula la tarea descrita anteriormente, específicamente con el método **createTreeItem**, un método recursivo que permite llevar a cabo esta funcionalidad. El código extraído de cada componente se almacena en el nodo correspondiente dentro del árbol, con el propósito de conservar tanto este código como información adicional, como el icono asociado, el nombre, la ruta, entre otra información necesaria. Para gestionar este proceso de almacenamiento y presentación, se introduce la clase auxiliar **NodeItemCode**, que sirve de contenedor para el código y los datos adicionales mencionados.

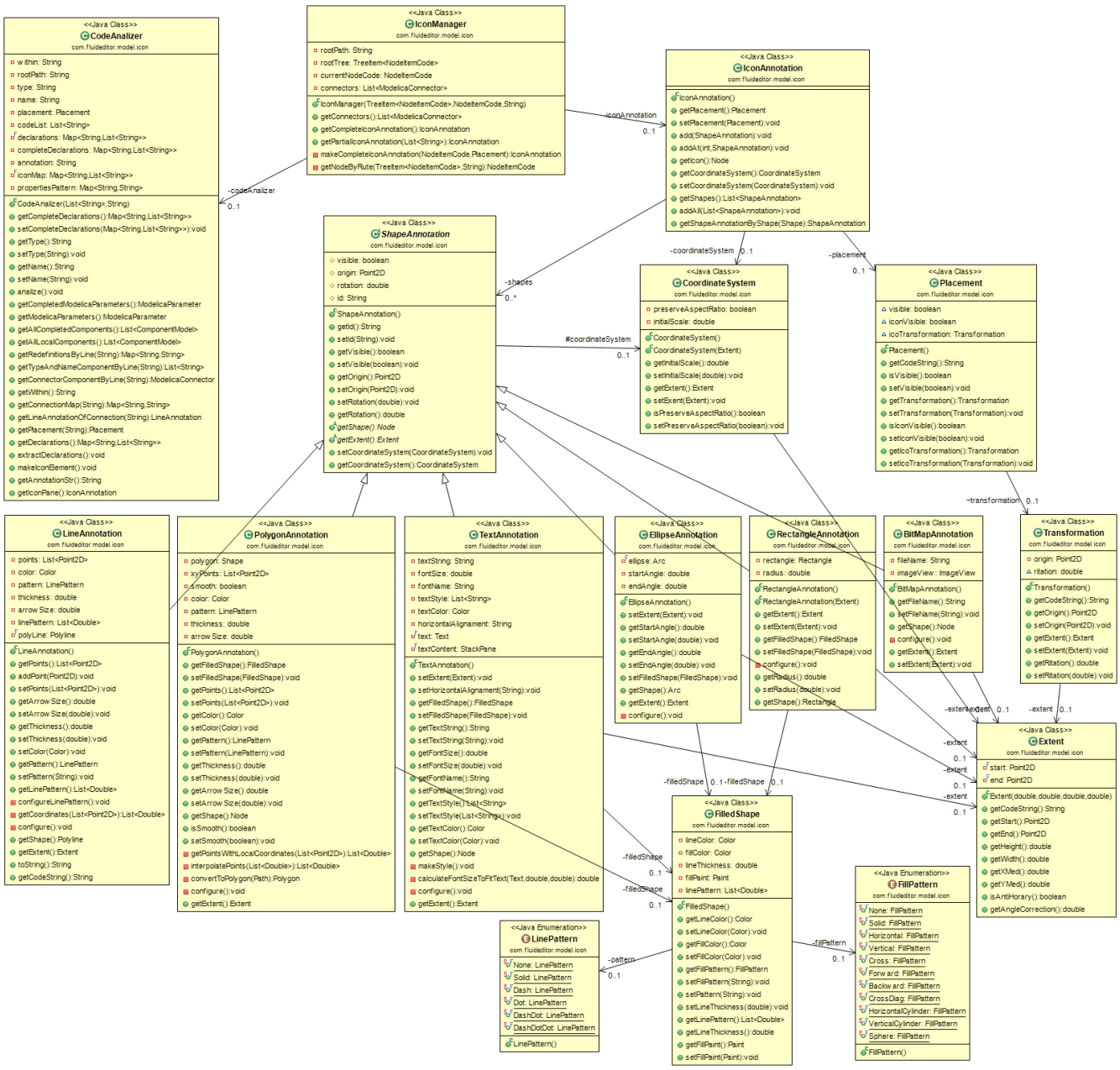

Figura 5.6: Diagrama de clases del paquete com.fluideditor.model.icon.

El control directo y la manipulación de la clase **ModelicaAnalizer** recae en el controlador. Esto se debe a que el procedimiento de lectura, extracción, generación y visualización del árbol debe llevarse a cabo antes de que la interfaz de la aplicación sea mostrada. En consecuencia, el controlador asume la responsabilidad de garantizar la ejecución exitosa de este proceso, previo a la presentación de la interfaz.

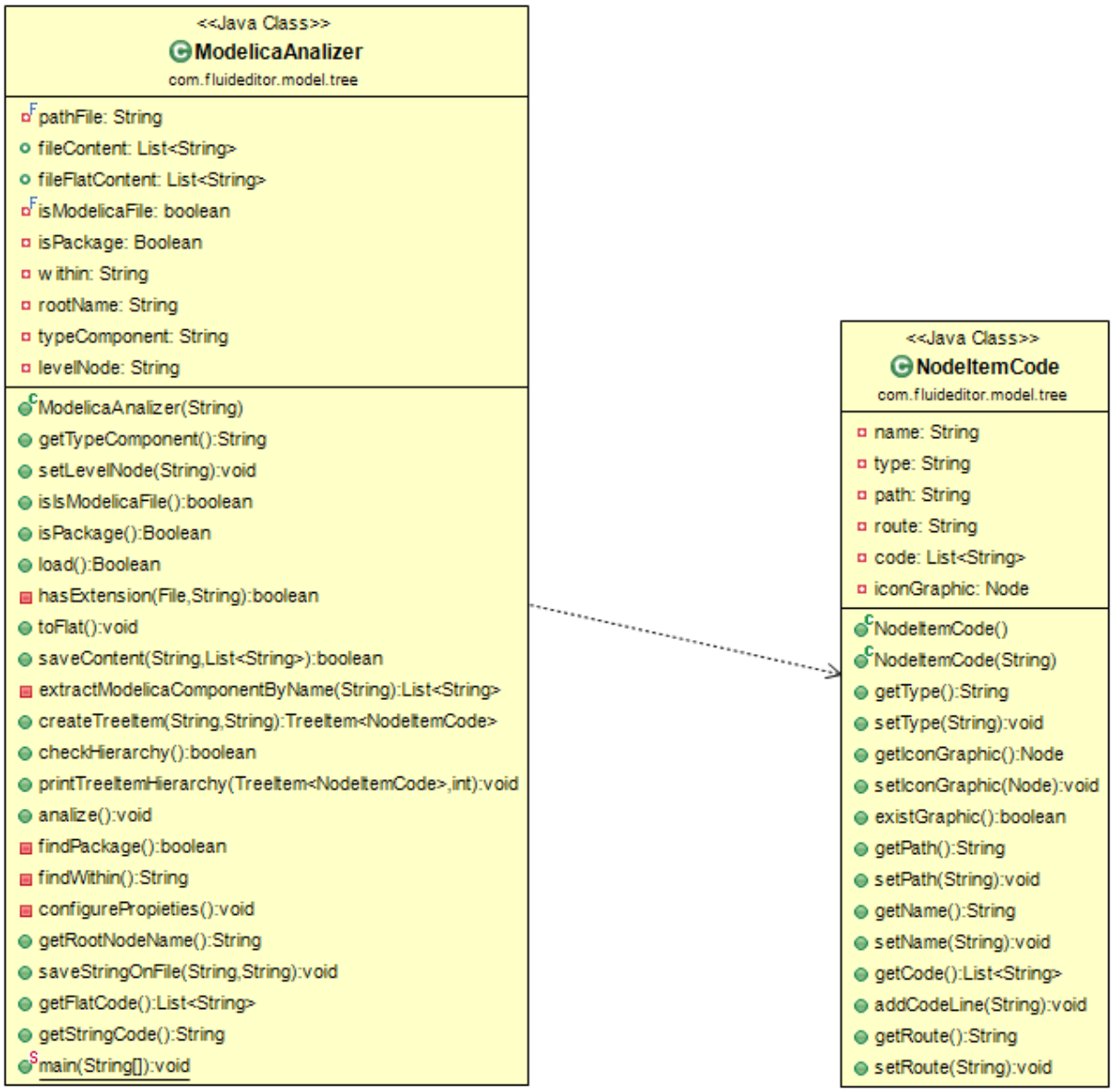

Figura 5.7: Diagrama de clases del paquete com.fluideditor.model.tree

5.3.3. Implementación de clases Modelica

En Modelica, la unidad fundamental de estructuración es la clase. Las clases proporcionan la estructura para los objetos, también conocidos como instancias. Estas clases pueden contener ecuaciones que sirven como base para el código ejecutable utilizado en los cálculos en Modelica. Todos los objetos de datos en Modelica se instancian a partir de clases, lo que incluye los tipos de datos básicos como Real, Integer, String y Boolean, así como los tipos enumerados. Las declaraciones son las construcciones sintácticas necesarias para introducir clases y objetos (componentes), y estas declaraciones se almacenan en archivos Modelica con extensión .mo.

En Modelica existen varios tipos de clase, por un lado tenemos `Class` que es la clase base abstracta, mientras que el resto de clases (`Model`, `Block`, `Package`, `Record`, etc.) son clases especializadas que contienen algunas restricciones en función de su objetivo.

Para el desarrollo de esta aplicación, se ha implementado una clase abstracta llamada **ModelicaClass**, que representa la clase abstracta Modelica `Class`. A partir de la clase abstracta **ModelicaClass**, es posible extender otras clases especializadas que representen las clases Modelica, como `Model`, `Block`, `Function`, entre otras. En esta aplicación de igual manera se ha implementando la clase Modelica `Model` para representar los modelos que se diseñan en la misma. Además, se ha dejado abierta la posibilidad de implementar otros tipos de clases especializadas Modelica sin necesidad de realizar modificaciones significativas en la aplicación. Si se desea implementar otra clase especializada, basta con heredar de **ModelicaClass** y desarrollar la lógica específica de esa clase. Esta elección se basa en uno de los principios SOLID, el principio Open/Close, que establece que una clase debe estar abierta a la extensión pero cerrada a las modificaciones. En otras palabras, se puede agregar funcionalidad sin necesidad de modificar las clases existentes, las cuales deben mantenerse inalterables [Martin, 2000].

La Figura 5.8 muestra el diagrama de clases que abarca todo el conjunto de clases internas de la aplicación destinadas a gestionar la información de los componentes Modelica, su composición y su lógica. Además de este conjunto de clases, se puede distinguir la presencia de la clase **ModelManager**, la cual asume la responsabilidad de gestionar el comportamiento de las demás clases, en calidad de gestor. Este gestor se convierte en el apoyo fundamental para el controlador principal de la aplicación, le permite interactuar con todas las clases sin tener que acoplarse a ninguna de ellas y de esta forma eliminar dependencias, promover la modularidad, la cohesión y disminuir la complejidad. También destacar que esta clase le permite al controlador principal, manipular la información necesaria de los modelos internos en cada momento mediante delegación a la misma.

La clase **Model** que hereda de **ModelicaClass** es la clase en la que se almacena la información y la lógica referente a los objetos Model de Modelica. Esta clase a su vez debe contener un conjunto de componentes, los mismos que serán los componentes que conforman el modelo, es decir, los componentes de composición del propio modelo. Cada uno de estos componentes son modelos de la librería estándar de modélica (MSL), en especifico los de la librería Fluid, que atiende a los objetivos de esta aplicación. Para tener una representación de estos componentes se ha desarrollado la clase **ModelicaParameter** que representa la información y la lógica inherente a cada componente declarado como parte de la composición del propio modelo. A su vez, esta clase tiene una relación de composición con la clase **ComponentModel** que representa las propiedades que tiene cada componente, esta clase da soporte al visualizador/editor de propiedades de componente (requisito de la aplicación). En cada entorno de modelado y simulación, la visualización/Edición de los parámetros y propiedades de un componente se muestran gracias a que obtienen la información de las notaciones declaradas el código Modélica del propio componente, estos parámetros se encuentran encerrados mediante la etiqueta **Dialog**. En este sentido, se ha implementando la clase **Dialog** que de soporte a la visualización, edición de cada uno de los parámetros obtenidos del código Modelica.

La clase **Model** también tiene una instancia de la clase **ModelicaConnection** para gestionar las conexiones entre componentes, a su vez la clase **ModelicaConnection** tiene dos instancias de la clase **ModelicaConector**, la primera llamada **firstConnector** para referirse al conector de origen, mientras que la otra se llama **secondConnector** para referirse al conector de destino. Las dos instancias juntas dan sentido a la conexión.

5.4. Implementación de la interfaz gráfica de usuario

En esta sección, abordaremos en detalle la implementación de la Interfaz Gráfica de Usuario (GUI), cuyos aspectos generales ya se describieron en capítulos previos. Una característica importante es que la interfaz está desacoplada, gracias al empleo del patrón de diseño Modelo-Vista-Controlador (MVC). Como ya se ha comentado en varias ocasiones, este enfoque conlleva la gran ventaja de permitir que el diseño sea independiente del resto de la aplicación, lo que en última instancia nos brinda una alta flexibilidad.

Con la finalidad de mantener el mismo lenguaje de programación, se ha optado por emplear Java de manera unificada tanto para la implementación del diseño de la interfaz como para las demás capas del MVC. No obstante, es válido señalar que en un contexto de desarrollo profesional, existe la posibilidad de optar por otros lenguajes según las preferencias y especialidades del equipo de desarrollo.

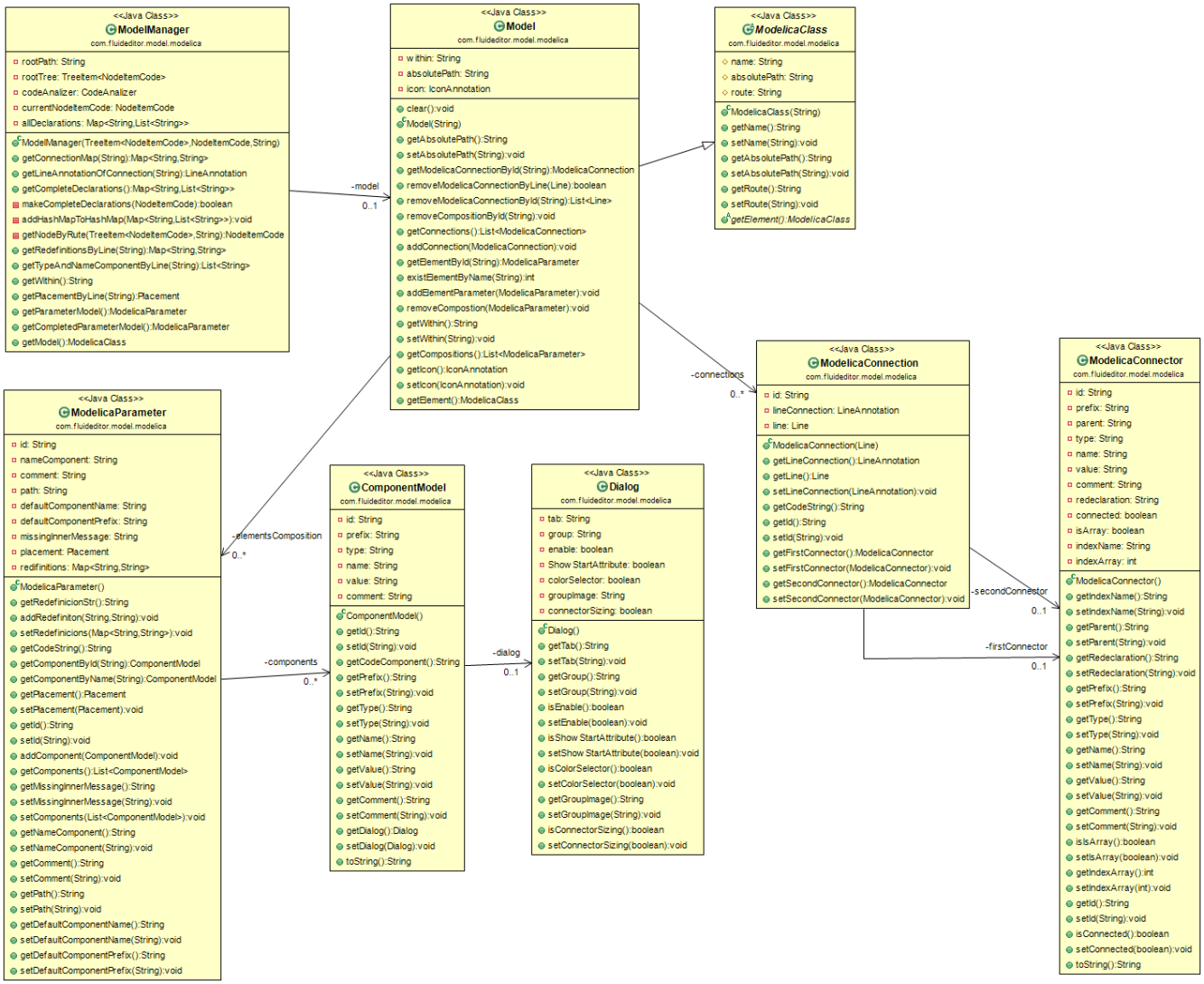

Figura 5.8: Diagrama de clases del paquete com.fluideditor.model.modelica.

Para el diseño de la interfaz nos hemos inspirado en otros entornos de modelado y simulación como pueden ser Dymola, Wolfram System Modeler, OpenModelica. Esto ayuda a que el usuario de la aplicación que ya tenga experiencia no se sienta confundido sino que aproveche dicha experiencia a la hora de utilizar la aplicación, incluso si es un usuario que por primera vez se adentra al mundo del modelado y simulación, esté adquiera experiencia aprovechable cuando tenga que utilizar otro entorno de modelado. En pocas palabras el diseño de la interfaz se divide en las siguientes partes:

- **Barra de herramientas:** En la parte superior se la aplicación se encuentra la barra de herramientas en las que podrá hacer acciones como: nuevo modelo, guardar modelo, abrir modelo. Esta barra de herramientas incluye atajos de teclado para acceder rápidamente, los atajos correspondientes se pueden observar al hacer clic en la barra de menú *File*.
- **Árbol de componentes:** El árbol de componentes se ubica en la parte izquierda, de aquí se pueden seleccionar cualquier componente y arrastrarlo al centro al área de diseño.
- **Área de diseño:** Esta área se encuentra ubicada en el centro de la aplicación, aquí es donde se arrastraran todos los componentes y se realizaran las conexiones pertinentes para componer el modelo que se este diseñando. En esta misma área, en la parte inferior derecha, podemos observar dos pestañas, que por defecto aparece seleccionada la pestaña de diseño, si presionamos en la pestaña de código nos permite conmutar a la visualización del código Modelica generado de la composición del actual modelo.
- **Barra de estado:** En esta barra se mostrará cualquier información relevante que la aplicación pueda generar. En esta aplicación de momento solo se muestra la información de la ruta donde se encuentra guardado nuestro modelo, y algún mensaje del estado del documento, por ejemplo, “unsaved”.

En la Figura 5.9 se puede observar recuadros que indican las divisiones de los bloques explicados previamente, que indican la estructura gráfica como se ha dividido la aplicación.

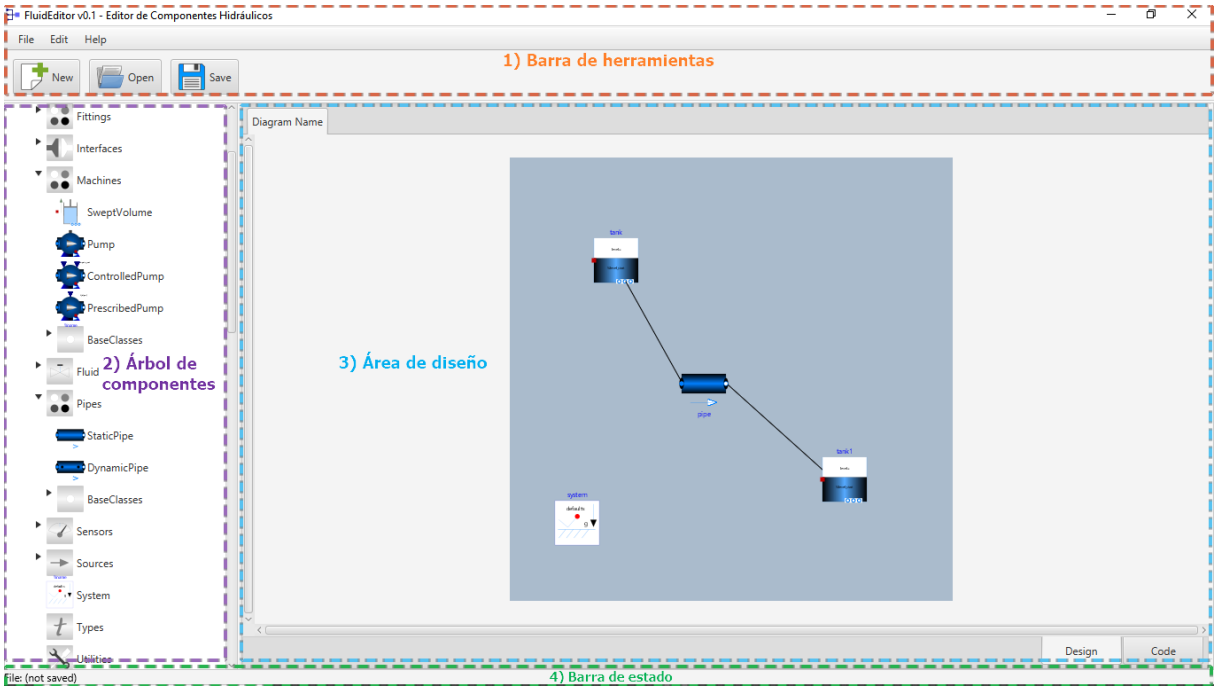

Figura 5.9: Divisiones de la interfaz de FluidEditor: 1) Barra de herramientas, 2) Árbol de componentes, 3) Área de diseño y 4) Barra de estado.

5.4.1. ¿Cómo se ha implementado la interfaz?

La implementación de la interfaz se ha llevado a cabo utilizando una herramienta de diseño que simplifica el proceso de diseño e implementación de interfaces, la herramienta **JavaFX Scene Builder** [Gluon, 2023], esta herramienta utiliza layouts, contenedores y componentes que pueden ser manipulados con la técnica "drag and drop"(arrastrar y soltar). Esta metodología evita la necesidad de escribir las tradicionales líneas de código Java, que se emplearían al trabajar con las clases propias de Java, las clases AWT, Swing o Swing+.

Para comenzar con el diseño, se inicia la herramienta y se procede a arrastrar cada uno de los contenedores, componentes al área de diseño, en donde se manipularan hasta obtener la interfaz deseada. En este caso una interfaz con los bloques comentados previamente (ver Figura 5.9). Una vez finalizado el diseño, la herramienta genera un archivo FXML que contiene la estructura de la interfaz en formato XML. Este archivo tenemos que guardarlo en el directorio del proyecto, puesto que este será el fichero que tenemos que cargar en memoria RAM al iniciar la aplicación para que se produzca la representación de nuestra interfaz. La bliblioteca JavaFX proporciona una clase que permite cargar este tipo de archivo FXML en la memoria, generando una representación de la interfaz, la misma que se mostrará en una ventana típica de Windows o del sistema operativo en la que se esta ejecutando.

Es relevante destacar que también es posible crear la interfaz directamente en el archivo FXML desde el entorno de desarrollo integrado (IDE), trabajando directamente con XML. Esto brinda flexibilidad en cuanto a la elección de enfoques en la implementación.

En las Figuras 2.14 y 2.15, se presentan respectivamente el diseño de la interfaz y el código FXML generado. Observamos que en la interfaz, se ha configurado únicamente la estructura y el esqueleto de la aplicación. Los demás elementos se cargarán de manera dinámica durante la ejecución del programa. Este enfoque permite que las interfaces diseñadas de esta manera sean manipulables tanto directamente en el archivo XML como a través de interacciones programáticas directamente desde Java.

Dentro del fichero FXML, después de que se agregan las declaraciones correspondientes a las importaciones de cada uno de los componentes utilizados (reconocidas por las etiquetas XML *import*), nos encontramos con las etiquetas XML que describen cada uno de los componentes que constituyen la interfaz, la primera de ellas es `StackPane`, cuyo código es similar al que se muestra a continuación:

```
<StackPane fx:controller="com.fluideditor.controller.MainController"
maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="760.0" prefWidth="1024.0"
xmlns="http://javafx.com/javafx/19"
xmlns:fx="http://javafx.com/fxml/1" >
```

Código 5.3: Código FXML parcial de la Interfaz Gráfica de Usuario (GUI).

Dentro de esta etiqueta se especifican propiedades, siendo una de ellas `fx:controller="com.fluideditor.controller.MainController"`. Esta propiedad adquiere gran relevancia ya que define la ubicación de la clase Java que implementa el controlador de esta interfaz gráfica. En este contexto, la clase designada es **MainController**, situada en el paquete **com.fluideditor.controller**. En la siguiente sección se detallará como se implementa este controlador.

Para cargar el fichero FXML que contiene la interfaz gráfica y de esta forma mostrar la visualización de la misma en una ventana de Windows, se ha implementado un método en la clase principal de la aplicación, la clase **App**, este método tiene un código similar al que se muestra en el Código 5.4.

```java
/** Método que permite cargar los ficheros fxml que contienen la
  * descripción de la GUI
  * @param fxml Contiene el nombre del fichero sin extensión.
  * @return FXMLLoader Devuelve objetos manejables por Java.
  * @throws IOException Excepciones durante la lectura del fichero.
  **/
private static FXMLLoader loadFXML(String fxml) throws IOException {
    FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(fxml +
        ".fxml"));
    return fxmlLoader;
}
```

Código 5.4: Método para cargar el fichero FXML que permite visualizar la Interfaz Gráfica de Usuario (GUI).

Este método recibe por parámetro el nombre del archivo FXML sin la extensión. Es fundamental que este nombre sea coherente con el nombre de algún archivo ubicado dentro del directorio de recursos de la aplicación, de lo contrario, se lanzará una excepción. Para acceder a la ruta de los recursos, se emplea el método estático característico de cada clase: `App.class.getResource()`. Para obtener una comprensión más detallada de este proceso, se recomienda revisar el código disponible en los anexos de esta memoria.

5.5. Implementación de los controladores

En esta sección, nos enfocamos en los controladores de la aplicación, en este caso solo tenemos dos controladores, el controlador principal y un controlador que gestiona todo lo relacionado con los parámetros de cada uno de las instancias de componentes Modelica que se añadan a la hora de crear un modelo Modelica en la área de diseño de la aplicación. Empezamos por el controlador principal, el cual entre las tareas que tiene que abordar podemos mencionar las siguientes:

- **Gestión de eventos del usuario:** El controlador responde a los eventos generados desde la interfaz gráfica, tales como clic de ratón, combinaciones de teclas, entre otros.
- **Carga del árbol de componentes:** Cada componente del árbol de componentes se extrae a partir de la información contenida en los archivos Modelica ubicados en el directorio ***/lib/Modelica***. Este directorio se sitúa al mismo nivel que el ejecutable de la aplicación (el archivo .jar).
- **Gestión del diseño del modelo:** El controlador asume la responsabilidad de administrar el modelo en proceso de diseño. Esto implica supervisar los componentes

arrastrados al área de diseño y gestionar las conexiones entre ellos, así como el movimiento, la selección y la eliminación de los mismos.

- **Generación de código Modelica:** El controlador realiza la tarea de traducir cada elemento del diseño, junto con sus conexiones en código Modelica.
- **Eliminación, guardado y carga de modelos:** Este rol comprende la eliminación coherente de modelos, el guardado preciso de los mismos, así como la carga adecuada de modelos, trasladando el código Modelica almacenado en archivos a su representación gráfica en la interfaz en el área de diseño, con sus conexiones correspondientes, de igual forma generando el código Modelica de manera de correcta.

En la Figura 5.10 se presenta el diagrama de clases correspondiente al controlador principal. Se puede observar que el controlador se apoya en la utilización de cuatro componentes clave: **ModelicaManager**, **IconManager**, **ModelicaAnalyzer** y **PropertiesViewController**.

La primera clase, **ModelicaManager**, desempeña un papel esencial en la gestión global del diseño del modelo Modelica (consulte la sección 5.3.3 para una explicación detallada sobre este tema).

La segunda clase, **IconManager**, está dedicada a la creación, administración y manipulación de iconos que representan cada uno de los componentes Modelica del diseño.

La tercera clase, **ModelicaAnalyzer**, encargada de leer los archivos Modelica, extrayendo la información de estos archivos para generar el árbol de componentes y su correspondiente código asociado.

Por último, **PropertiesViewController** es el controlador encargado de las gestión de los parámetros de cada uno de los componentes. Permitiendo la visualización, la modificación y actualización de los mismos.

La colaboración de estas cuatro clases dentro del controlador permite desarrollar las tareas encargadas al mismo.

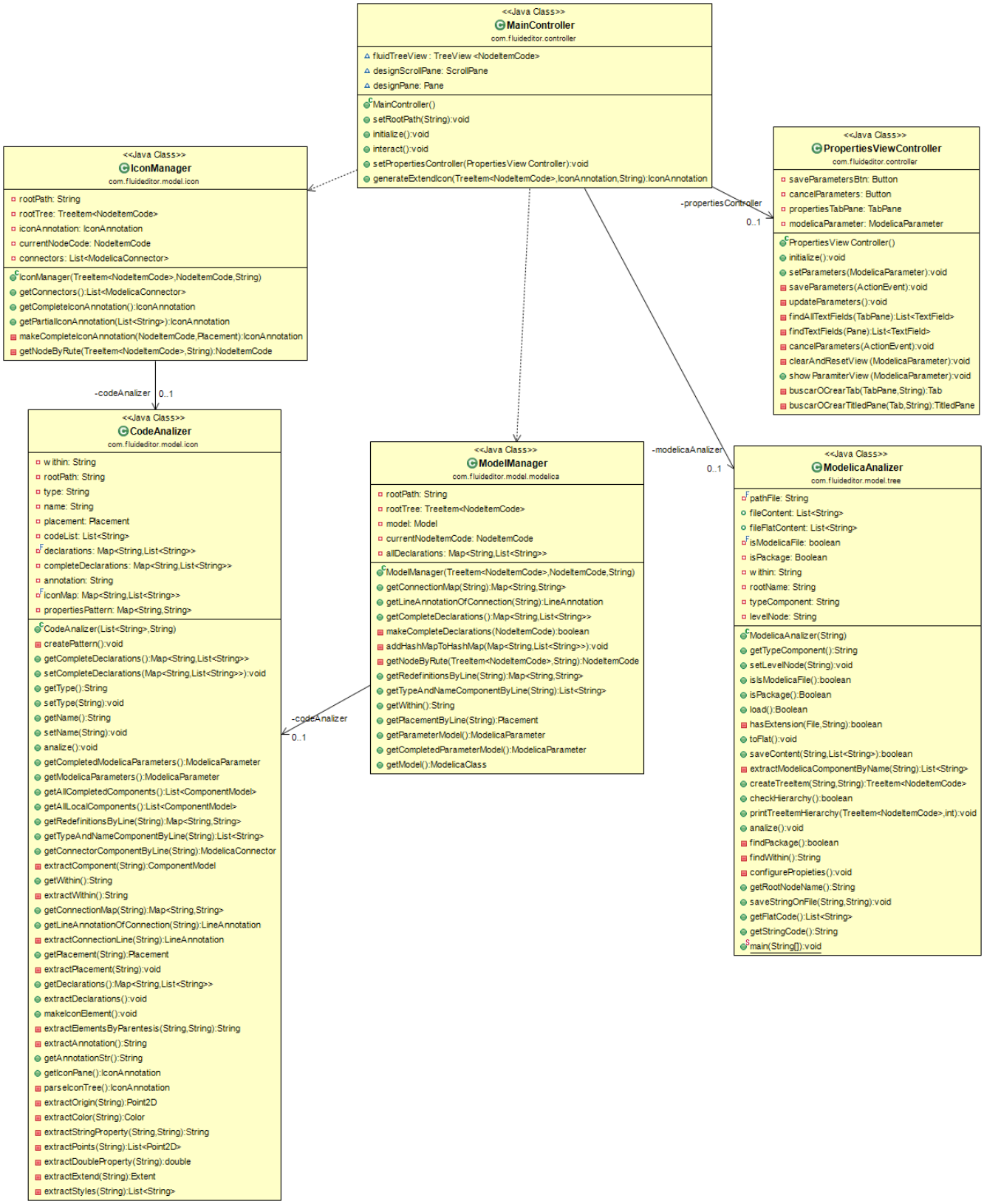

Figura 5.10: Diagrama de clases del controlador de la aplicación.

5.6. Conclusiones

En este capítulo, hemos explicado la implementación de cada uno de los componentes que forman parte de la aplicación, siguiendo la descomposición del patrón MVC. Para cada descomposición, hemos presentado el diagrama de clases correspondiente para facilitar la comprensión de las explicaciones.

Para una comprensión más completa de esta implementación, recomendamos revisar el código adjunto en los anexos de esta memoria. El código desarrollado se ha enfocado en la legibilidad y la claridad, utilizando nombres descriptivos para los métodos y propiedades, de esta manera permite comprender fácilmente la funcionalidad al leer sus propios nombres. Estos principios se han aplicado siguiendo las recomendaciones del libro "Código Limpio" [Martin, 2012], lo que ha contribuido a la calidad del código implementado.

6 PRUEBAS

6.1. Introducción

Este capítulo se centra en la realización de pruebas de la aplicación **FluidEditor v0.1** para verificar su funcionamiento adecuado. Para lograrlo, se crearán varios modelos, se examinará el código generado y se realizarán comprobaciones en entornos de modelado como OpenModelica o Wolfram System Modeler. Esto permitirá conocer y garantizar que el código generado por la aplicación sea correcto y funcional, o por lo contrario detectar fallos a corregir.

6.2. Prueba de visualización gráfica

Con el propósito de verificar la correcta funcionalidad de la selección, arrastre y soltado de componentes, así como la representación gráfica precisa en el área de diseño, procedemos a arrastrar una serie de componentes desde el árbol de componentes ubicado en la parte izquierda de la aplicación hasta el área central de diseño. Tras arrastrar varios componentes de ejemplo, obtenemos el resultado mostrado en la Figura 6.1.

El código Modelica generado por FluidEditor se muestra en la Figura 6.2, este mismo código se muestra en texto plano en Código 6.1. Tanto los iconos como el código generado parecen correctos a simple vista. Sin embargo, la verdadera comprobación se realiza al trasladar este código a otros entornos de simulación Modelica. Hemos cargado el código tanto en OpenModelica como en Wolfram System Modeler, y los resultados se muestran en las Figuras 6.3 y 6.4 respectivamente. En ambos casos, la ubicación y apariencia de los componentes son similares. En el caso de Wolfram System Modeler, la apariencia puede variar ligeramente debido a las personalizaciones del entorno. Lo más importante es que los iconos representen correctamente los modelos, estén ubicados adecuadamente y el código generado sea preciso en su representación.

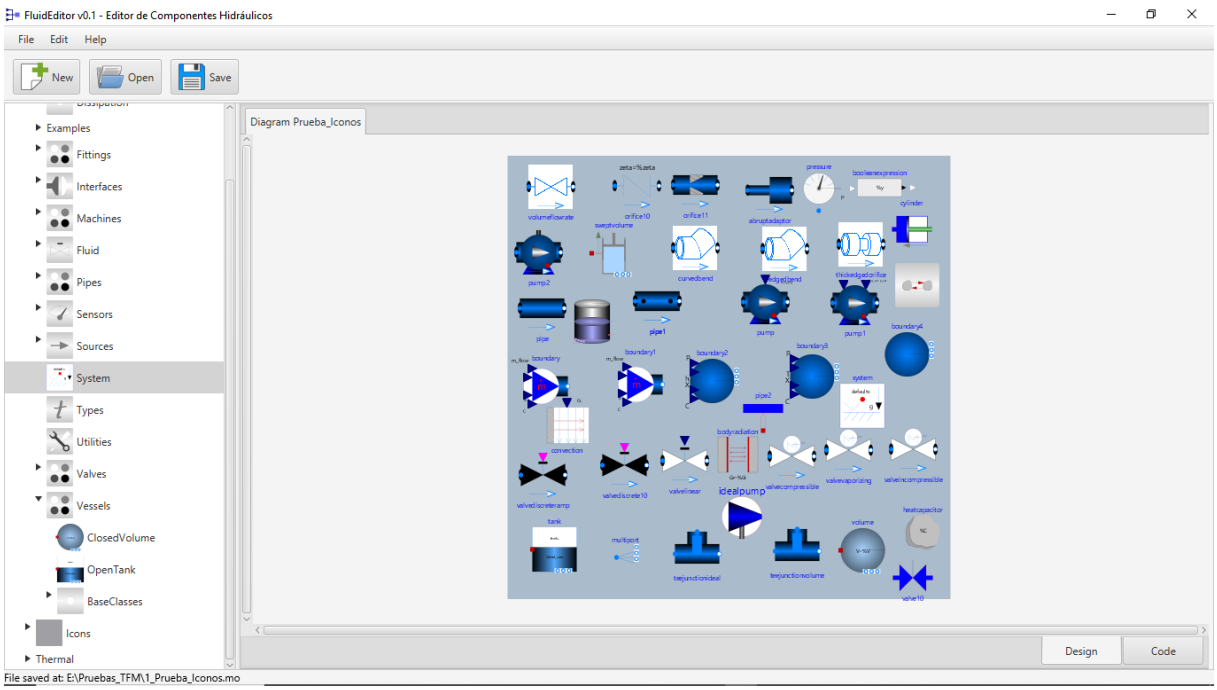

Figura 6.1: Verificación de la correcta representación de los iconos de los componentes Modelica en FluidEditor.

```
1  model Prueba_Iconos
2     Modelica.Fluid.Vessels.OpenTank tank annotation(Placement(visible=
          true,transformation(origin={-78,-78.4},extent
          ={{-10.0,-10.0},{10.0,10.0}})));
3     Modelica.Fluid.Vessels.ClosedVolume volume annotation(Placement(
          visible=true,transformation(origin={61.2,-78.8},extent
          ={{-10.0,-10.0},{10.0,10.0}})));
4     Modelica.Fluid.Valves.ValveDiscreteRamp valvediscreteramp annotation(
          Placement(visible=true,transformation(origin={-83.2,-44.8},extent
          ={{-10.0,-10.0},{10.0,10.0}})));
5     Modelica.Fluid.Valves.ValveDiscrete valvediscrete10 annotation(
          Placement(visible=true,transformation(origin={-46.4,-40.4},extent
          ={{-10.0,-10.0},{10.0,10.0}})));
6     Modelica.Fluid.Valves.ValveLinear valvelinear annotation(Placement(
          visible=true,transformation(origin={-19.2,-38.4},extent
          ={{-10.0,-10.0},{10.0,10.0}})));
7     Modelica.Fluid.Valves.ValveCompressible valvecompressible annotation(
          Placement(visible=true,transformation(origin={29.2,-36.4},extent
          ={{-10.0,-10.0},{10.0,10.0}})));
8     Modelica.Fluid.Valves.ValveVaporizing valvevaporizing annotation(
          Placement(visible=true,transformation(origin={54.8,-33.6},extent
          ={{-10.0,-10.0},{10.0,10.0}})));
9     Modelica.Fluid.Valves.ValveIncompressible valveincompressible
          annotation(Placement(visible=true,transformation(origin
          ={84,-33.2},extent={{-10.0,-10.0},{10.0,10.0}})));
10    Modelica.Fluid.Sources.MassFlowSource_h boundary annotation(Placement
          (visible=true,transformation(origin={-82,-4.8},extent
          ={{-10.0,-10.0},{10.0,10.0}})));
11    Modelica.Fluid.Sources.MassFlowSource_T boundary1 annotation(
          Placement(visible=true,transformation(origin={-39.2,-4},extent
          ={{-10.0,-10.0},{10.0,10.0}})));
12    Modelica.Fluid.Sources.Boundary_ph boundary2 annotation(Placement(
          visible=true,transformation(origin={-6.8,-2.4},extent
          ={{-10.0,-10.0},{10.0,10.0}})));
13    Modelica.Fluid.Sources.Boundary_pT boundary3 annotation(Placement(
```

```
        visible=true,transformation(origin={38.4,-0.4},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
14    Modelica.Fluid.Sources.FixedBoundary boundary4 annotation(Placement(
        visible=true,transformation(origin={81.2,9.6},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
15    Modelica.Fluid.Pipes.StaticPipe pipe annotation(Placement(visible=
        true,transformation(origin={-83.6,30.4},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
16    Modelica.Fluid.Pipes.DynamicPipe pipe1 annotation(Placement(visible=
        true,transformation(origin={-31.6,33.6},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
17    Modelica.Fluid.Machines.PrescribedPump pump annotation(Placement(
        visible=true,transformation(origin={17.2,33.2},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
18    Modelica.Fluid.Machines.ControlledPump pump1 annotation(Placement(
        visible=true,transformation(origin={57.6,32.8},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
19    Modelica.Fluid.Machines.Pump pump2 annotation(Placement(visible=true,
        transformation(origin={-85.2,55.6},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
20    Modelica.Fluid.Machines.SweptVolume sweptvolume annotation(Placement(
        visible=true,transformation(origin={-51.2,55.2},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
21    Modelica.Fluid.Fittings.Bends.CurvedBend curvedbend annotation(
        Placement(visible=true,transformation(origin={-14.4,57.6},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
22    Modelica.Fluid.Fittings.Bends.EdgedBend edgedbend annotation(
        Placement(visible=true,transformation(origin={26,57.2},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
23    Modelica.Fluid.Fittings.Orifices.ThickEdgedOrifice thickedgedorifice
        annotation(Placement(visible=true,transformation(origin
        ={60.4,59.2},extent={{-10.0,-10.0},{10.0,10.0}})));
24    Modelica.Fluid.Fittings.GenericResistances.VolumeFlowRate
        volumeflowrate annotation(Placement(visible=true,transformation(
        origin={-79.6,85.2},extent={{-10.0,-10.0},{10.0,10.0}})));
25    Modelica.Fluid.Fittings.SimpleGenericOrifice orifice10 annotation(
        Placement(visible=true,transformation(origin={-40.8,85.6},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
26    Modelica.Fluid.Fittings.SharpEdgedOrifice orifice11 annotation(
        Placement(visible=true,transformation(origin={-14.4,86},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
27    Modelica.Fluid.Fittings.AbruptAdaptor abruptadaptor annotation(
        Placement(visible=true,transformation(origin={19.2,83.6},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
28    Modelica.Fluid.Fittings.MultiPort multiport annotation(Placement(
        visible=true,transformation(origin={-46,-82},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
29    Modelica.Fluid.Fittings.TeeJunctionIdeal teejunctionideal annotation(
        Placement(visible=true,transformation(origin={-13.6,-80.4},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
30    Modelica.Fluid.Fittings.TeeJunctionVolume teejunctionvolume
        annotation(Placement(visible=true,transformation(origin
        ={31.6,-79.2},extent={{-10.0,-10.0},{10.0,10.0}})));
31    Modelica.Blocks.Sources.BooleanExpression booleanexpression
        annotation(Placement(visible=true,transformation(origin
        ={68.8,84.8},extent={{-10.0,-10.0},{10.0,10.0}})));
32    Modelica.Thermal.HeatTransfer.Components.BodyRadiation bodyradiation
        annotation(Placement(visible=true,transformation(origin
        ={4.8,-35.6},extent={{-10.0,-10.0},{10.0,10.0}})));
33    Modelica.Thermal.HeatTransfer.Components.Convection convection
        annotation(Placement(visible=true,transformation(origin
```

```
        ={-72.4,-22.4},extent={{-10.0,-10.0},{10.0,10.0}})));
34    Modelica.Thermal.HeatTransfer.Components.HeatCapacitor heatcapacitor
        annotation(Placement(visible=true,transformation(origin
        ={88.4,-69.2},extent={{-10.0,-10.0},{10.0,10.0}})));
35    Modelica.Thermal.HeatTransfer.HeatTransfer HeatTransfer annotation(
        Placement(visible=true,transformation(origin={85.8,40.8},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
36    Modelica.Thermal.FluidHeatFlow.Sources.IdealPump idealpump annotation
        (Placement(visible=true,transformation(origin={6.6,-63.6},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
37    Modelica.Thermal.FluidHeatFlow.Components.OpenTank OpenTank
        annotation(Placement(visible=true,transformation(origin
        ={-61,24.4},extent={{-10.0,-10.0},{10.0,10.0}})));
38    Modelica.Thermal.FluidHeatFlow.Components.Pipe pipe2 annotation(
        Placement(visible=true,transformation(origin={16.2,-14.8},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
39    Modelica.Thermal.FluidHeatFlow.Components.Cylinder cylinder
        annotation(Placement(visible=true,transformation(origin
        ={83.4,66},extent={{-10.0,-10.0},{10.0,10.0}})));
40    Modelica.Thermal.FluidHeatFlow.Components.Valve valve10 annotation(
        Placement(visible=true,transformation(origin={83.8,-91.2},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
41    Modelica.Fluid.Sensors.Pressure pressure annotation(Placement(visible
        =true,transformation(origin={41.4,84.4},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
42  inner Modelica.Fluid.System system annotation(Placement(visible=true,
        transformation(origin={61,-13.6},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
43  equation
44  end Prueba_Iconos;
```

Código 6.1: Código Modelica generado en FluidEditor: visualización de iconos.

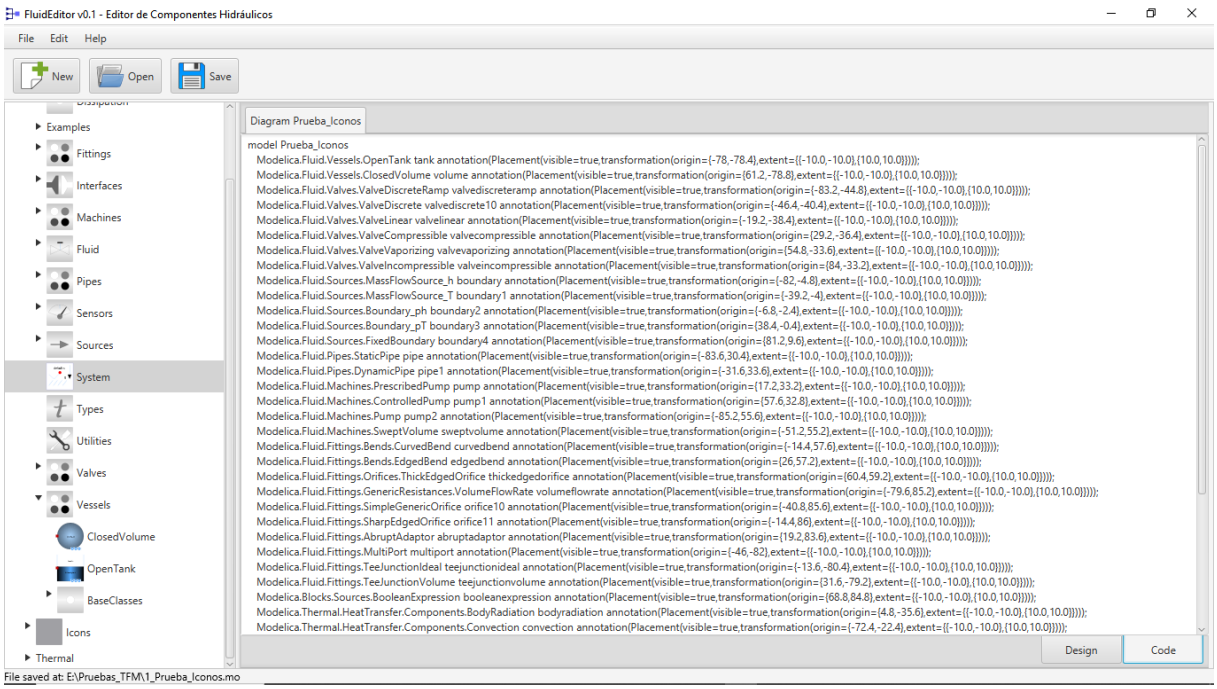

Figura 6.2: Código Modelica generado por FluidEditor para representar los iconos de la Figura 6.1.

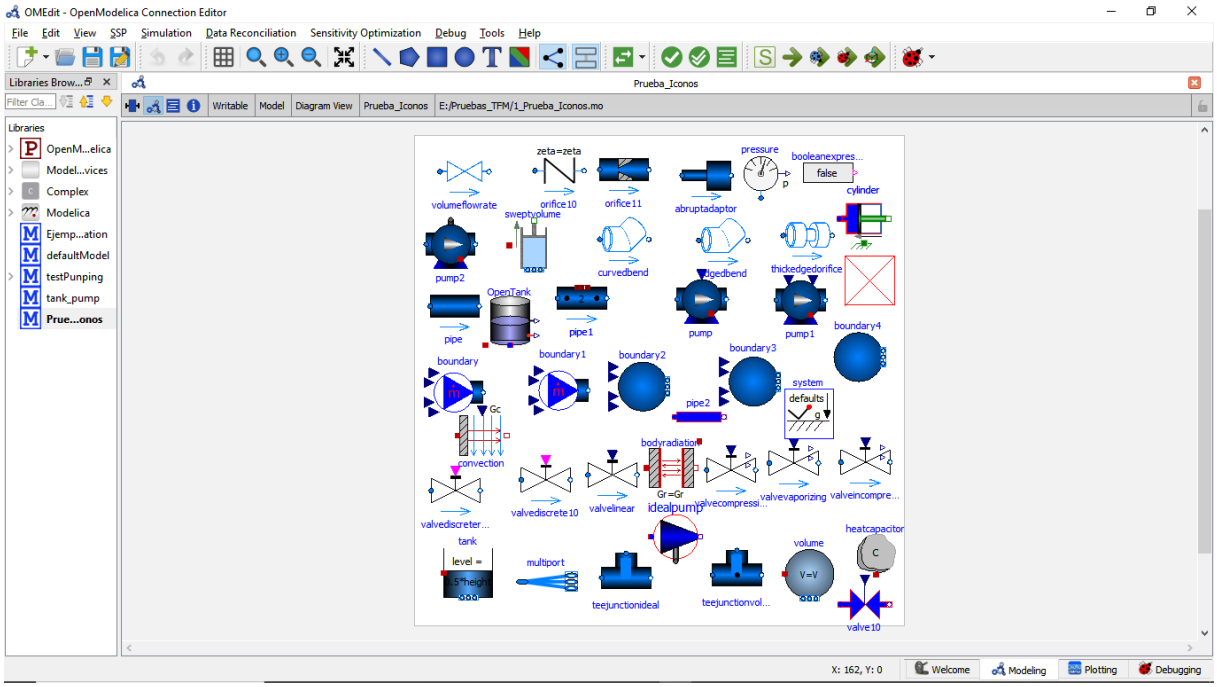

Figura 6.3: Visualización de los iconos de prueba en OpenModelica.

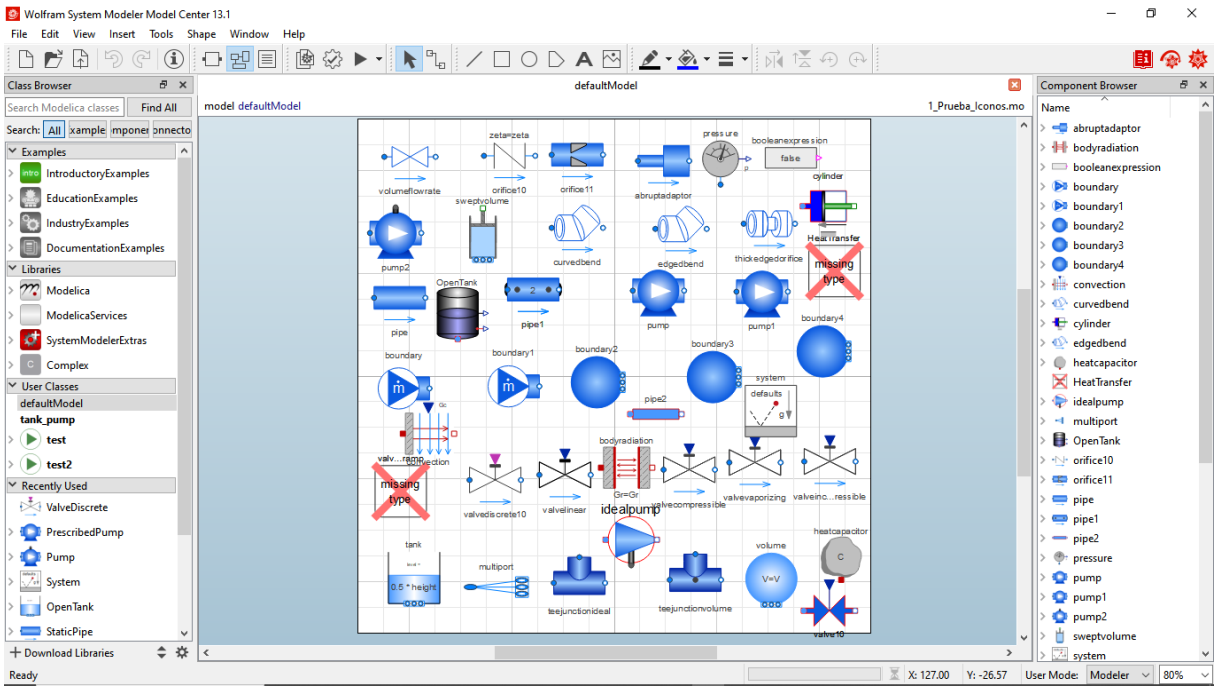

Figura 6.4: Visualización de los iconos de prueba en Wolfram System Modeler.

6.3. Primer modelo de prueba: *EmptyTanks*

Como primer ejemplo que permita comprobar la correcta generación de código funcional en FluidEditor, vamos a utilizar un ejemplo que se encuentra integrado en el propio paquete **Fluid** de la librería estándar Modelica, el ejemplo está ubicado en *Modelica.Icons.Example.Tanks.EmptyTanks* de las librerías cargadas en OpenModelica. El diagrama de este ejemplo se muestra en la Figura 6.5.

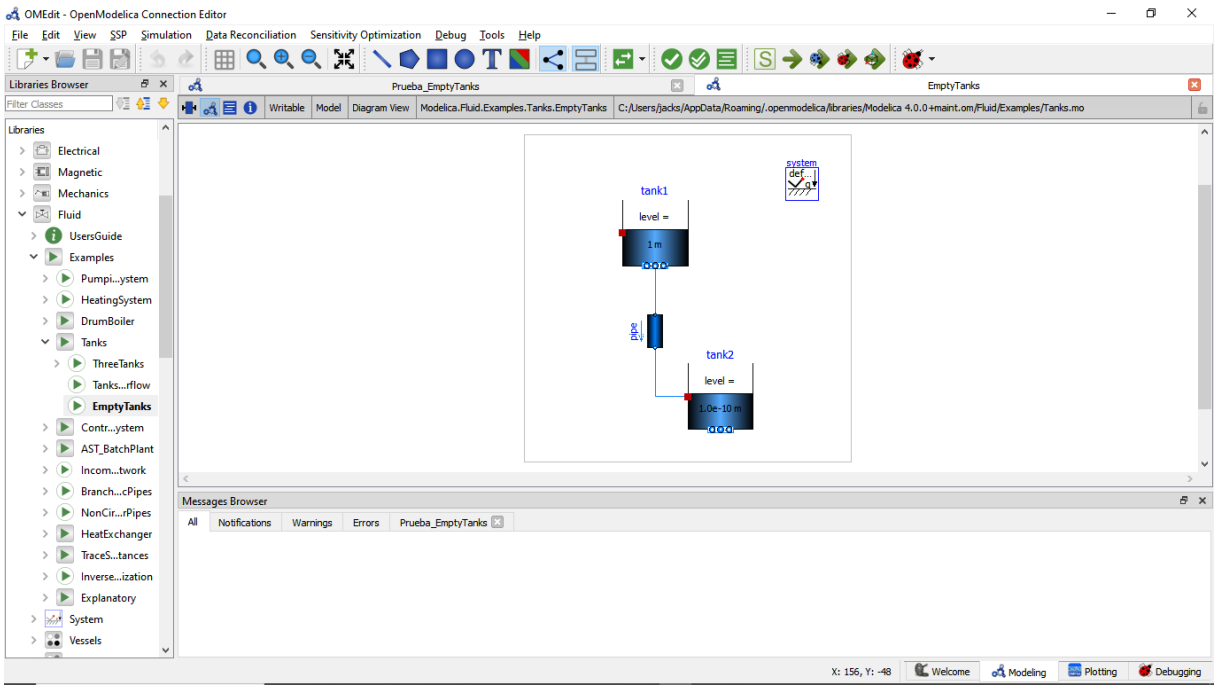

Figura 6.5: Diagrama del modelo de ejemplo *EmptyTanks* en OpenModelica.

Una vez arrancada la aplicación FluidEditor, arrastramos los componentes necesarios (Tanks, Pipe y System) para crear el modelo de ejemplo mostrado en la Figura 6.5 obteniendo algo similar a lo que se muestra en la Figura 6.6.

Una vez que hemos añadido todos los componentes del ejemplo al área de diseño, así como establecido cada una de las conexiones, el siguiente paso consiste en configurar individualmente cada componente. Para hacerlo, hacemos doble clic en cada componente, lo que abrirá una ventana de configuración. En esta ventana, copiaremos los valores de los parámetros extraídos del ejemplo de OpenModelica comentado al inicio de esta sección, para una mejor facilidad de reproducción, estos parámetros de cada componente se han resumido en la Tabla 6.2, 6.1 y 6.3, se han resaltado con negrita aquellos parámetros que son diferentes de los parámetros por defecto que posee cada componente.

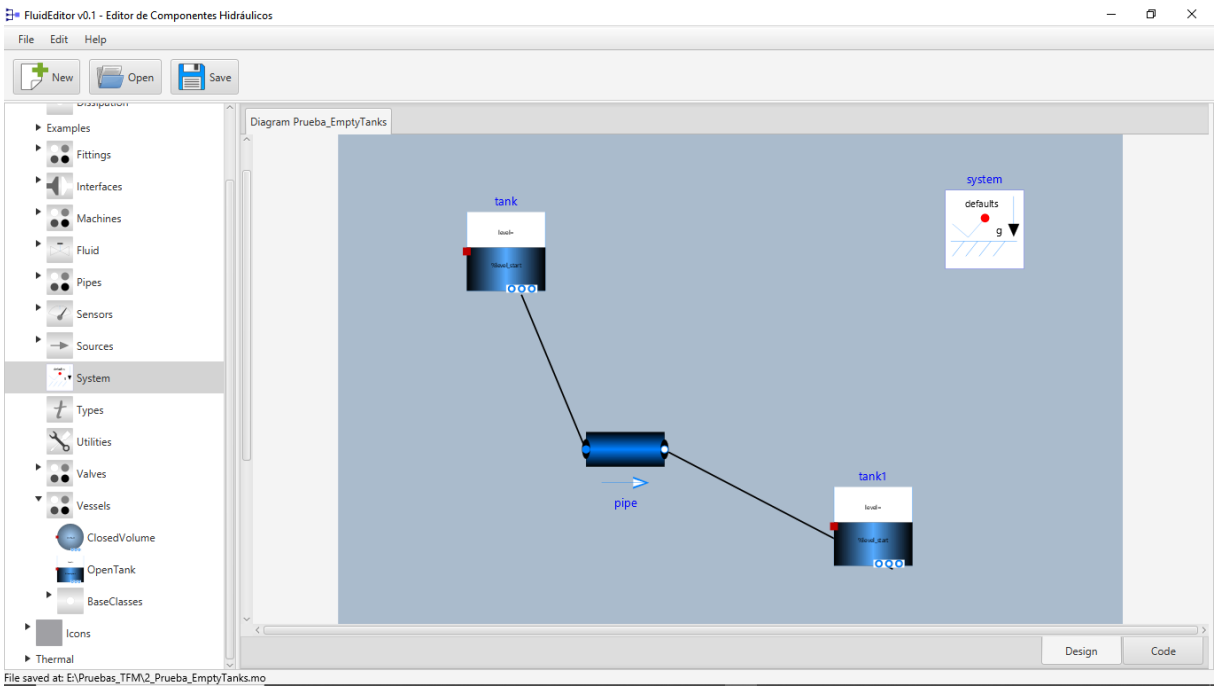

Figura 6.6: Visualización del modelo de prueba *EmptyTanks* en FluidEditor.

La ventana de configuración de los componentes tendrá una apariencia similar a la mostrada en la Figura 6.7. En esta imagen, se resalta un parámetro llamado **Medium**, el cual no está disponible en la ventana de configuración de componentes de OpenModelica. Hemos añadido este parámetro a FluidEditor para permitir la configuración del medio fluido que se utilizará en el modelo. En el ejemplo de la librería, este medio está descrito directamente en el código Modelica. Sin embargo, en nuestro caso, al trabajar en un entorno gráfico sin la capacidad de modificar el código, hemos incluido este parámetro, como un parámetro de configuración. De esta manera, en cada componente que involucre un fluido, será necesario realizar una configuración de este medio (Pipe, Pump, Tank, etc.) En este ejemplo, hemos configurado un fluido constante de agua (*Modelica.Media.Water.ConstantPropertyLiquidWater*), como se muestra en la flecha de la misma imagen.

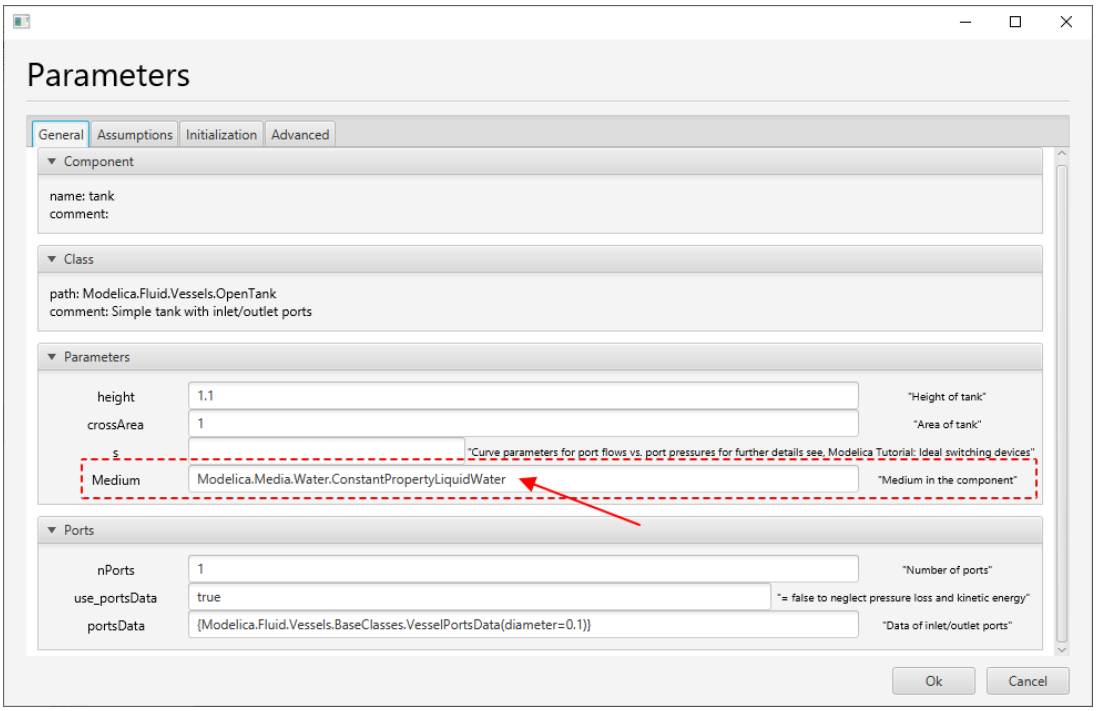

Figura 6.7: Configuración de los componentes del ejemplo de prueba *EmptyTanks*.

Parámetro	Valor
nParallel	1
lenght	**1**
isCircular	true
diameter	**0.1**
crossArea	Modelica.Constants.pi*diameter*diameter/4
perimeter	Modelica.Constants.pi*diameter
roughness	2.5e-5
height$_a$b	**-1**
FlowModel	Modelica.Fluid.Pipes.BaseClasses.FlowModels.DetailedPipeFlow
roughness	2.5e-5
Medium	**Modelica.Media.Water.ConstantPropertyLiquidWater**
p_a_start	system.p_start
p_b_start	p_a_start
m_flow_start	system.m_flow_start
allowFlowReversal	system.allowFlowReversal

Tabla 6.1: Configuración de los parámetros del componente **pipe** del modelo *EmptyTanks*.

Parámetro	Valor
height	**1.1**
crossArea	1
s	
Medium	**Modelica.Media.Water.ConstantPropertyLiquidWater**
nPorts	**1**
use_PortsData	true
portData	**{Modelica.Fluid.Vessels.BaseClasses.VesselPortsData(diameter=0.1)}**
p_ambient	system.p_ambient
T_ambient	system.T_ambient
use_HeatTransfer	false
HeatTransfer	Modelica.Fluid.Vessels.BaseClasses.HeatTransfer.IdealHeatTransfer
energyDynamics	system.energyDynamics
massDynamics	system.massDynamics
level_start	**1**
p_start	system.p_start
use_T_start	true
T_start	if use_T_start then system.T_start else Medium.temperature_phX
h_start	if use_T_start then Medium.specificEnthalpy_pTX else Medium.h_default
X_start	Medium.X_default
C_start	Medium.C_default
m_flow_nominal	if system.use_eps_Re then system.m_flow_nominal else 1e2*system.m_flow_small
m_flow_small	if system.use_eps_Re then system.eps_m_flow*m_flow_nominal else system.m_flow_small
use_Re	system.use_eps_Re

Tabla 6.2: Configuración de los parámetros del componente **tank** del modelo *EmptyTanks*.

Parámetro	Valor
height	**1.1**
crossArea	**1**
s	
Medium	**Modelica.Media.Water.ConstantPropertyLiquidWater**
nPorts	**1**
use_PortsData	true
portData	**{Modelica.Fluid.Vessels.BaseClasses.VesselPortsData(diameter=0.1, height=0.5)}**
p_ambient	system.p_ambient
T_ambient	system.T_ambient
use_HeatTransfer	false
HeatTransfer	Modelica.Fluid.Vessels.BaseClasses.HeatTransfer.IdealHeatTransfer
energyDynamics	system.energyDynamics
massDynamics	system.massDynamics
level_start	**1.0e-10**
p_start	system.p_start
use_T_start	true
T_start	if use_T_start then system.T_start else Medium.temperature_phX
h_start	if use_T_start then Medium.specificEnthalpy_pTX else Medium.h_default
X_start	Medium.X_default
C_start	Medium.C_default
m_flow_nominal	if system.use_eps_Re then system.m_flow_nominal else 1e2*system.m_flow_small
m_flow_small	if system.use_eps_Re then system.eps_m_flow*m_flow_nominal else system.m_flow_small
use_Re	system.use_eps_Re

Tabla 6.3: Configuración de los parámetros del componente **tank1** del modelo *EmptyTanks*.

El código generado por FluidEditor se muestra en el Código 6.2. A continuación, debemos verificar este código en otro entorno de modelado, en este caso vamos a utilizar OpenModelica. Para lograr esto, guardamos el archivo en un directorio de nuestra elección utilizando el botón correspondiente de guardar dentro de FluidEditor. Luego, desde OpenModelica, cargamos el archivo previamente guardado, lo que resulta en una interfaz similar a la que se muestra en la Figura 6.8. Al comparar esta representación con la de la Figura 6.6, podemos confirmar que la representación gráfica en OpenModelica es correcta.

```
1  model Prueba_EmptyTanks
2     Modelica.Fluid.Vessels.OpenTank tank(redeclare package Medium=
          Modelica.Media.Water.ConstantPropertyLiquidWater,nPorts=1,
          level_start=1,crossArea=1,portsData={Modelica.Fluid.Vessels.
          BaseClasses.VesselPortsData(diameter=0.1)},height=1.1) annotation
          (Placement(visible=true,transformation(origin={-56.4,58.8},extent
          ={{-10.0,-10.0},{10.0,10.0}})));
3     Modelica.Fluid.Pipes.StaticPipe pipe(redeclare package Medium=
          Modelica.Media.Water.ConstantPropertyLiquidWater,diameter=0.1,
          length=1,height_ab=-1) annotation(Placement(visible=true,
          transformation(origin={-26.8,9.2},extent
          ={{-10.0,-10.0},{10.0,10.0}})));
4     Modelica.Fluid.Vessels.OpenTank tank1(redeclare package Medium=
          Modelica.Media.Water.ConstantPropertyLiquidWater,nPorts=1,
          level_start=1.0e-10,crossArea=1,portsData={Modelica.Fluid.Vessels
          .BaseClasses.VesselPortsData(diameter=0.1, height=0.5)},height
          =1.1) annotation(Placement(visible=true,transformation(origin
          ={36.4,-10.4},extent={{-10.0,-10.0},{10.0,10.0}})));
5    inner Modelica.Fluid.System system(energyDynamics=Modelica.Fluid.Types
         .Dynamics.FixedInitial) annotation(Placement(visible=true,
         transformation(origin={65.6,64.4},extent
         ={{-10.0,-10.0},{10.0,10.0}})));
6
7
8  equation
9   connect(tank.ports[1],pipe.port_a) annotation(Line(points
        ={{-53.2,48.4},{-36.4,8}},pattern=LinePattern.Solid,color={0,0,0}))
        ;
10  connect(pipe.port_b,tank1.ports[1]) annotation(Line(points
        ={{-15.6,8.4},{41.2,-21.2}},pattern=LinePattern.Solid,color
        ={0,0,0}));
11
12 end Prueba_EmptyTanks;
```

Código 6.2: Código Modelica generado con FluidEditor del modelo EmptyTanks.

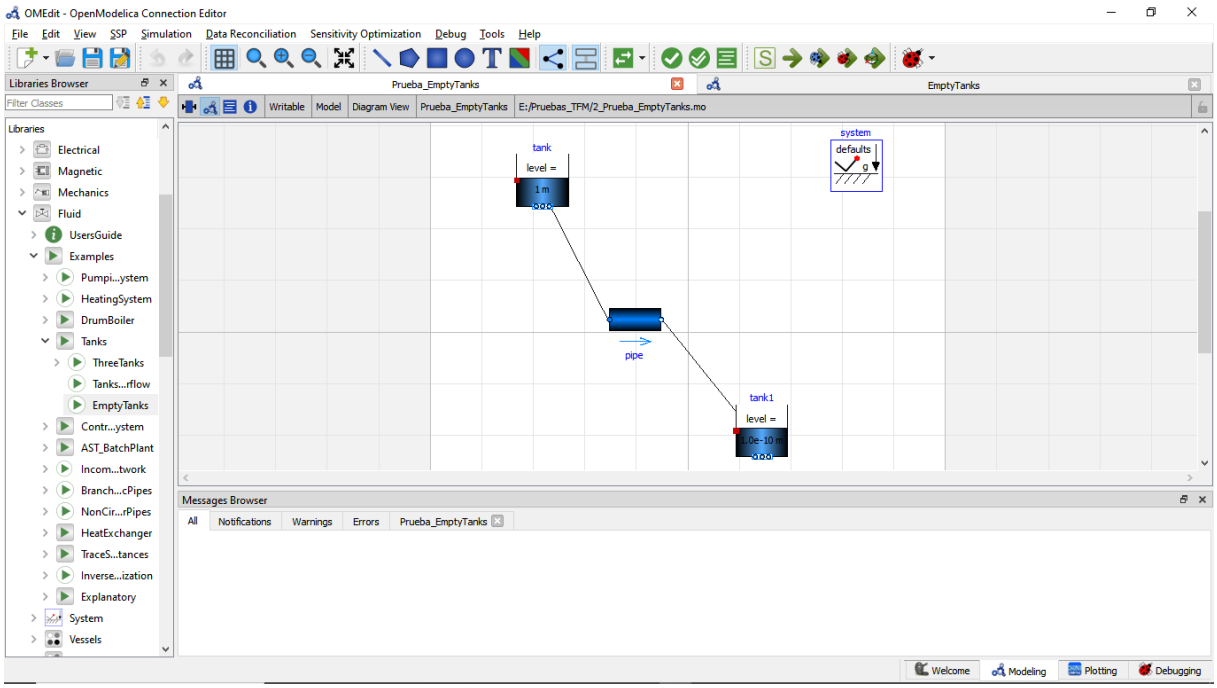

Figura 6.8: Visualización del modelo de prueba *EmptyTanks* en OpenModelica.

A continuación, procedemos a verificar el código cargado en OpenModelica. Al hacer clic en el botón correspondiente, se genera un mensaje que indica que el modelo es correcto, mostrando el número de ecuaciones y variables involucradas, como se ilustra en la Figura 6.9.

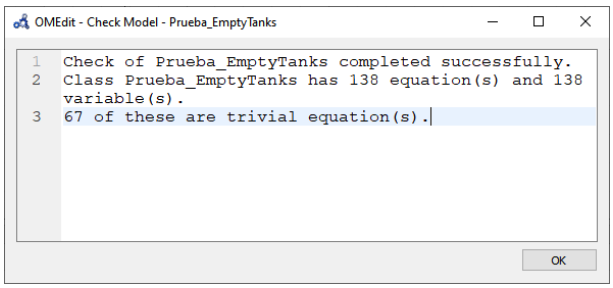

Figura 6.9: Comprobación del modelo *EmptyTanks* en OpenModelica.

Finalmente, procedemos a simular ambos modelos: el modelo diseñado en FluidEditor y el modelo de ejemplo original de la librería Fluid. Para ello, configuramos los parámetros de simulación, como se muestra en la Figura 6.10. Esto iniciará la simulación, debemos de realizar el procedimiento para ambos casos. Una vez finalizada la simulación, representaremos el volumen de líquido de los tanques en ambos modelos. Como se trata del vaciado de un tanque (*Tank1* a *Tank2*) se observará que uno de los tanques su volumen disminuye, mientras que en el otro tanque, el volumen aumenta. Los resultados se presentan en la Figura 6.11.

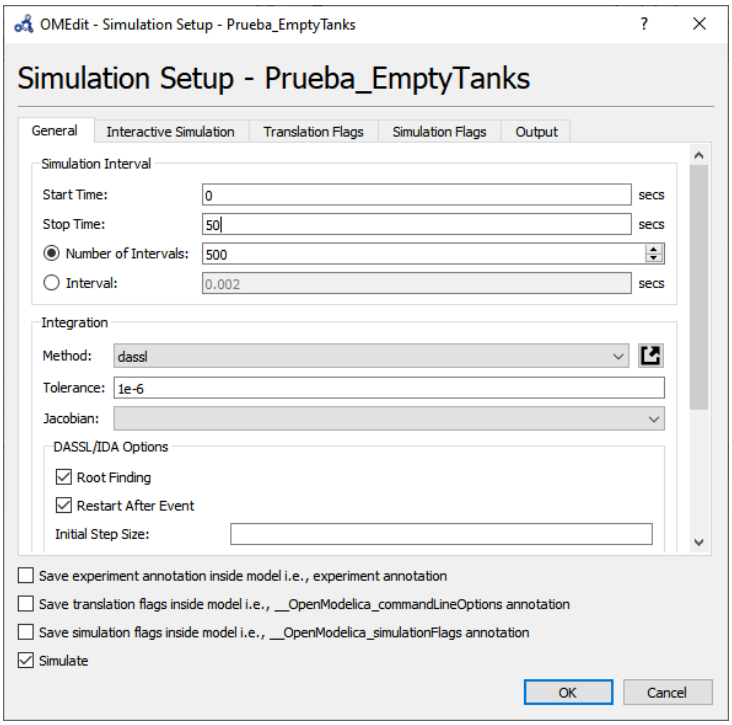

Figura 6.10: Configuración de los parámetros de simulación para el modelo *EmptyTanks*.

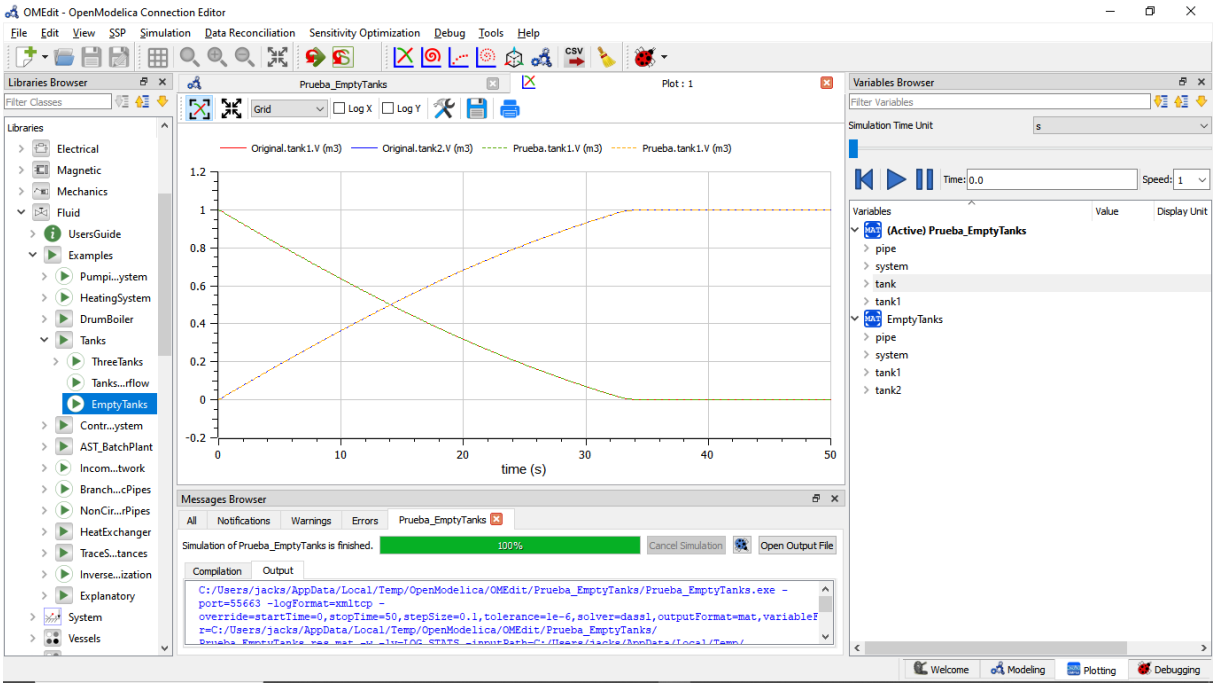

Figura 6.11: Visualización de los resultados de la simulación para ambos modelos (original y generado con FluidEditor).

La similitud de los resultados entre ambos modelos se muestra en la Figura 6.12 y demuestra que el código generado por FluidEditor es correcto y funcional. Se han obtenido los mismos resultados en la simulación para ambos casos. En las siguientes subs-ecciones, abordaremos otros ejemplos ligeramente más complejos para seguir evaluando el correcto funcionamiento de la aplicación, en lo que respecta a generar código correcto y funcional.

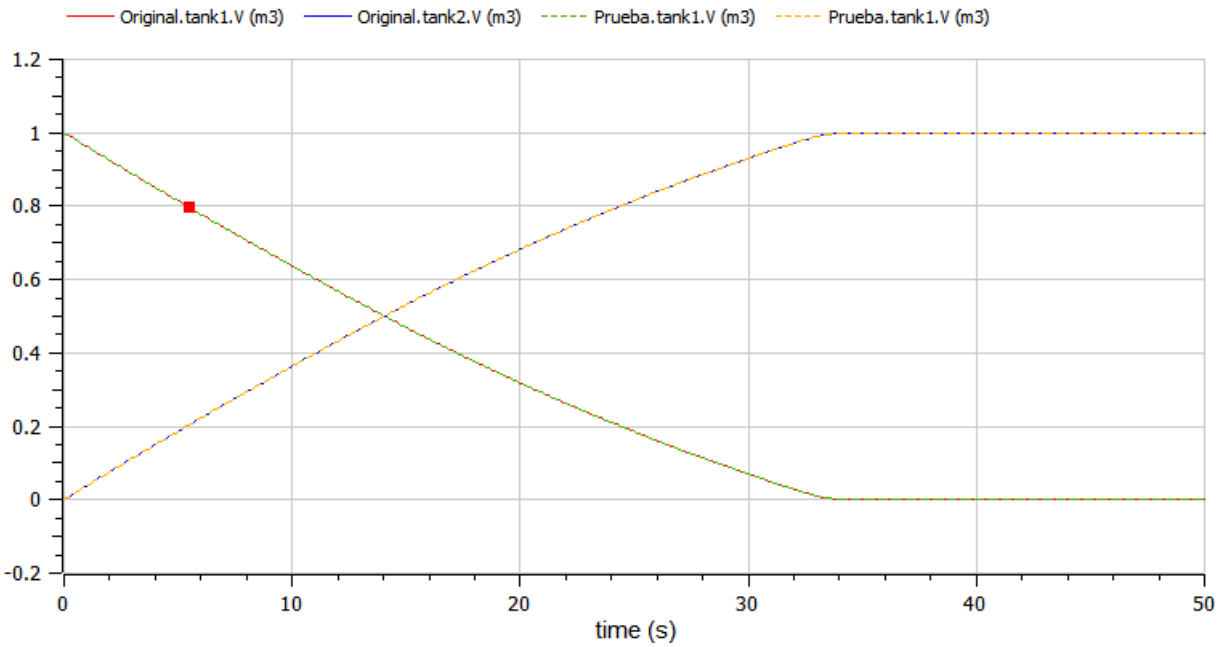

Figura 6.12: Comparación del volumen de líquido de los tanques para ambos modelos (original y generado con FluidEditor).

6.4. Segundo modelo de prueba: *ThreeTanks*

En este segundo modelo utilizamos como prueba, el ejemplo incluido en en el propio paquete **Fluid** de la librería estándar Modelica, conocido como *ThreeTanks*, el cual se encuentra en la ruta *Modelica.Icons.Example.Tanks.TreeTanks*. En la Figura 6.13 se muestra el diagrama asociado a este modelo.

En FluidEditor, arrastramos los componentes necesarios desde el árbol de componentes (ubicado a la izquierda de la aplicación) hacia el área de diseño hasta conseguir el modelo del ejemplo que estamos planteando, el mismo que tiene que tener las conexiones correspondientes. El resultado del diseño será similar a lo que se muestra en la Figura 6.14.

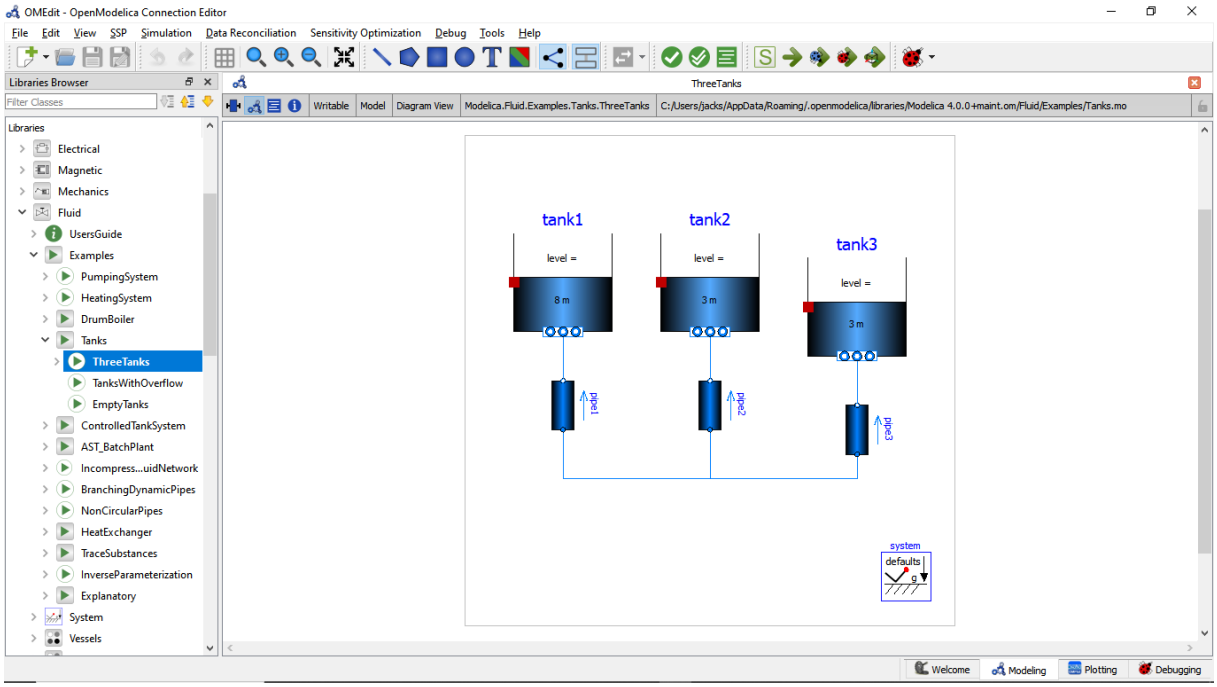

Figura 6.13: Diagrama del segundo modelo de prueba: *ThreeTanks*.

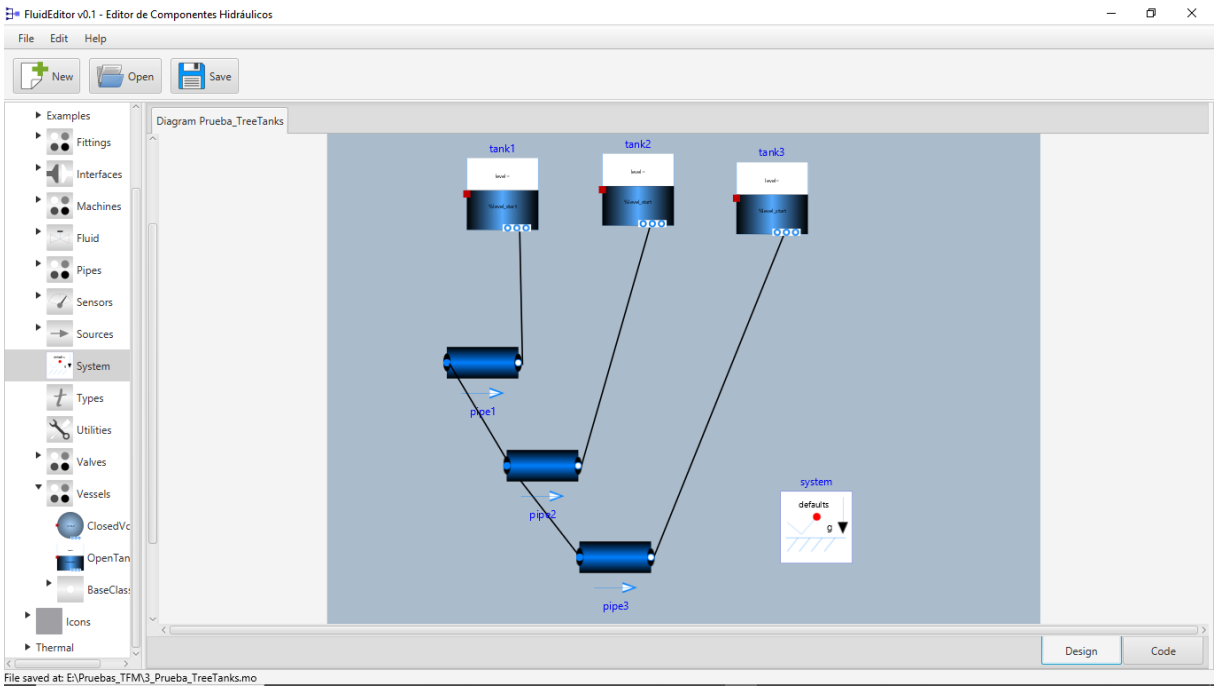

Figura 6.14: Diagrama del modelo *ThreeTanks* diseñado en FluidEditor.

Configuramos cada uno de los parámetros de los componentes, extrayendo la información del propio ejemplo de OpenModelica que estamos considerando, para facilitar este proceso, estos parámetros se han incluido en la Tabla 6.4 y 6.5, en donde solo se han incluido aquellos parámetros que son distintos respecto a los parámetros por defecto que trae configurado cada componente.

Una vez completada la etapa de diseño del modelo, pasamos al modo de vista de código para visualizar el código Modelica generado, este será similar a lo que se muestra en la Figura 6.15, de igual forma dicho código se incluye en texto plano en Código 6.3.

Componente	Parámetro	Valor
tank1	height	**12**
	crossArea	**1**
	Medium	**Modelica.Media.Water.ConstantPropertyLiquidWater**
	nPorts	**1**
	portData	**{Modelica.Fluid.Vessels.BaseClasses.VesselPortsData(diameter=0.1)}**
	level_start	**8**
Componente	Parámetro	Valor
tank2	height	**12**
	crossArea	**1**
	Medium	**Modelica.Media.Water.ConstantPropertyLiquidWater**
	nPorts	**1**
	portData	**{Modelica.Fluid.Vessels.BaseClasses.VesselPortsData(diameter=0.1)}**
	level_start	**3**
Componente	Parámetro	Valor
tank3	height	**12**
	crossArea	**1**
	Medium	**Modelica.Media.Water.ConstantPropertyLiquidWater**
	nPorts	**1**
	portData	**{Modelica.Fluid.Vessels.BaseClasses.VesselPortsData(diameter=0.1)}**
	level_start	**3**

Tabla 6.4: Configuración de los parámetros de los componentes **tanks** del modelo *ThreeTanks*.

Componente	Parámetro	Valor
pipe1	lenght	**2**
	diameter	**0.1**
	height_ab	**2**
	Medium	**Modelica.Media.Water.ConstantPropertyLiquidWater**
Componente	Parámetro	Valor
pipe2	lenght	**2**
	diameter	**0.1**
	height_ab	**2**
	Medium	**Modelica.Media.Water.ConstantPropertyLiquidWater**
Componente	Parámetro	Valor
pipe3	lenght	**2**
	diameter	**0.1**
	height_ab	**-1**
	Medium	**Modelica.Media.Water.ConstantPropertyLiquidWater**

Tabla 6.5: Configuración de los parámetros de los componente **pipe** del modelo *ThreeTanks*.

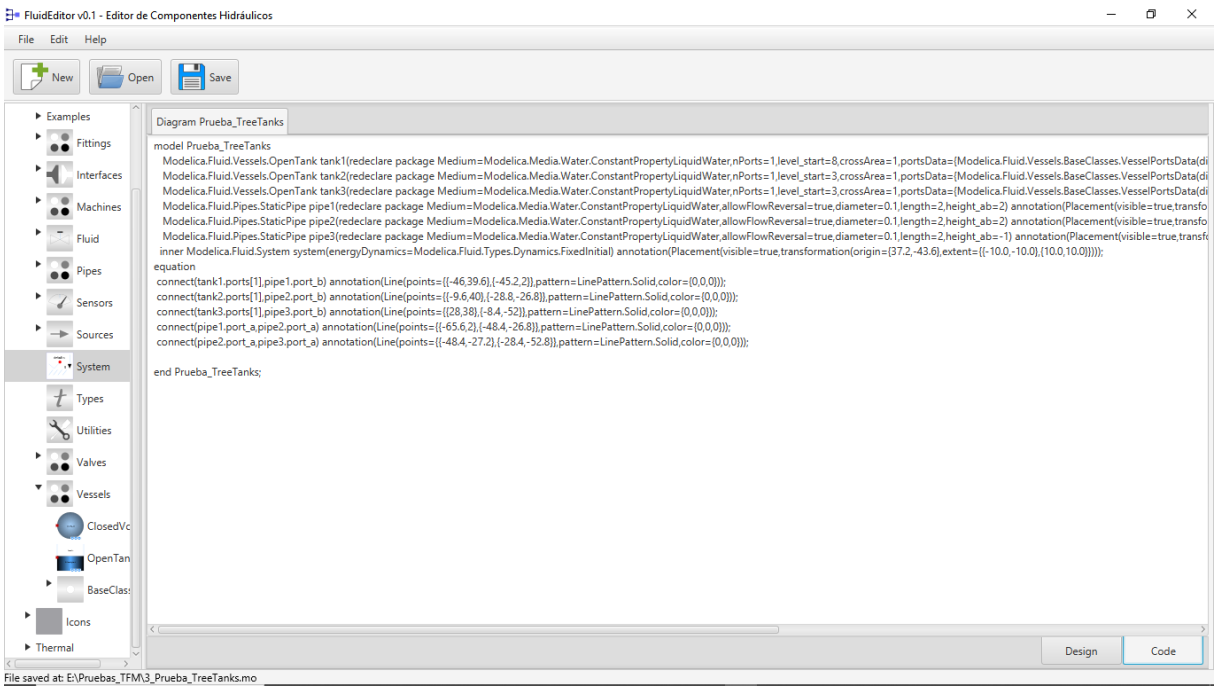

Figura 6.15: Código Modelica del modelo *ThreeTanks* generado por FluidEditor.

```
1 model Prueba_ThreeTanks
2   Modelica.Fluid.Vessels.OpenTank tank1(redeclare package Medium=
        Modelica.Media.Water.ConstantPropertyLiquidWater,nPorts=1,
        level_start=8,crossArea=1,portsData={Modelica.Fluid.Vessels.
        BaseClasses.VesselPortsData(diameter = 0.1)},height=12,
        use_portsData=true) annotation(Placement(visible=true,
        transformation(origin={-50.8,49.6},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
3   Modelica.Fluid.Vessels.OpenTank tank2(redeclare package Medium=
        Modelica.Media.Water.ConstantPropertyLiquidWater,nPorts=1,
        level_start=3,crossArea=1,portsData={Modelica.Fluid.Vessels.
        BaseClasses.VesselPortsData(diameter = 0.1)},height=12,
        use_portsData=true) annotation(Placement(visible=true,
        transformation(origin={-12.8,50.8},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
4   Modelica.Fluid.Vessels.OpenTank tank3(redeclare package Medium=
        Modelica.Media.Water.ConstantPropertyLiquidWater,nPorts=1,
        level_start=3,crossArea=1,portsData={Modelica.Fluid.Vessels.
        BaseClasses.VesselPortsData(diameter = 0.1)},height=12,
        use_portsData=true) annotation(Placement(visible=true,
        transformation(origin={24.8,48.4},extent
        ={{-10.0,-10.0},{10.0,10.0}})));
5   Modelica.Fluid.Pipes.StaticPipe pipe1(redeclare package Medium=
        Modelica.Media.Water.ConstantPropertyLiquidWater,
        allowFlowReversal=true,diameter=0.1,length=2,height_ab=2)
        annotation(Placement(visible=true,transformation(origin
        ={-55.6,1.6},extent={{-10.0,-10.0},{10.0,10.0}})));
6   Modelica.Fluid.Pipes.StaticPipe pipe2(redeclare package Medium=
        Modelica.Media.Water.ConstantPropertyLiquidWater,
        allowFlowReversal=true,diameter=0.1,length=2,height_ab=2)
        annotation(Placement(visible=true,transformation(origin
        ={-39.6,-26.4},extent={{-10.0,-10.0},{10.0,10.0}})));
7   Modelica.Fluid.Pipes.StaticPipe pipe3(redeclare package Medium=
        Modelica.Media.Water.ConstantPropertyLiquidWater,
```

```
        allowFlowReversal=true,diameter=0.1,length=2,height_ab=-1)
        annotation(Placement(visible=true,transformation(origin
        ={-19.2,-52},extent={{-10.0,-10.0},{10.0,10.0}})));
  inner Modelica.Fluid.System system(energyDynamics=Modelica.Fluid.Types
      .Dynamics.FixedInitial) annotation(Placement(visible=true,
      transformation(origin={37.2,-43.6},extent
      ={{-10.0,-10.0},{10.0,10.0}})));
equation
  connect(tank1.ports[1],pipe1.port_b) annotation(Line(points
      ={{-46,39.6},{-45.2,2}},pattern=LinePattern.Solid,color={0,0,0}));
  connect(tank2.ports[1],pipe2.port_b) annotation(Line(points
      ={{-9.6,40},{-28.8,-26.8}},pattern=LinePattern.Solid,color={0,0,0})
      );
  connect(tank3.ports[1],pipe3.port_b) annotation(Line(points
      ={{28,38},{-8.4,-52}},pattern=LinePattern.Solid,color={0,0,0}));
  connect(pipe1.port_a,pipe2.port_a) annotation(Line(points
      ={{-65.6,2},{-48.4,-26.8}},pattern=LinePattern.Solid,color={0,0,0})
      );
  connect(pipe2.port_a,pipe3.port_a) annotation(Line(points
      ={{-48.4,-27.2},{-28.4,-52.8}},pattern=LinePattern.Solid,color
      ={0,0,0}));

end Prueba_ThreeTanks;
```

Código 6.3: Código Modelica generado con FluidEditor del modelo ThreeTanks.

Al igual que con el modelo de prueba anterior, guardamos el modelo en un archivo dentro de un directorio de nuestra elección. Luego, cargamos dicho archivo en OpenModelica para verificar el modelo y, finalmente, procedemos a llevar a cabo la simulación. Una vez que la simulación se completa, observamos y comparamos los resultados entre el modelo original (el ejemplo de la librería Fluid) y el modelo generado con FluidEditor. En este caso, representamos la evolución de los volúmenes de los tanques. Los resultados se presentan en la Figura 6.16.

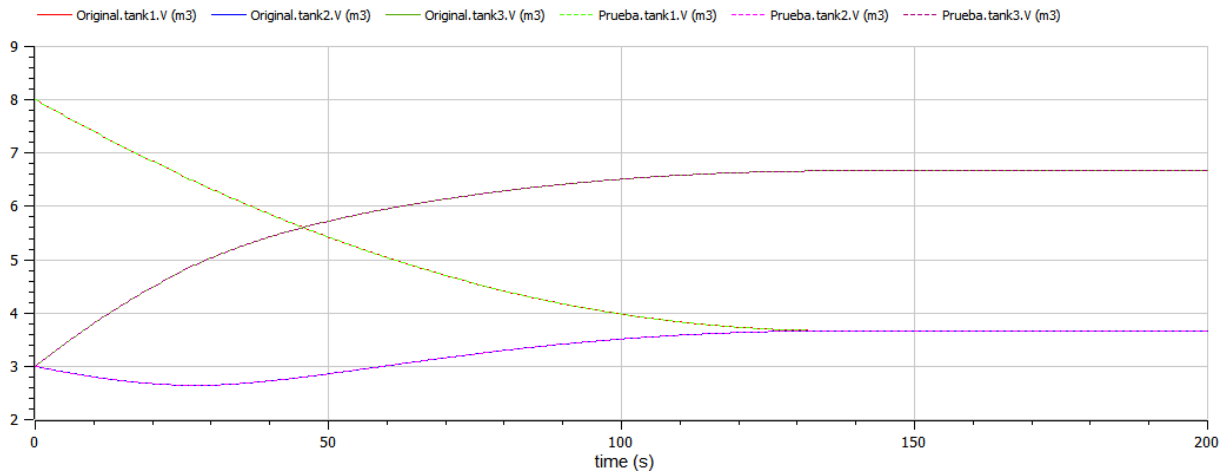

Figura 6.16: Comparación de la evolución de los volúmenes de los tanques para el modelo *ThreeTanks* original y el obtenido mediante FluidEditor.

La comparación de la variación de los volúmenes de líquido de los tanques entre ambos modelos (original y generado por FluidEditor) es el mismo. Tanto el primer como el segundo tanque convergen a un nivel similar de aproximadamente $3{,}6m^3$, mientras que el tercer tanque converge a un nivel de $6{,}6m^3$. Esto se debe a que la altura de las tuberías que conectan los dos primeros tanques se encuentra a la misma diferencia de alturas, mientras que la tubería que conecta al tercer tanque está un metro por debajo. En definitiva, en este tipo de problemas, los volúmenes de líquido convergen al equilibrio.

6.5. Tercer modelo de prueba: *PumpingSystem*

En este tercer modelo vamos a utiliza el ejemplo *PumpingSystem* que se ubica en la ruta *Modelica.Icons.Example.PumpingSystem*. La documentación describe el sistema de la siguiente manera:

"El agua se bombea desde una fuente mediante una bomba (provista de válvulas de retención), a través de una tubería cuya salida está $50m$ por encima de la fuente, hasta un depósito. Los usuarios están representados por una válvula equivalente, conectada al depósito.

El controlador de agua es un simple controlador *on-off*, que regula la presión manométrica medida en la base de la torre; la salida del controlador es la velocidad de rotación de la bomba, que está representada por la salida de una función de transferencia de primer orden. Se utiliza una velocidad de rotación pequeña pero distinta de cero para representar el estado de espera de las bombas, con el fin de evitar singularidades en las características del flujo.

Cuando comienza la simulación, el nivel está por encima del punto de ajuste, por lo que el estado inicial del controlador de la bomba es apagado. Por lo tanto, la válvula de retención de la bomba está activada. Para facilitar la solución del problema de inicialización, se establece el parámetro homotopyType.

Simular durante $2000s$. Cuando se abre la válvula en el momento $t = 200s$, la bomba comienza a encenderse y se apaga para mantener el nivel del depósito alrededor de dos metros ($2m$), lo que corresponde aproximadamente a una presión manométrica de $200mbar$." (Información extraída de https://reference.wolfram.com/system-modeler/libraries/Modelica/Modelica.Fluid.Examples.PumpingSystem.html).

El diagrama original de este ejemplo se muestra en la Figura 6.17.

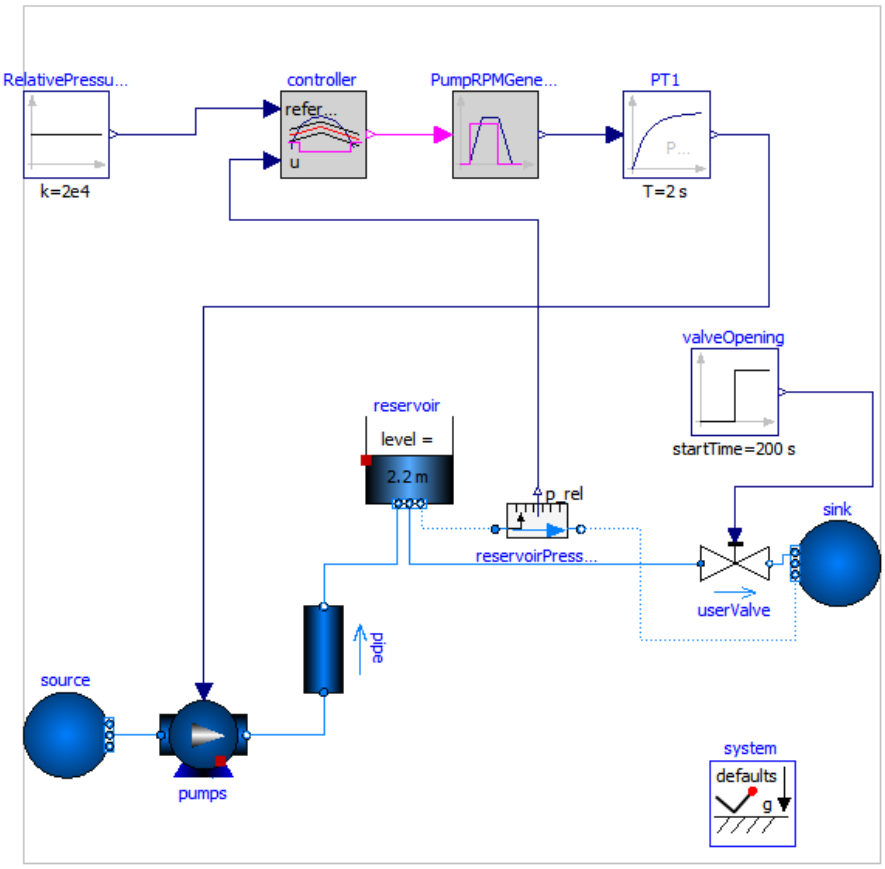

Figura 6.17: Diagrama original del modelo de ejemplo *PumpingSystem*.

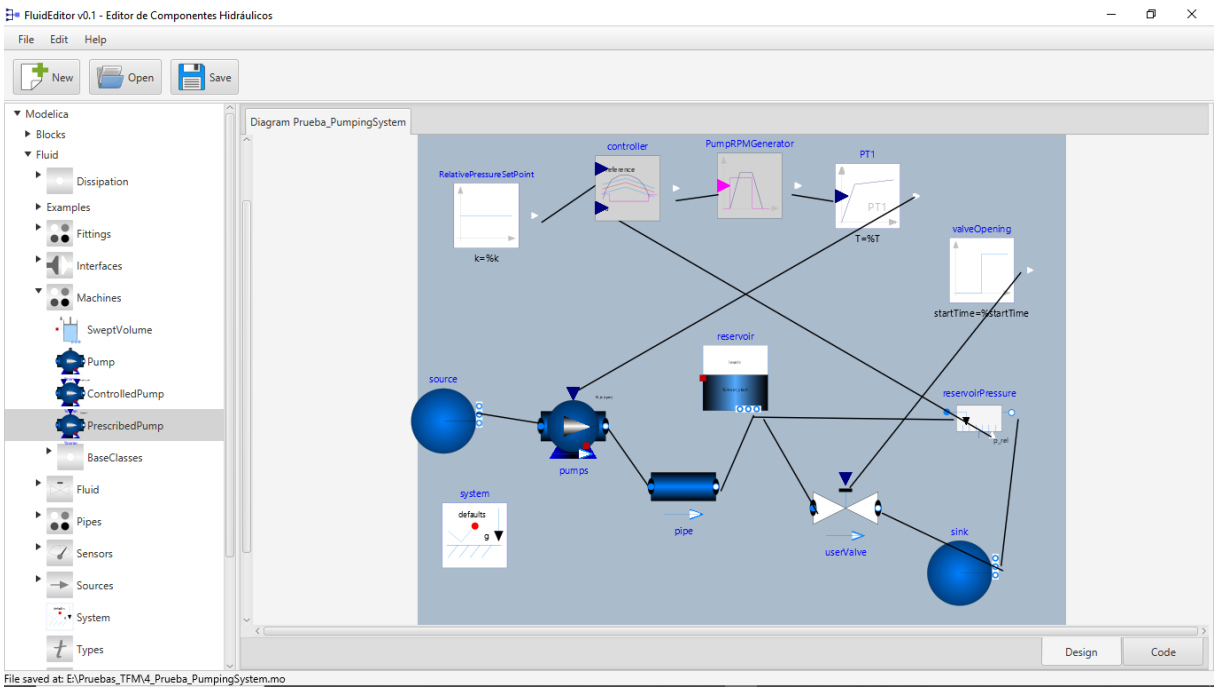

Figura 6.18: Diseño del modelo *PumpingSystem* en FluidEditor.

Posteriormente, procedemos a diseñar nuestro propio modelo utilizando FluidEditor. En esta etapa, arrastramos y configuramos cada uno de los componentes necesarios, establecemos las conexiones y ajustamos los parámetros de acuerdo a los valores del ejemplo original. El resultado de este proceso de diseño se puede observar en la Figura 6.18.

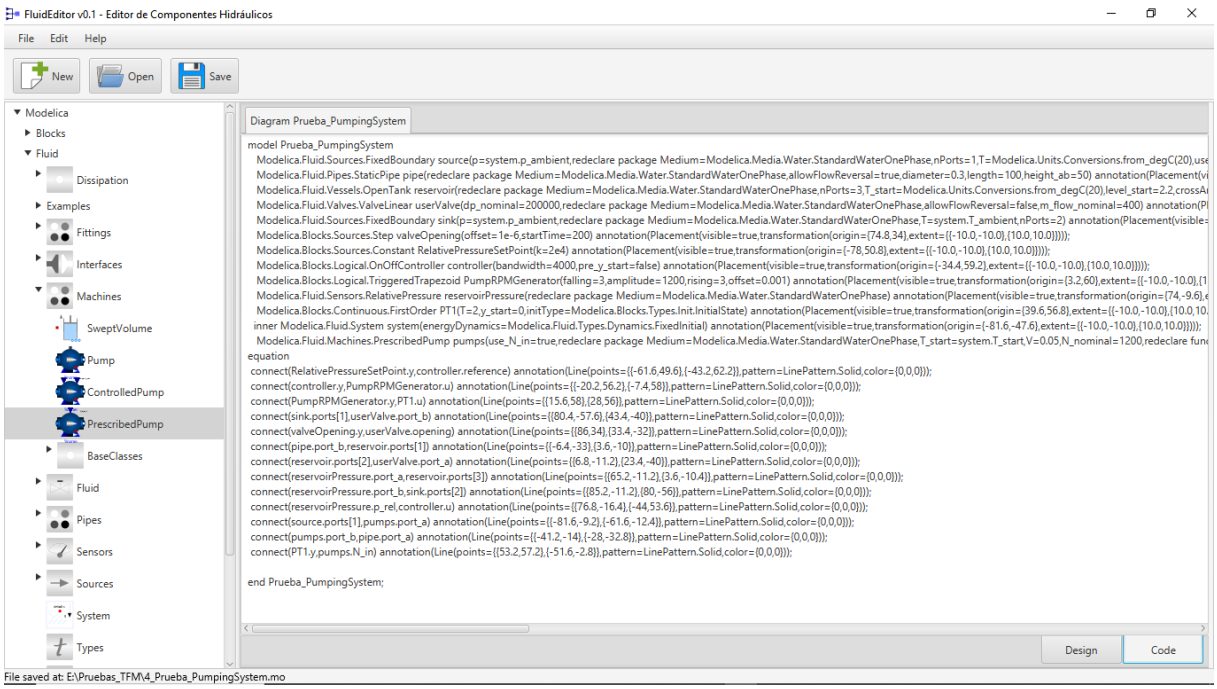

Figura 6.19: Código generado por FluidEditor para el modelo *PumpingSystem*.

Si nos dirigimos al área de código y exploramos el código generado, obtenemos algo similar a lo que se muestra en la Figura 6.19, de igual forma se incluido dicho código en texto plano en Código 6.4.

```
1 model Prueba_ThreeTanks
2    Modelica.Fluid.Vessels.OpenTank tank1(redeclare package Medium=
         Modelica.Media.Water.ConstantPropertyLiquidWater,nPorts=1,
         level_start=8,crossArea=1,portsData={Modelica.Fluid.Vessels.
         BaseClasses.VesselPortsData(diameter = 0.1)},height=12,
         use_portsData=true) annotation(Placement(visible=true,
         transformation(origin={-50.8,49.6},extent
         ={{-10.0,-10.0},{10.0,10.0}})));
3    Modelica.Fluid.Vessels.OpenTank tank2(redeclare package Medium=
         Modelica.Media.Water.ConstantPropertyLiquidWater,nPorts=1,
         level_start=3,crossArea=1,portsData={Modelica.Fluid.Vessels.
         BaseClasses.VesselPortsData(diameter = 0.1)},height=12,
         use_portsData=true) annotation(Placement(visible=true,
         transformation(origin={-12.8,50.8},extent
         ={{-10.0,-10.0},{10.0,10.0}})));
4    Modelica.Fluid.Vessels.OpenTank tank3(redeclare package Medium=
         Modelica.Media.Water.ConstantPropertyLiquidWater,nPorts=1,
         level_start=3,crossArea=1,portsData={Modelica.Fluid.Vessels.
         BaseClasses.VesselPortsData(diameter = 0.1)},height=12,
         use_portsData=true) annotation(Placement(visible=true,
         transformation(origin={24.8,48.4},extent
         ={{-10.0,-10.0},{10.0,10.0}})));
```

```
5     Modelica.Fluid.Pipes.StaticPipe pipe1(redeclare package Medium=
          Modelica.Media.Water.ConstantPropertyLiquidWater,
          allowFlowReversal=true,diameter=0.1,length=2,height_ab=2)
          annotation(Placement(visible=true,transformation(origin
          ={-55.6,1.6},extent={{-10.0,-10.0},{10.0,10.0}})));
6     Modelica.Fluid.Pipes.StaticPipe pipe2(redeclare package Medium=
          Modelica.Media.Water.ConstantPropertyLiquidWater,
          allowFlowReversal=true,diameter=0.1,length=2,height_ab=2)
          annotation(Placement(visible=true,transformation(origin
          ={-39.6,-26.4},extent={{-10.0,-10.0},{10.0,10.0}})));
7     Modelica.Fluid.Pipes.StaticPipe pipe3(redeclare package Medium=
          Modelica.Media.Water.ConstantPropertyLiquidWater,
          allowFlowReversal=true,diameter=0.1,length=2,height_ab=-1)
          annotation(Placement(visible=true,transformation(origin
          ={-19.2,-52},extent={{-10.0,-10.0},{10.0,10.0}})));
8    inner Modelica.Fluid.System system(energyDynamics=Modelica.Fluid.Types
         .Dynamics.FixedInitial) annotation(Placement(visible=true,
         transformation(origin={37.2,-43.6},extent
         ={{-10.0,-10.0},{10.0,10.0}})));
9  equation
10   connect(tank1.ports[1],pipe1.port_b) annotation(Line(points
         ={{-46,39.6},{-45.2,2}},pattern=LinePattern.Solid,color={0,0,0}));
11   connect(tank2.ports[1],pipe2.port_b) annotation(Line(points
         ={{-9.6,40},{-28.8,-26.8}},pattern=LinePattern.Solid,color={0,0,0})
         );
12   connect(tank3.ports[1],pipe3.port_b) annotation(Line(points
         ={{28,38},{-8.4,-52}},pattern=LinePattern.Solid,color={0,0,0}));
13   connect(pipe1.port_a,pipe2.port_a) annotation(Line(points
         ={{-65.6,2},{-48.4,-26.8}},pattern=LinePattern.Solid,color={0,0,0})
         );
14   connect(pipe2.port_a,pipe3.port_a) annotation(Line(points
         ={{-48.4,-27.2},{-28.4,-52.8}},pattern=LinePattern.Solid,color
         ={0,0,0}));
15
16 end Prueba_ThreeTanks;
```

Código 6.4: Código Modelica generado con FluidEditor del modelo PumpingSystem.

Una vez completada la fase de diseño, guardamos el modelo en un directorio de preferencia y lo cargamos en el entorno de OpenModelica. Allí, verificamos que todo esté correctamente configurado a nivel visual, a nivel de parámetros y a nivel de planteamiento del problema, finalmente procedemos a simular el sistema durante 2000 segundos, utilizando intervalos de 0,4 segundos, lo que equivale a 5000 intervalos en total. Los resultados de dicha simulación para ambos modelos, tanto el original como el generado por FluidEditor, se presentan en las Figuras 6.20 y 6.21. En la primera figura, se muestra la evolución del volumen del tanque (la variable `Original.reservoir.v` de color rojo para el original yla variable `Prueba.reservoir.v` de color verde para el de prueba), mientras que en la segunda figura se visualiza la evolución de la función de transferencia que modifica la velocidad de la bomba (`Original.PT1.y` para el original y `Prueba.PT1.y` para el de prueba). Se puede observar cómo el volumen del tanque comienza con un valor constante, luego empieza a disminuir hasta aproximadamente ($92m^3$) que corresponde al nivel del deposito de 2 metros, y cuando alcanza dicho punto, la función de transferencia activa la

bomba, llenando el tanque. Este proceso se repite a lo largo del tiempo.

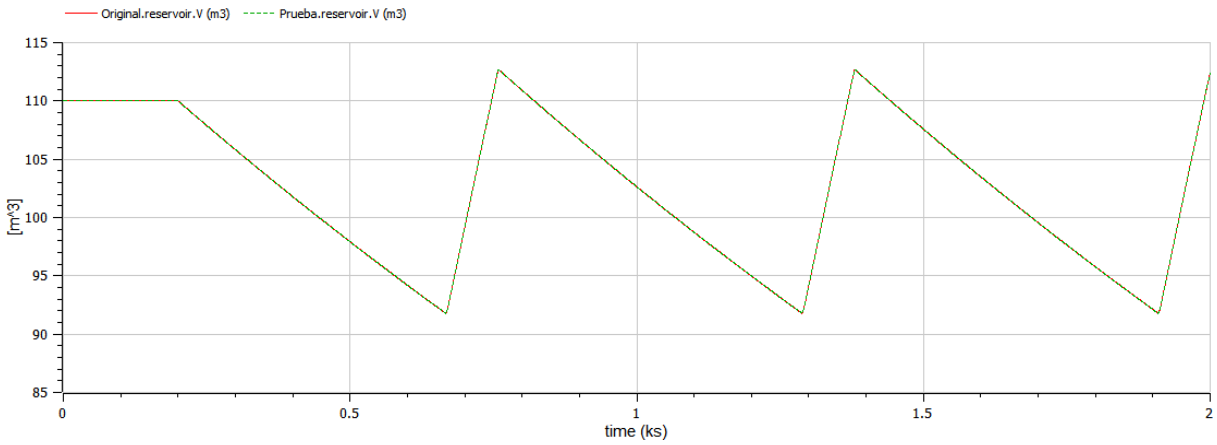

Figura 6.20: Evolución del volumen del líquido en el tanque (reservorio) del modelo *PumpingSystem*.

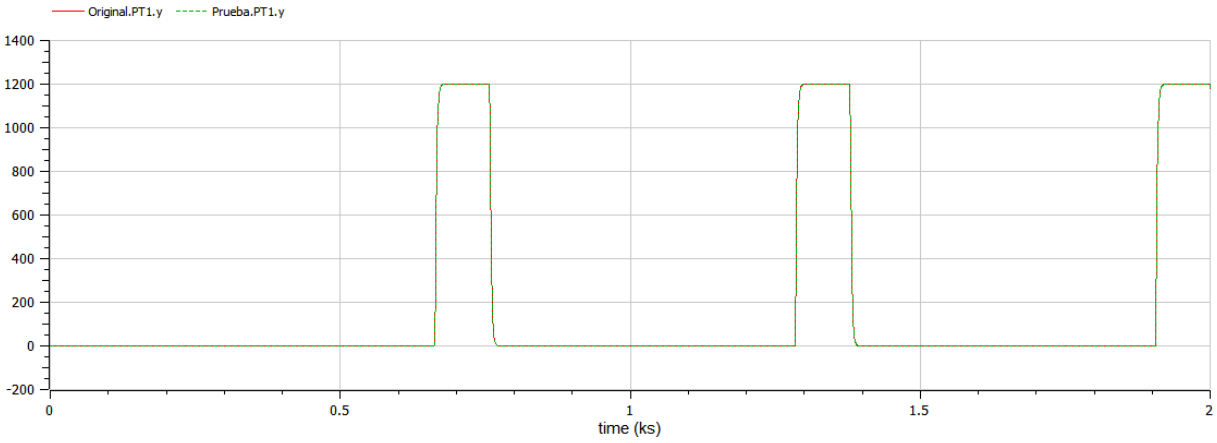

Figura 6.21: Evolución de la función de transferencia que permite activar la bomba en el modelo *PumpingSystem*.

Para una mejor comparación, repetimos la simulación utilizando Wolfram System Modeler. Iniciamos la aplicación y cargamos nuestro modelo. El diseño cargado se asemeja al mostrado en la Figura 6.4. Sin embargo, al intentar verificar el sistema, nos encontramos con un error, tal como se muestra con una flecha en la Figura 6.4. Este error se debe a que la función de transformación de unidades *Modelica.Units.Conversions.`from_degC`(20)* utilizada en los componentes *Source* y *reservoir* no está disponible en esta ruta (posiblemente por la versión de librearía MSL utilizada). Para resolver este problema, modificamos la función que causa el problema por *Modelica.SIunits.Conversions.`from_degC`(20)*, como se indica en la Figura 6.23.

Una vez completado este proceso de corrección, estamos listos para simular ambos modelos y comparar sus resultados. Las Figuras 6.24 muestran los resultados de la simulación, comparando el modelo original con el modelo generado por FluidEditor, utilizando Wolfram System Modeler.

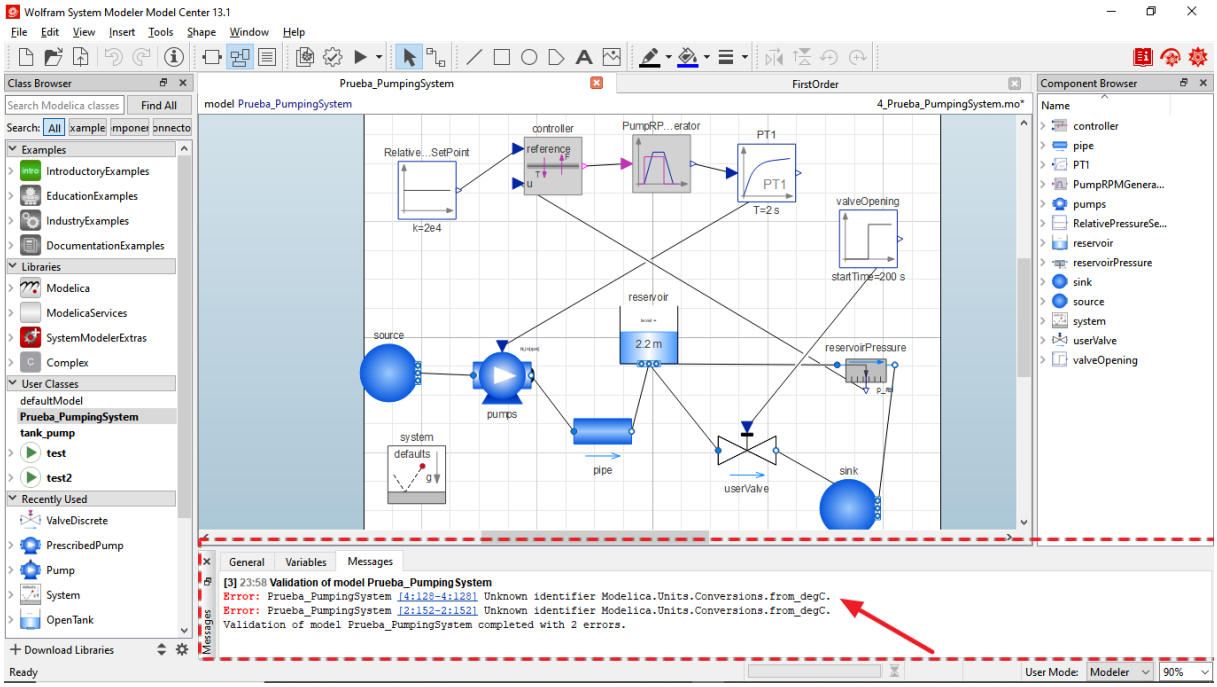

Figura 6.22: Diagrama del modelo *PumpingSystem* cargado en Wolfram System Modeler.

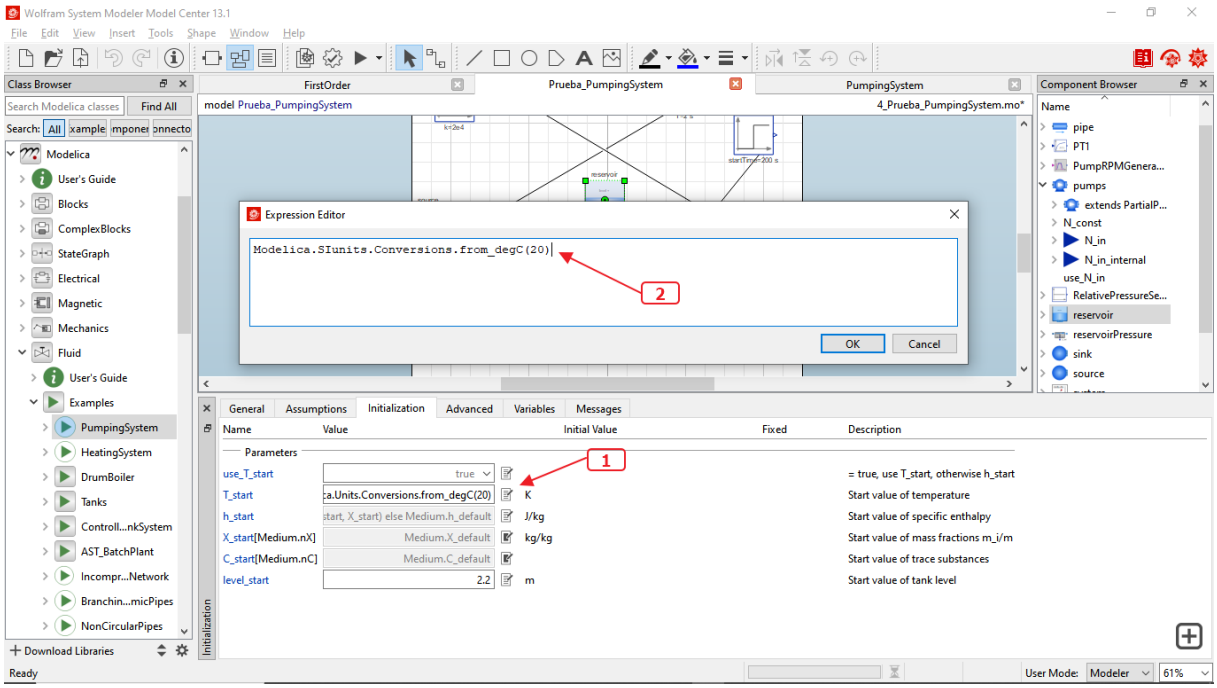

Figura 6.23: Proceso para cambiar una función de transformación de temperatura no disponible en Wolfram System Modeler.

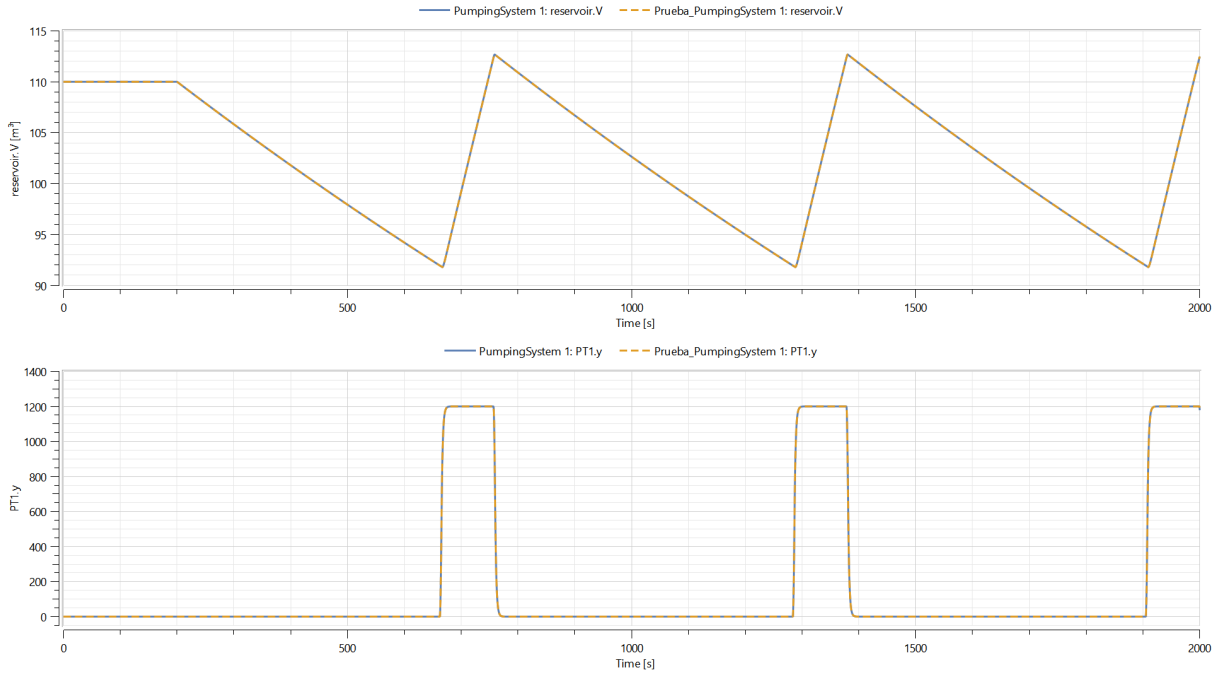

Figura 6.24: Simulación entre el modelo original y el modelo generado con FluidEditor mediante Wolfram System Modeler.

6.6. Conclusiones

En este capítulo, se han elaborado tres modelos compuestos basados en ejemplos provenientes del propio paquete **Fluid** de la librería estándar Modelica, con el propósito de evaluar el desempeño y la funcionalidad de la aplicación FluidEditor. La finalidad principal es la validación de distintos aspectos fundamentales, que incluyen:

- Verificar la correcta representación gráfica de los componentes en el entorno de FluidEditor.
- Confirmar la precisión en el proceso de interconexión entre los diferentes componentes del sistema.
- Evaluar la adecuada visualización y edición de los parámetros inherentes a cada componente.
- Asegurar la generación precisa de código Modelica acorde a la estructura del modelo diseñado.
- Validar la capacidad de almacenar el modelo en un archivo y recuperarlo posteriormente.
- Comprobar que los ficheros guardados se cargan de manera correcta en otros entornos de modelado y simulación.

Para garantizar la veracidad y la funcionalidad del código Modelica generado, se ha recurrido a la utilización de otros entornos de modelado y simulación, específicamente OpenModelica y Wolfram System Modeler 13.1. Los resultados obtenidos han sido altamente satisfactorios, tal como se detalla y se demuestra en los modelos de prueba abordados en las secciones anteriores.

7 CONCLUSIONES Y TRABAJOS FUTUROS

7.1. Introducción

En este capítulo se presenta una conclusión general del proyecto, destacando los resultados alcanzados, los desafíos encontrados durante el desarrollo y las oportunidades de mejora y trabajo futuro que pueden ampliar la funcionalidad de la aplicación.

7.2. Conclusiones

El objetivo central de este proyecto ha sido el desarrollo de una aplicación en lenguaje Java que permita la creación gráfica de modelos compuestos y que genere de manera automática código Modelica. Esta funcionalidad implica seleccionar, arrastrar, soltar y conectar componentes en un área de diseño para formar un modelo deseado. La realización de una aplicación de esta magnitud ha representado un desafío significativo, pero al mismo tiempo ha brindado una oportunidad para aplicar los conocimientos adquiridos durante el proceso. En este contexto, el desarrollo de la aplicación se ha basado en principios fundamentales que trascienden el ámbito del diseño de software, como el principio de "dividir para conquistar". Este enfoque consiste en descomponer el problema en partes más manejables, facilitando así su abordaje.

Siguiendo este enfoque de descomposición, a lo largo del proyecto se ha empleado uno de los patrones más utilizados y consolidados en el desarrollo de software: el patrón Modelo-Vista-Controlador (MVC). La elección de este patrón se debe a su simplicidad y a su amplia aceptación a lo largo del tiempo. El MVC busca desacoplar la interfaz de usuario (vistas) de los modelos subyacentes, utilizando un controlador como intermediario entre ambos.

La implementación exitosa de la aplicación, siguiendo el patrón MVC, ha sido un logro significativo. Sin embargo, el uso del patrón MVC por sí solo no es suficiente. Para cada una de las capas (modelo, vista y controlador), se requiere la elección de tecnologías apropiadas. En el mercado existen múltiples tecnologías, algunas más especializadas en ciertas capas que otras.

Dado el plazo de desarrollo disponible, se optó por utilizar el mismo lenguaje de programación para las tres capas, en este caso, Java, al que se le han sumado algunas librerías como JavaFX para proporcionar a la aplicación una interfaz gráfica moderna.

La división en capas permitió centrarse en cada una de ellas por separado, sin tener que implementar todas al mismo tiempo, con la excepción del controlador, que requería la existencia de las demás. La secuencia de implementación en este proyecto fue la siguiente: modelos, vistas y controlador. Aunque no se trata de una implementación estrictamente secuencial, sino más bien iterativa, esta secuencia proporcionó una estructura organizada para el desarrollo.

A partir del proceso de diseño y desarrollo de la aplicación, se han extraído varias conclusiones clave:

- La fragmentación de una aplicación en partes o bloques más pequeños y manejables resulta esencial en el proceso de diseño, facilitando su desarrollo, su depuración y su mantenimiento en general.
- Identificar las partes críticas, bloques o funcionalidades desde el principio permite evaluar la viabilidad de la aplicación y priorizar recursos de manera efectiva.
- La selección de tecnologías debe basarse en investigación y análisis exhaustivos, eligiendo aquellas que mejor se adapten a cada capa del proyecto. Si se opta por utilizar las mismas tecnologías en todas las capas, el diseño debe contemplar la posibilidad de futuros cambios de tecnología sin afectar drásticamente la aplicación.
- La escritura de código limpio y descriptivo es fundamental para la comprensión del mismo, reduciendo la necesidad de comentarios excesivos.
- La asignación adecuada de responsabilidades a las clases es esencial; distribuir tareas de manera equilibrada entre objetos mejora la cohesión y la mantenibilidad de la aplicación.
- Un profundo conocimiento del modelado en el dominio, así como de las herramientas de modelado y simulación, es esencial para diseñar herramientas de modelado efectivas, como la aplicación desarrollada en este proyecto.

Además, algunas conclusiones específicas relacionadas con el modelado y Modelica son las siguientes:

- Modelica ofrece una amplia biblioteca estándar de modelos predefinidos conocida como MSL (Modelica Standard Library), que puede servir de base para la creación

de otras librerías. Conocer su implementación es esencial para el desarrollo de nuevas herramientas de modelado y simulación.

- El lenguaje Modelica es altamente flexible y permite la creación de diversos modelos, tanto simples como compuestos utilizando la sintaxis de este lenguaje. No obstante, esta flexibilidad también puede conllevar a una mayor complejidad al diseñar herramientas gráficas, ya que se deben contemplar múltiples posibilidades para asegurar una traducción precisa de los elementos gráficos a código Modelica.

En resumen, este proyecto ha proporcionado un escenario valioso para la aplicación de conceptos teóricos y habilidades prácticas en el desarrollo de software, modelado y la integración de tecnologías. Aunque se han logrado avances significativos, también se abre la puerta a futuras iteraciones y mejoras, consolidando un aprendizaje continuo en la intersección del desarrollo de software y la ingeniería de sistemas. La siguiente sección mencionará algunas mejoras y trabajos futuros.

7.3. Trabajos futuros

La aplicación ha demostrado su funcionalidad a través de las pruebas realizadas con los ejemplos presentados en el capítulo de pruebas. Sin embargo, como ocurre con cualquier aplicación de software, su ciclo de vida no culmina con su desarrollo inicial, sino más bien se convierte en un ente vivo que requiere constante actualización, mejora y mantenimiento. Por lo general, una aplicación emerge inicialmente como una versión beta, a pesar de ser completamente funcional y de superar pruebas manuales y automáticas; es en el uso cotidiano cuando pueden surgir errores no anticipados durante las fases de desarrollo y pruebas. En este contexto, una aplicación siempre tiene margen de mejora, y esta no es una excepción. En base a esto, es posible proponer diversas mejoras:

- Mejorar la apariencia de las conexiones entre componentes. Actualmente, el proceso implica seleccionar un conector como origen y arrastrar el ratón hasta el conector de destino, formando una línea recta de conexión. Sería ideal implementar un algoritmo automático que determine la ruta óptima de conexión, o permitir que las líneas de conexión sean más flexibles, permitiendo al usuario definir la ruta con cada clic. De esta forma, la estética del diseño mejoraría de manera notable.

- Mejorar la presentación del código generado. Esto podría incluir la incorporación de librerías o complementos que resalten las palabras clave del lenguaje Modelica, numeración de líneas, entre otros aspectos.

- Agregar una barra de herramientas que permita realizar operaciones como deshacer, copiar, clonar, etc.
- Proporcionar la capacidad de crear iconos personalizados para los componentes.
- Implementar la posibilidad de visualizar el código de cada componente individual.

Además de las mejoras mencionadas, existen diversas áreas que pueden explorarse en futuros desarrollos. Algunas de estas oportunidades de trabajo futuro son las siguientes:

- Implementar la capacidad de cargar la totalidad de la librería estándar Modelica. La aplicación actualmente se ha limitado a cargar las librerías Fluid, Block y Thermal, pero presenta la posibilidad de cargar las restantes librerías añadiendo sus archivos al directorio. Sin embargo, para una implementación más eficiente, podría considerarse una carga dinámica que solo cargue en memoria los componentes necesarios, evitando un consumo excesivo de recursos.
- Explorar la opción de cargar librerías de terceros directamente desde la aplicación. Actualmente, es posible cargar una librería externa, pero esta debe ubicarse en el mismo directorio que las demás librerías. Una mejora sería permitir la carga de librerías externas sin la necesidad de mover archivos manualmente.
- Habilitar la creación y gestión de múltiples modelos mediante pestañas. Esto permitiría a los usuarios trabajar en varios modelos de manera simultánea.
- Mejorar la manipulación de los componentes en el área de diseño, permitiendo ajustar el tamaño de los iconos, rotar e incluso cambiar su apariencia, cambiar los colores (propiedades gráficas).
- Implementar un analizador sintáctico del código generado para identificar posibles errores en el modelo planteado.
- Explorar la posibilidad de agregar los modelos diseñados o cargados en el árbol de componentes, con el objetivo de reutilizarlos para construir modelos más complejos, creando un "modelo de modelos".
- Refinar la edición de parámetros, proporcionando asistencia al usuario para garantizar la inserción de datos correctos o notificar cuando se introduzca un dato que no proceda, ya sea por incompatibilidad de tipos como por valores fuera de rango.

Cada uno de estos aspectos de mejora y oportunidades de trabajo futuro puede ser abordado en siguientes iteraciones, consolidando así la evolución continua de la aplicación y conduciendo a nuevas versiones más avanzadas y versátiles de la misma.

Bibliografía

[Apache Maven, 2023] Apache Maven (2023). Apache Maven - Gestor de dependencias para aplicaciones Java. https://maven.apache.org/. Accessed: 2023-09-01. 57

[Apache NetBeans, 2023] Apache NetBeans (2023). Apache NetBeans 19. https://netbeans.apache.org/. Accessed: 2023-08-15. 38

[Dassault-Systèmes, 2020] Dassault-Systèmes (2020). Dassault Systèmes. https://www.3ds.com/fileadmin/PRODUCTS/CATIA/DYMOLA/PDF/What-is-Dymola-2020x.pdf. Accessed: 2023-08-15. 23, 24

[Dassault-Systèmes, 2023] Dassault-Systèmes (2023). Página oficial Dymola. https://www.3ds.com/es/productos-y-servicios/catia/productos/dymola/. Accessed: 2023-08-15. 2, 16, 22

[Elmqvist et al., 1998] Elmqvist, H., Åström, K., and Mattsson, S. (1998). Evolution of continuous-time modeling and simulation. 12th European Simulation Multiconference, ESM'98 ; Conference date: 16-06-1998 Through 19-06-1998. 12

[Fehlberg, 1969] Fehlberg, E. (1969). *Low-order Classical Runge-Kutta Formulas with Stepsize Control and Their Application to Some Heat Transfer Problems*. NASA technical report. National Aeronautics and Space Administration. 13

[Friedl, 2006] Friedl, J. E. F. (2006). *Mastering Regular Expressions*. O'Reilly, Beijing, 3 edition. 36

[Fritzson, 2011] Fritzson, P. (2011). *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica*. Wiley-IEEE Press. 7, 8

[Gluon, 2023] Gluon (2023). JavaFX Scene Builder. https://gluonhq.com/products/scene-builder/. Accessed: 2023-08-18. 34, 71

[Grace, 1991] Grace, A. C. W. (1991). Simulab, an integrated environment for simulation and control. *1991 American Control Conference*, pages 1015–1020. 14

[Hilding, 1978] Hilding, E. (1978). A structured model language for large continuous systems. department of automatic control. *Lund University Sweden. ISRN LUTFD2/TFRT-1015-SE*. 22

[Larman and Valle, 2003] Larman, C. and Valle, B. (2003). *UML y patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado*. Pearson Educación. 28, 42

[Ljung and Glad, 1994] Ljung, L. and Glad, T. (1994). *Modeling of Dynamic Systems*. Prentice-Hall information and system sciences series. PTR Prentice Hall.

[Martin, 2000] Martin, R. C. (2000). Design principles and design patterns. *Object Mentor*, 1(34):597. 67

[Martin, 2012] Martin, R. C. (2012). *Código limpio : manual de estilo para el desarrollo ágil de software*. Anaya Multimedia. 76

[Modelica Association, 2001] Modelica Association (2001). Modelica Overview. https://modelica.org/documents/ModelicaOverview14.pdf. Accessed: 2023-09-09. 15, 16

[Modelica Association, 2023a] Modelica Association (2023a). Modelica—A unified object-oriented language for system modeling and simulation. https://modelica.org/documents/MLS.pdf. Accessed: 2023-09-10. 19, 21

[Modelica Association, 2023b] Modelica Association (2023b). The Modelica Association. https://modelica.org/. Accessed: 2023-08-15. 2, 15, 16

[OpenJFX, 2023] OpenJFX (2023). Documentación JavaFX. https://openjfx.io/. Accessed: 2023-08-18. 32, 109

[OpenModelica, 2023] OpenModelica (2023). Documentación de OpenModelica. https://openmodelica.org/useresresources/userdocumentation/. Accessed: 2023-08-16. 2, 16, 27, 29

[Oracle, 2023] Oracle (2023). Documentación Java. https://docs.oracle.com/en/java/. Accessed: 2023-08-18. 31

[Sokolowski and Banks, 2010] Sokolowski, J. and Banks, C. (2010). *Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains*. Wiley.

[Tiller, 2014] Tiller, M. M. (2014). Modelica by example. https://reference.wolfram.com/system-modeler/GettingStarted/ModelicaByExample.html.en.

[Urquía and Martín, 2016] Urquía, A. and Martín, C. (2016). *MÉTODOS DE SIMULACIÓN Y MODELADO*. Editorial UNED. 8, 9, 11, 13, 14, 19

[Wiegers, 2003] Wiegers, K. (2003). *Software Requirements*. Microsoft Press. 41

[Wolfram, 2023] Wolfram (2023). Documentación Wolfram System Modeler. https://reference.wolfram.com/system-modeler/. Accessed: 2023-08-16. 2, 24, 25

ANEXO A: MANUAL DE USUARIO

A-1. Instalación de FluidEditor v0.1

La aplicación FluidEditor v0.1 está desarrollada en el lenguaje de programación Java, utilizando JDK 19 y la librería gráfica JavaFX en su versión 19. En el caso de JavaFX, se ha empleado la versión de código abierto, conocida como OpenJavaFX. Para obtener más información, visite [OpenJFX, 2023].

Para la comodidad del usuario, la aplicación se ha empaquetado en un archivo ejecutable .JAR (por sus siglas en inglés, en inglés Java ARchive), eliminando así la necesidad de una instalación completa. Basta con tener el archivo ejecutable y ejecutarlo en un directorio con los permisos de ejecución adecuados.

Es importante tener en cuenta que en el mismo directorio debe existir la carpeta *lib/Modelica*, la cual contiene los archivos Modelica necesarios para que la aplicación funcione correctamente. Sin estos archivos, la aplicación se iniciará, pero no mostrará ningún componente Modelica, ya que necesita estos archivos para obtener información sobre cada uno de los componentes.

La Figura A.1 muestra los archivos distribuidos para la ejecución de FluidEditor v0.1. El primer archivo, *lib*, es un directorio que contiene los archivos Modelica de los paquetes *Fluid*, *Blocks*, *Thermal*, *Icons*, necesarios para el correcto funcionamiento de la aplicación. El siguiente archivo, *FluidEditor-1.0-SNAPSHOT.jar*, es una versión empaquetada de la aplicación sin las dependencias. Para que esta versión se ejecute correctamente, es necesario tener instalada y configurada la biblioteca Java conocida como JavaFX.

El tercer archivo, *FluidEditor-1.0-SNAPSHOT-jar-with-dependencies.jar*, es un archivo empaquetado que incluye todas las dependencias necesarias. Esto significa que no es necesario tener instalada la biblioteca JavaFX. Se recomienda utilizar esta versión para garantizar el correcto funcionamiento. Incluso si tiene una versión de JavaFX instalada, la aplicación utilizará sus propias dependencias incluidas en el paquete, lo que evitará problemas de incompatibilidad.

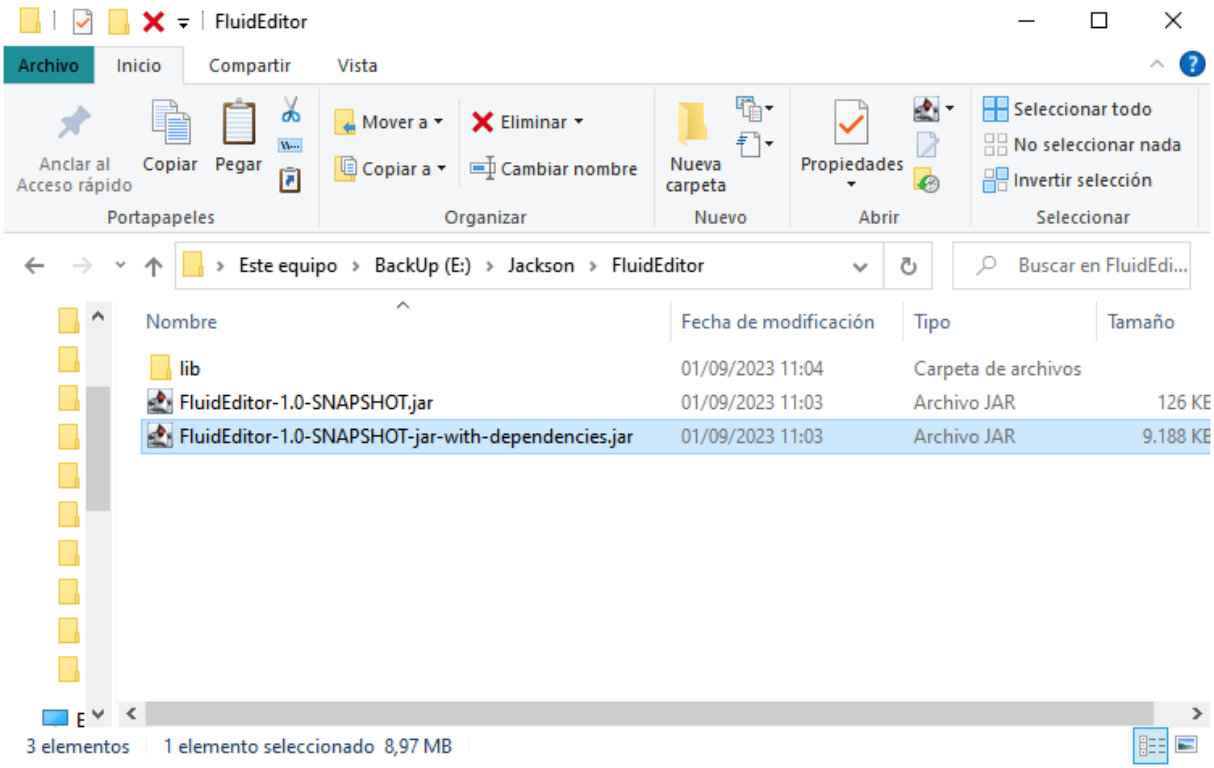

Figura A.1: Archivos empaquetados (.jar) de FluidEditor distribuidos para su ejecución.

Por último, es importante destacar que también existe la posibilidad de ejecutar la aplicación desde la consola de comandos, en este caso, el comando a utilizar sería el siguiente:

```
1   java -jar FluidEditor-1.0-SNAPSHOT-jar-with-dependencies.jar
```

Esta opción es útil para visualizar en la consola, cualquier mensaje de advertencia o error que pueda generar la aplicación durante su ejecución.

A-2. Interfaz de edición de modelos

Al iniciar la aplicación, notará que esta se divide en dos partes principales. En el lado izquierdo se encuentra el árbol de componentes Modelica, mientras que en el centro se ubica el lienzo de diseño, también conocido como área de diseño. Además, en la parte inferior derecha de esta área, encontrará dos pestañas que le permitirán alternar entre el modo de diseño y el modo de código. El último permite visualizar automáticamente el código Modelica generado a partir del modelo diseñado gráficamente.

A modo de demostración, crearemos un modelo simple compuesto por dos tanques conectados por una tubería. Para ello, siga estos pasos:

1. En el árbol de componentes, navegue hasta el nodo **Vessel**. Al expandirlo, verá varios componentes, incluido el que necesitamos: **OpenTank**. Selecciónelo y arrástrelo al área de diseño. Cuando lo suelte, aparecerá una ventana que le pedirá un nombre para el componente. Es importante destacar que este nombre debe ser único en todo el modelo y no debe contener espacios ni caracteres especiales. Repita este proceso para el segundo tanque.

2. Ahora, vaya nuevamente al árbol de componentes, al nodo **Pipes**, donde encontrará dos tipos de tuberías. Seleccione la tubería **StaticPipe** y arrástrela al área de diseño de manera similar a los pasos anteriores.

3. Otro componente necesario para cada modelo es el componente System. Repita el proceso anterior.

Una vez que tenga los tres componentes necesarios para este ejemplo, deberá realizar las conexiones. Siga estos pasos:

1. Mueva el cursor sobre uno de los conectores hasta que este cambie a una forma de cruz, lo que indicará que se trata de un conector del componente. Haga clic para comenzar a trazar la conexión.

2. Mueva el cursor hasta el conector de destino. En este punto, debería ver una cruz que indica que es un conector válido. Haga clic en el conector de destino para finalizar la conexión.

Después de completar la conexión, es posible que aparezca una nueva línea de conexión. En este caso, presione la tecla **ESC** para cancelar esa línea no deseada. Puede usar la misma tecla en cualquier momento durante el proceso de conexión para cancelar una conexión en curso.

Una vez finalizada la conexión, puede observar el código Modelica generado, cambiando al modo de código mediante la pestaña correspondiente. Puede ver estos pasos ilustrados en la Figura A.2 y la Figura A.3.

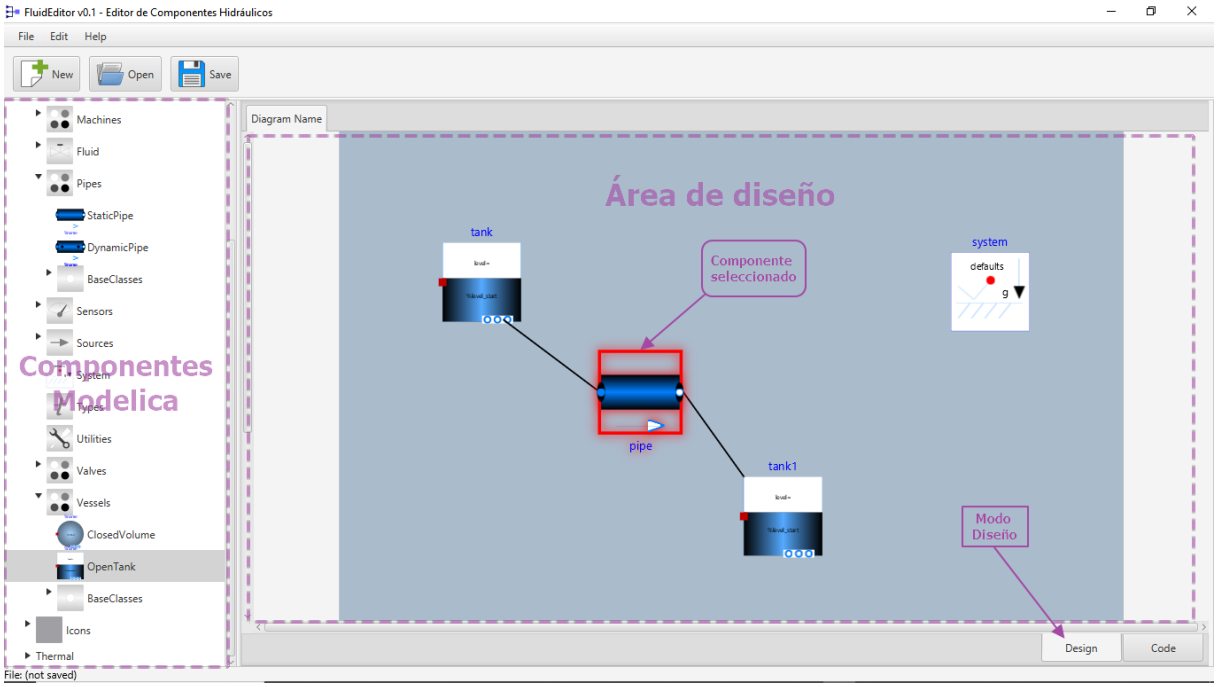

Figura A.2: Editando modelo de ejemplo en forma gráfica con FluidEditor.

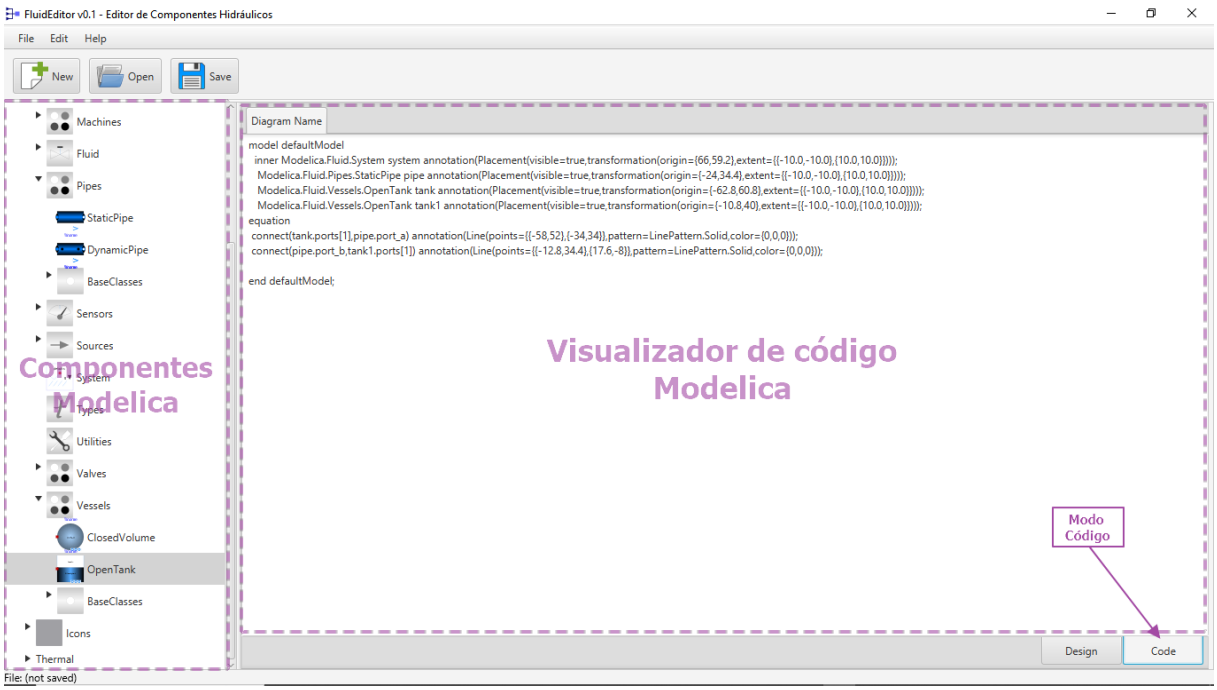

Figura A.3: Visualizando el código Modelica generado del modelo editado en FluidEditor.

A-3. Otras opciones de diseño

Para eliminar tanto un componente como una conexión, debes seleccionarlo con un clic. Esto resaltará sus bordes en color rojo, indicando que están seleccionados. Luego, puedes eliminar el componente o la conexión presionando la tecla **DEL**. Para mover un componente a la posición deseada, simplemente selecciona el componente y, sin soltar el clic, arrástralo a la ubicación deseada. Si deseas hacer zoom, mantén presionada la tecla **CTRL** y, al mismo tiempo, gira la rueda del ratón hacia adelante para reducir el zoom o hacia atrás para aumentarlo.

A-4. Editar parámetros de los componentes

Cada uno de los componentes del modelo que estamos diseñando admite parámetros que se pueden visualizar, modificar y actualizar simplemente haciendo doble clic sobre el componente de interés. Este proceso es similar al que se realiza en otros entornos de modelado y simulación, como OpenModelica.

Por ejemplo, si hacemos doble clic en el componente de la tubería, llamado **pipe** en el modelo de ejemplo de la sección anterior, se abrirá una nueva ventana que mostrará los parámetros del componente. La mayoría de estos parámetros estarán completados con valores por defecto, mientras que otros estarán vacíos y deberán ser rellenados obligatoriamente.

En la Figura A.4, se puede observar la ventana correspondiente a los parámetros de la tubería. Esta ventana está dividida en paneles, donde cada panel contiene los parámetros organizados en tres columnas: el nombre del parámetro, el valor editable y un comentario sobre dicho parámetro. Toda esta información se extrae del propio archivo Modelica en el que se describe el componente.

En este ejemplo, los parámetros que están vacíos y deben ser completados, en este caso son **length** (que corresponde a la longitud de la tubería) y **diameter** (que corresponde al diámetro de la tubería). De manera similar, se pueden configurar los demás parámetros en las diferentes pestañas. Estas configuraciones se pueden realizar para cada uno de los componentes.

Una vez que todos los parámetros hayan sido editados, se debe hacer clic en el botón "Guardar"para confirmar los valores. Esto cerrará la ventana y permitirá cambiar al modo de código para ver los cambios en código generado al modificar estos parámetros.

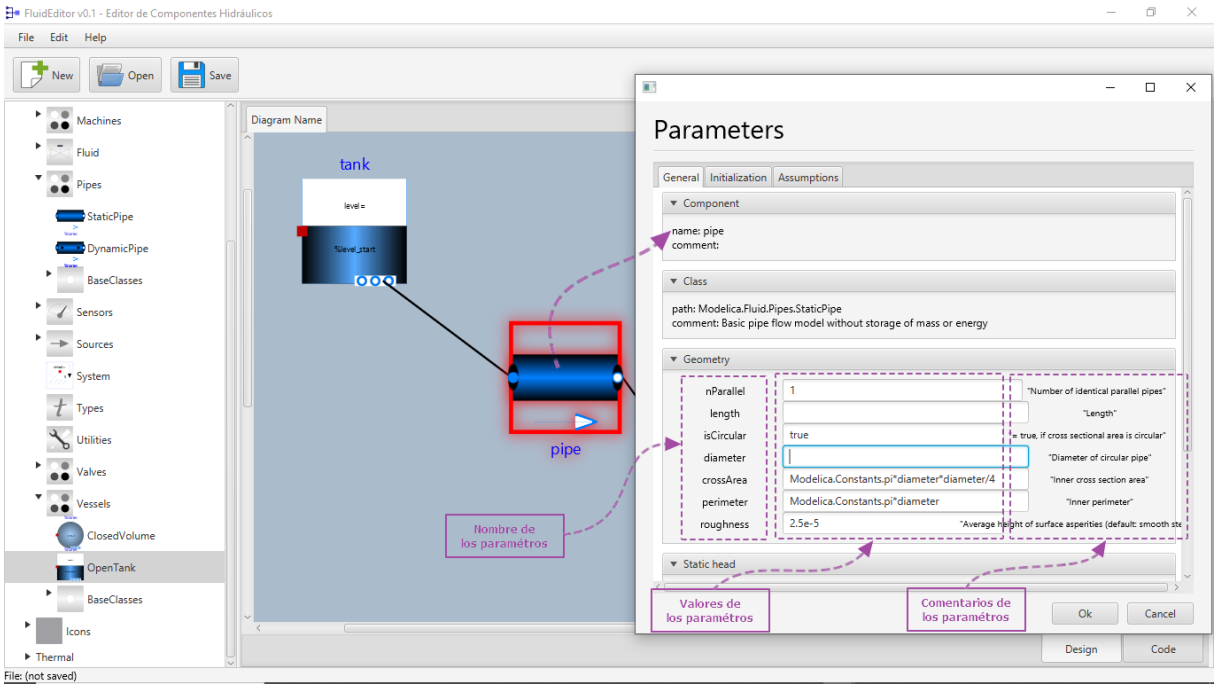

Figura A.4: Ejemplo de la edición de los parámetros de una tubería.

ANEXO B: CÓDIGO FUENTE

En este anexo se adjunta el código fuente de la aplicación **FluidEditor v0.1**. Cada sección corresponde a uno de los paquetes Java que implementan la Vista, el Controlador y el Modelo de la aplicación, siguiendo el patrón Modelo-Vista-Controlador (MVC), como se describe en la Figura 4.1. En las subsecciones se presenta el código de cada una de las clases que componen los paquetes correspondientes de cada sección. Estas clases se pueden observar en el árbol de directorio mostrado en la Figura 5.1 (parte izquierda).

B-1. Implementación de la vista

B-1.1. Código de la aplicación principal: App.java

```
1  package com.fluideditor.ui;
2
3  import com.fluideditor.controller.MainController;
4  import java.io.File;
5  import javafx.application.Application;
6  import javafx.fxml.FXMLLoader;
7  import javafx.scene.Parent;
8  import javafx.scene.Scene;
9  import javafx.stage.Stage;
10 import java.io.IOException;
11 import javafx.scene.image.Image;
12 import javafx.scene.image.ImageView;
13 import javafx.scene.layout.StackPane;
14
15 /**
16  * Clase principal. Arranca FluidEditor cargando los ficheros fxml de la
       GUI
17  *
18  * @author Jackson F. Reyes Bermeo
19  */
20 public class App extends Application {
21
22     private static Scene scene;
23     private static String rootPath;
24
25
26     @Override
27     public void start(Stage stage) throws IOException {
28         // Crear un icono para la ventana
29         Image iconImage = new Image(App.class.getResourceAsStream("icon.
               png"));
```

```
30          stage.getIcons().add(iconImage);
31
32          // Crear una animación de carga
33          stage.setScene(new Scene(new StackPane(new ImageView(iconImage))
                ,500,500));
34          stage.setTitle("Cargando FluidEditor...");
35          stage.show();
36
37          FXMLLoader fxmlLoader = loadFXML("MainView");
38          Parent rootParent = fxmlLoader.load();
39          MainController mainController = fxmlLoader.getController();
40          mainController.setRootPath(rootPath);
41          mainController.interact();
42
43          //inyectar la ventana de propiedades
44          MainController rootController = fxmlLoader.getController();
45          FXMLLoader loader = loadFXML("ParametersView");
46          Stage propitiesStage = new Stage();
47          propitiesStage.setScene(new Scene(loader.load()));
48          rootController.setPropertiesController(loader.getController());
49
50          // Configurar el evento de cierre en el controlador
51          stage.setOnCloseRequest(event -> {
52              mainController.confirmClose(event);
53          });
54
55          //cambiar de scena root
56          scene = new Scene(rootParent);
57          stage.setTitle("FluidEditor v0.1 - Editor de Componentes Hidrá
                ulicos");
58          stage.setScene(scene);
59          stage.setMaximized(true);
60      }
61
62      /**
63       * Cargar ficheros fxml. Permite cargar ficheros fxml que contentan
             la
64       * descripción de la GUI
65       * @param fxml Contiene el nombre del fichero sin extensión.
66       * @return FXMLLoader Traducción del FXML a objetos manejables por
             Java.
67       * @throws IOException Excepciones ocasionadas con la lectura del
             fichero.
68       */
69      private static FXMLLoader loadFXML(String fxml) throws IOException {
70          FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(
                fxml + ".fxml"));
71          return fxmlLoader;
72      }
73
74      /**
75       * Arranque de FluidEditor. Permite arrancar la aplicación
76       * @param args
77       */
78      public static void main(String[] args) {
79          rootPath = App.getRootPath();
80          launch();
81      }
82
83      /**
```

```
84      * Obtiene la ruta de la aplicación. Permite obtener la ruta padre
            en la que
85      * se va ejecutar la aplicación.
86      *
87      * @return Ruta padre del fichero ejecutable
88      */
89     public static String getRootPath() {
90         String excutableAppPath = App.class.getProtectionDomain().
               getCodeSource().getLocation().getPath();
91         // Si el archivo .jar se ejecuta desde un sistema Windows,
               elimina el primer caracter "/" de la ruta.
92         if (excutableAppPath.startsWith("/")) {
93             excutableAppPath = excutableAppPath.substring(1);
94         }
95         String parentExcutableAppPath = new File(excutableAppPath).
               getParentFile().getAbsolutePath();
96         return parentExcutableAppPath;
97     }
98 }
```

Código B.1: Implementación Java de la clase principal de la aplicación.

B-1.2. Código FXML de la Interfaz de Usuario (GUI): MainView.fxml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import javafx.geometry.Insets?>
4  <?import javafx.scene.control.Button?>
5  <?import javafx.scene.control.Menu?>
6  <?import javafx.scene.control.MenuBar?>
7  <?import javafx.scene.control.MenuItem?>
8  <?import javafx.scene.control.ScrollPane?>
9  <?import javafx.scene.control.Separator?>
10 <?import javafx.scene.control.SeparatorMenuItem?>
11 <?import javafx.scene.control.SplitPane?>
12 <?import javafx.scene.control.Tab?>
13 <?import javafx.scene.control.TabPane?>
14 <?import javafx.scene.control.TextArea?>
15 <?import javafx.scene.control.TreeView?>
16 <?import javafx.scene.image.Image?>
17 <?import javafx.scene.image.ImageView?>
18 <?import javafx.scene.input.KeyCodeCombination?>
19 <?import javafx.scene.layout.AnchorPane?>
20 <?import javafx.scene.layout.BorderPane?>
21 <?import javafx.scene.layout.HBox?>
22 <?import javafx.scene.layout.Pane?>
23 <?import javafx.scene.layout.StackPane?>
24 <?import javafx.scene.layout.VBox?>
25 <?import javafx.scene.text.Text?>
```

```
26
27 <StackPane fx:controller="com.fluideditor.controller.MainController"
       maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
       minWidth="-Infinity" prefHeight="760.0" prefWidth="1024.0" xmlns="
       http://javafx.com/javafx/19" xmlns:fx="http://javafx.com/fxml/1" >
28 <children>
29 <BorderPane prefHeight="200.0" prefWidth="200.0">
30 <top>
31 <VBox>
32 <children>
33 <Separator prefWidth="200.0" />
34 <MenuBar>
35 <menus>
36 <Menu mnemonicParsing="false" text="File">
37 <items>
38 <MenuItem mnemonicParsing="false" onAction="#newButtonAction" text="New
       Model">
39  <graphic>
40     <ImageView fitHeight="30.0" fitWidth="30.0" pickOnBounds="true"
           preserveRatio="true">
41        <image>
42           <Image url="@../controller/new.png" />
43        </image>
44     </ImageView>
45  </graphic>
46  <accelerator>
47     <KeyCodeCombination alt="UP" code="N" control="DOWN" meta="UP" shift
           ="UP" shortcut="UP" />
48  </accelerator>
49 </MenuItem>
50 <SeparatorMenuItem mnemonicParsing="false" />
51 <MenuItem mnemonicParsing="false" onAction="#openButtonAction" text="
       Open Model">
52  <graphic>
53     <ImageView fitHeight="30.0" fitWidth="30.0" pickOnBounds="true"
           preserveRatio="true">
54        <image>
55           <Image url="@../controller/open.png" />
56        </image>
57     </ImageView>
58  </graphic>
59  <accelerator>
60     <KeyCodeCombination alt="UP" code="O" control="DOWN" meta="UP" shift
           ="UP" shortcut="UP" />
61  </accelerator>
62 </MenuItem>
63 <SeparatorMenuItem mnemonicParsing="false" />
```

```
<MenuItem mnemonicParsing="false" onAction="#saveButtonAction" text="
    Save Model">
 <graphic>
    <ImageView fitHeight="30.0" fitWidth="30.0" pickOnBounds="true"
        preserveRatio="true">
       <image>
          <Image url="@../controller/save.png" />
       </image>
    </ImageView>
 </graphic>
 <accelerator>
    <KeyCodeCombination alt="UP" code="S" control="DOWN" meta="UP" shift
        ="UP" shortcut="UP" />
 </accelerator>
</MenuItem>
<MenuItem mnemonicParsing="false" onAction="#saveAsButtonAction" text="
    Save As... Model">
 <graphic>
    <ImageView fitHeight="30.0" fitWidth="30.0" pickOnBounds="true"
        preserveRatio="true">
       <image>
          <Image url="@../controller/saveas.png" />
       </image>
    </ImageView>
 </graphic>
 <accelerator>
    <KeyCodeCombination alt="UP" code="A" control="DOWN" meta="UP" shift
        ="UP" shortcut="UP" />
 </accelerator>
</MenuItem>
<SeparatorMenuItem mnemonicParsing="false" />
<MenuItem mnemonicParsing="false" onAction="#confirmClose" text="Quit">
 <graphic>
    <ImageView fitHeight="30.0" fitWidth="30.0" pickOnBounds="true"
        preserveRatio="true">
       <image>
          <Image url="@../controller/quit.png" />
       </image>
    </ImageView>
 </graphic>
 <accelerator>
    <KeyCodeCombination alt="UP" code="X" control="DOWN" meta="UP" shift
        ="UP" shortcut="UP" />
 </accelerator></MenuItem>
</items>
</Menu>
<Menu mnemonicParsing="false" text="Edit">
```

```
<items>
<MenuItem mnemonicParsing="false" onAction="#deleteModelItemAction" text
    ="Delete Model">
 <graphic>
    <ImageView fitHeight="25.0" fitWidth="25.0" pickOnBounds="true"
        preserveRatio="true">
       <image>
          <Image url="@../controller/delete.png" />
       </image>
    </ImageView>
 </graphic></MenuItem>
</items>
</Menu>
<Menu mnemonicParsing="false" text="Help">
<items>
<MenuItem mnemonicParsing="false" onAction="#onAboutItemAction" text="
    About">
 <graphic>
    <ImageView fitHeight="25.0" fitWidth="25.0" pickOnBounds="true"
        preserveRatio="true">
       <image>
          <Image url="@../controller/icon.png" />
       </image>
    </ImageView>
 </graphic></MenuItem>
</items>
</Menu>
</menus>
</MenuBar>
<HBox prefHeight="49.0" prefWidth="1024.0" spacing="10.0">
<children>
<Button fx:id="newButton" minHeight="40.0" minWidth="30.0"
    mnemonicParsing="false" onAction="#newButtonAction" text="New" />
<Button fx:id="openButton" minHeight="40.0" minWidth="30.0"
    mnemonicParsing="false" onAction="#openButtonAction" text="Open" />
<Button fx:id="saveButton" minHeight="40.0" minWidth="30.0"
    mnemonicParsing="false" onAction="#saveButtonAction" text="Save" />
</children>
<padding>
<Insets bottom="6.0" left="10.0" right="6.0" top="10.0" />
</padding>
</HBox>
<Separator prefWidth="200.0" />
</children>
</VBox>
</top>
<bottom>
```

```
<HBox prefHeight="9.0" prefWidth="1024.0" BorderPane.alignment="CENTER">
<children>
<Text fx:id="textModelSavedPath" strokeType="OUTSIDE" strokeWidth="0.0"
    text="File: (not saved)" wrappingWidth="1016.548828125">
<HBox.margin>
<Insets bottom="2.0" />
</HBox.margin>
</Text>
</children></HBox>
</bottom>
<center>
<HBox nodeOrientation="LEFT_TO_RIGHT" BorderPane.alignment="CENTER">
<children>
<SplitPane dividerPositions="0.09" nodeOrientation="LEFT_TO_RIGHT" HBox.
    hgrow="ALWAYS">
<items>
<TreeView fx:id="fluidTreeView" />
<VBox alignment="CENTER" nodeOrientation="LEFT_TO_RIGHT">
<children>
<TabPane nodeOrientation="LEFT_TO_RIGHT" tabClosingPolicy="ALL_TABS"
    tabMinHeight="28.0" VBox.vgrow="ALWAYS">
<tabs>
  <Tab fx:id="tabDiagram" closable="false" text="Diagram Name">
    <content>
          <TabPane fx:id="internalTabPane" nodeOrientation="
              RIGHT_TO_LEFT" prefHeight="673.0" prefWidth="824.0" side="
              BOTTOM" tabClosingPolicy="UNAVAILABLE" tabMinHeight="30.0"
               tabMinWidth="80.0">
            <tabs>
              <Tab onSelectionChanged="#onCodeViewAction" text="Code">
                <content>
                  <AnchorPane>
                         <children>
                            <TextArea fx:id="rootCodeArea" editable="
                                false" layoutX="565.0" layoutY="47.0"
                                nodeOrientation="LEFT_TO_RIGHT"
                                prefHeight="573.0" prefWidth="824.0"
                                promptText="//code" AnchorPane.
                                bottomAnchor="0.0" AnchorPane.leftAnchor
                                ="0.0" AnchorPane.rightAnchor="0.0"
                                AnchorPane.topAnchor="0.0" />
                         </children>
                      </AnchorPane>
                </content>
              </Tab>
              <Tab fx:id="tabDesign" text="Design">
                   <content>
```

```
177                         <ScrollPane fx:id="designScrollPane" fitToHeight="
                                true" fitToWidth="true">
178                           <content>
179                               <StackPane>
180                                   <children>
181                                       <Pane fx:id="designPane" maxHeight="
                                              500.0" maxWidth="500.0" minHeight=
                                              "500.0" minWidth="500.0"
                                              nodeOrientation="LEFT_TO_RIGHT"
                                              prefHeight="500.0" prefWidth="
                                              500.0" style="-fx-background-
                                              color: #ABC;" />
182                                   </children>
183                               </StackPane>
184                           </content>
185                         </ScrollPane>
186                     </content>
187                 </Tab>
188               </tabs>
189             </TabPane>
190       </content>
191     </Tab>
192 </tabs>
193  <VBox.margin>
194       <Insets right="5.0" />
195  </VBox.margin>
196 </TabPane>
197 </children>
198 </VBox>
199 </items>
200 </SplitPane>
201 </children>
202 </HBox>
203 </center>
204 </BorderPane>
205 </children>
206 </StackPane>
```

Código B.2: Contenido del fichero MainView.fxml que describe la Interfaz Gráfica de Usuario (GUI) principal.

B-1.3. Código FXML de la Interfaz de visualización de parámetros: ParametersView.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.ButtonBar?>
<?import javafx.scene.control.ScrollPane?>
<?import javafx.scene.control.Separator?>
<?import javafx.scene.control.Tab?>
<?import javafx.scene.control.TabPane?>
<?import javafx.scene.control.TitledPane?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.layout.StackPane?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<AnchorPane id="AnchorPane" prefHeight="600.0" prefWidth="800.0" xmlns="
    http://javafx.com/javafx/19" xmlns:fx="http://javafx.com/fxml/1"
    fx:controller="com.fluideditor.controller.PropertiesViewController">
   <children>
      <BorderPane prefHeight="500.0" prefWidth="700.0" AnchorPane.
          bottomAnchor="0.0" AnchorPane.leftAnchor="0.0" AnchorPane.
          rightAnchor="0.0" AnchorPane.topAnchor="0.0">
         <bottom>
            <ButtonBar prefHeight="31.0" prefWidth="700.0" BorderPane.
                alignment="CENTER">
              <buttons>
                <Button fx:id="saveParametersBtn" mnemonicParsing="false
                    " onAction="#saveParameters" text="Ok" />
                  <Button fx:id="cancelParameters" mnemonicParsing="
                      false" onAction="#cancelParameters" text="Cancel"
                      />
              </buttons>
               <padding>
                  <Insets bottom="10.0" right="20.0" top="10.0" />
               </padding>
            </ButtonBar>
         </bottom>
         <top>
            <VBox prefHeight="42.0" prefWidth="700.0" spacing="5.0"
                BorderPane.alignment="CENTER">
              <children>
```

```
35                  <Text strokeType="OUTSIDE" strokeWidth="0.0" text="
                        Parameters">
36                     <font>
37                        <Font size="30.0" />
38                     </font>
39                  </Text>
40                  <Separator prefWidth="200.0" />
41               </children>
42               <BorderPane.margin>
43                  <Insets />
44               </BorderPane.margin>
45               <padding>
46                  <Insets bottom="10.0" left="20.0" right="20.0" top="
                        10.0" />
47               </padding>
48            </VBox>
49         </top>
50         <center>
51            <StackPane BorderPane.alignment="CENTER">
52               <children>
53                  <TabPane fx:id="propertiesTabPane" prefHeight="800.0"
                        tabClosingPolicy="UNAVAILABLE">
54                    <tabs>
55                      <Tab closable="false" text="General">
56                        <content>
57                              <ScrollPane fitToHeight="true" fitToWidth=
                                    "true">
58                                 <content>
59                                    <VBox style="-fx-background-color:
                                          white;">
60                                       <children>
61                                          <TitledPane animated="false"
                                                minWidth="600.0" prefWidth
                                                ="600.0" text="Component"
                                                textOverrun="
                                                LEADING_ELLIPSIS">
62                                             <padding>
63                                                <Insets bottom="5.0"
                                                      left="10.0" right="
                                                      10.0" top="5.0" />
64                                             </padding>
65                                             <content>
66                                                <VBox>
67                                                   <children>
68                                                      <Text strokeType="
                                                            OUTSIDE"
                                                            strokeWidth="
```

```
                                                    0.0" text="
                                                    name:" />
69                                                <Text strokeType="
                                                    OUTSIDE"
                                                    strokeWidth="
                                                    0.0" text="
                                                    Comment:" />
70                                             </children>
71                                          </VBox>
72                                       </content>
73                                    </TitledPane>
74                                    <TitledPane graphicTextGap="
                                         2.0" text="Class">
75                                       <padding>
76                                          <Insets bottom="5.0"
                                               left="10.0" right="
                                               10.0" top="5.0" />
77                                       </padding>
78                                       <content>
79                                          <VBox prefHeight="52.0"
                                               prefWidth="481.0">
80                                             <children>
81                                                <Text strokeType="
                                                    OUTSIDE"
                                                    strokeWidth="
                                                    0.0" text="
                                                    Path:" />
82                                                <Text strokeType="
                                                    OUTSIDE"
                                                    strokeWidth="
                                                    0.0" text="
                                                    Comment:" />
83                                             </children>
84                                          </VBox>
85                                       </content>
86                                    </TitledPane>
87                                 </children>
88                              </VBox>
89                           </content>
90                        </ScrollPane>
91                     </content>
92                  </Tab>
93               </tabs>
94            </TabPane>
95         </children>
96         <padding>
97            <Insets left="20.0" right="20.0" />
```

```
98                </padding>
99             </StackPane>
100          </center>
101       </BorderPane>
102    </children>
103 </AnchorPane>
```

Código B.3: Contenido del fichero ParametersView.fxml que describe la Interfaz de visualización de parámetros.

B-2. Implementación del controlador

B-2.1. Código del controlador principal: MainController.java

```
1  package com.fluideditor.controller;
2
3  import com.fluideditor.model.modelica.ComponentModel;
4  import com.fluideditor.model.modelica.Model;
5  import com.fluideditor.model.modelica.ModelManager;
6  import com.fluideditor.model.modelica.ModelicaConnection;
7  import com.fluideditor.model.modelica.ModelicaConnector;
8  import com.fluideditor.model.modelica.ModelicaParameter;
9  import com.fluideditor.model.tree.NodeItemCode;
10 import com.fluideditor.model.icon.*;
11 import com.fluideditor.model.tree.ModelicaAnalizer;
12 import java.io.File;
13 import java.io.FileWriter;
14 import java.io.IOException;
15 import java.util.ArrayList;
16 import java.util.List;
17 import java.util.Map;
18 import java.util.Random;
19 import javafx.event.ActionEvent;
20 import javafx.event.Event;
21 import javafx.fxml.FXML;
22 import javafx.geometry.Point2D;
23 import javafx.scene.Group;
24 import javafx.scene.Node;
25 import javafx.scene.control.Alert;
26 import javafx.scene.control.Alert.AlertType;
27 import javafx.scene.control.Button;
28 import javafx.scene.control.ButtonType;
29 import javafx.scene.control.ScrollPane;
30 import javafx.scene.control.SingleSelectionModel;
31 import javafx.scene.control.Tab;
32 import javafx.scene.control.TabPane;
33 import javafx.scene.control.TextArea;
34 import javafx.scene.control.TextInputDialog;
35 import javafx.scene.control.TreeCell;
36 import javafx.scene.control.TreeItem;
37 import javafx.scene.control.TreeView;
38 import javafx.scene.image.Image;
39 import javafx.scene.image.ImageView;
```

```
40 import javafx.scene.input.ClipboardContent;
41 import javafx.scene.input.DragEvent;
42 import javafx.scene.input.Dragboard;
43 import javafx.scene.input.KeyCode;
44 import javafx.scene.input.MouseButton;
45 import javafx.scene.input.MouseEvent;
46 import javafx.scene.input.ScrollEvent;
47 import javafx.scene.input.TransferMode;
48 import javafx.scene.layout.Pane;
49 import javafx.scene.layout.StackPane;
50 import javafx.scene.paint.Color;
51 import javafx.scene.shape.Line;
52 import javafx.scene.shape.Polygon;
53 import javafx.scene.shape.Shape;
54 import javafx.scene.text.Text;
55 import javafx.scene.transform.Scale;
56 import javafx.stage.DirectoryChooser;
57 import javafx.stage.FileChooser;
58
59 /**
60  * FXML Controller class. Controlador de la GUI principal de la aplicaci
        ón.
61  * Realiza la lógica de la aplicación
62  *
63  * @author Jackson F. Reyes Bermeo
64  */
65 public class MainController {
66
67     @FXML
68     TreeView<NodeItemCode> fluidTreeView;
69     @FXML
70     ScrollPane designScrollPane;
71     @FXML
72     Pane designPane;
73     @FXML
74     private TabPane internalTabPane;
75     @FXML
76     private Tab tabDesign;
77     @FXML
78     private TextArea rootCodeArea;
79     @FXML
80     private Button openButton;
81     @FXML
82     private Button newButton;
83     @FXML
84     private Button saveButton;
85     @FXML
86     private Tab tabDiagram;
87     @FXML
88     private Text textModelSavedPath;
89
90     private ModelicaAnalizer modelicaAnalizer;//Analizador de ficheros m
           ódilica para extraer el arbol de componentes
91     private PropertiesViewController propertiesController; //Controlador
            de las propiedades de cada componente
92     private Model rootModel; // El modelo actual
93     private String rootPath; // Ruta padre del ejecutable
94     private Pane selectedPane; // El actual icono seleccionado en el
           dise o
95     private Line selectedLine; // La linea de conexión seleccionada
96     private double startX, startY; // para dibujar la linea
```

```
97      private Line currentLineToConnect;//Linea temporal que se va
            redibujando cuando se hace la conexión
98      private Shape currentConnectIconAnnotation; //El componente actual
            durante la conexión
99      private ModelicaConnection modelicaConnection = null; // Representa
            la conexión
100     private boolean isDrawingLine = false;
101     private static final double ZOOM_FACTOR = 1.1; // Ajusta este valor
            para controlar la velocidad del zoom
102     private static final double SCALE_VIEW_ICON = 2.5; //Escalado del
            icono para mejorar la visualización
103     private int countClicks = 0;
104     private final boolean DEBUG = false; // para el desarrollador
105
106     public void setRootPath(String rootPath) {
107         this.rootPath = rootPath;
108     }
109
110     /**
111      * Inicialización del controlador previa a la carga de la GUI.
112      */
113     public void initialize() {
114         this.initialGUIConfigure();
115         this.selectInitialTabToShow(tabDesign);
116     }
117
118     /**
119      * *
120      * Gestiona la carga de la libreria y distintas configuraciones de
              eventos.
121      */
122     public void interact() {
123         if (DEBUG) {
124             System.out.println("Controller ready!\nRuta: " + rootPath);
125         }
126         rootModel = new Model("defaultModel");//Modelo por defecto
127         File rootDirectory = new File(rootPath, "/lib/Modelica");
128         if (rootDirectory.exists()) {
129             this.makeTreeView(rootDirectory.toString()); // Cargar el
                    arbol de directorio
130         } else {
131             if (DEBUG) {
132                 System.out.println("No existe la libreria Modelica, ruta
                        actual: " + rootDirectory.toString());
133             }
134             manageModelicaLibPathNotFound();
135         }
136         this.configureTreeViewEvents(); // Registrar eventos para el
                treeView
137         this.configureDesignPaneEvents();
138         this.configureScrollPane(); // Ajuste del zoom
139     }
140
141     private void manageModelicaLibPathNotFound() {
142         Alert alert = new Alert(Alert.AlertType.WARNING);
143         alert.setTitle("Libreria Fluid no encontrada");
144         alert.setHeaderText(" Desea  buscar la ruta de la libreria?");
145         alert.setContentText("Para evitar esta advertencia,por favor,
                copie dicha libreria al mismo nivel del ejecutable con el
                siguiente formato:\n \\lib\\Modelica\\");
146         ButtonType buttonTypeYes = new ButtonType("Aceptar");
```

```
147         ButtonType buttonTypeNo = new ButtonType("Cancelar");
148         alert.getButtonTypes().setAll(buttonTypeYes, buttonTypeNo);
149         alert.showAndWait().ifPresent(buttonType -> {
150             if (buttonType == buttonTypeYes) {// Si el usuario elige "
                    Aceptar"
151                 createSearchLibPathDialog();
152             }
153         });
154     }
155 
156     private void createSearchLibPathDialog() {
157         DirectoryChooser directoryChooser = new DirectoryChooser();
158         directoryChooser.setTitle("Seleccionar directorio lib\\Modelica"
                );
159         File modelicaLibPath = directoryChooser.showDialog(null);
160         if (modelicaLibPath != null) {
161             this.makeTreeView(modelicaLibPath.getAbsolutePath()); //
                    cargar el arbol de directorio
162         }
163     }
164 
165     public void setPropertiesController(PropertiesViewController
            controller) {
166         this.propertiesController = controller;
167     }
168 
169     private void configureScrollPane() {
170         designScrollPane.setPannable(true);
171         designScrollPane.setFitToHeight(true);
172         designScrollPane.setFitToWidth(true);
173         designScrollPane.setVbarPolicy(ScrollPane.ScrollBarPolicy.ALWAYS
                ); // Mostrar barra de desplazamiento vertical
174         designScrollPane.setHbarPolicy(ScrollPane.ScrollBarPolicy.ALWAYS
                ); // Mostrar barra de desplazamiento horizontal
175         final Group scroller = new Group(designPane);// hay que
                envolverlo en un grupo sino no funciona!!!
176         StackPane scrollContent = new StackPane(scroller);
177         designScrollPane.setContent(scrollContent);
178         designPane.addEventFilter(ScrollEvent.ANY, event -> {// Agregar
                manejo de eventos para el zoom
179             if (event.isControlDown()) {
180                 double zoomFactor = event.getDeltaY() > 0 ? ZOOM_FACTOR
                        : 1 / ZOOM_FACTOR;
181                 designPane.setScaleX(designPane.getScaleX() * zoomFactor
                        );
182                 designPane.setScaleY(designPane.getScaleY() * zoomFactor
                        );
183                 designScrollPane.requestLayout(); // Actualizar las
                        barras de desplazamiento
184                 event.consume();
185             }
186         });
187     }
188 
189     private void configureDesignPaneEvents() {
190         designPane.setFocusTraversable(true);// Foco para recibir
                eventos de teclado
191         designPane.setOnDragOver((DragEvent event) -> { //Aceptar
                elementos arrastrables
192             if (event.getGestureSource() != designPane && event.
                    getDragboard().hasString()) {
```

```
193                 event.acceptTransferModes(TransferMode.COPY);
194             }
195             event.consume();
196         });
197 
198         designPane.setOnDragDropped((DragEvent event) -> {// cuando se
                suelta el componente
199             Dragboard dragboard = event.getDragboard();
200             boolean success = false;
201             if (dragboard.hasString()) {
202                 DraggableNode dragNode;
203                 NodeItemCode nodeItem = this.getNodeByRute(fluidTreeView
                        .getRoot(), dragboard.getString());
204                 IconManager iconManager = new IconManager(fluidTreeView.
                        getRoot(), nodeItem, rootPath);
205                 IconAnnotation completeIcon = iconManager.
                        getCompleteIconAnnotation();
206                 ModelManager modelicaManager = new ModelManager(
                        fluidTreeView.getRoot(), nodeItem, rootPath);
207                 ModelicaParameter parameter = modelicaManager.
                        getAllParameterModel();// crear la clase parameters
208 
209                 for (ShapeAnnotation shape : completeIcon.getShapes()) {
                         // dialogo asignación nombre componente
210                     if (shape instanceof TextAnnotation) {
211                         String textString = ((TextAnnotation) shape).
                                getTextString();
212                         if (textString.contains("%name")) {
213                             String nameComponent = parameter.
                                    getDefaultComponentName();
214                             int nElem = rootModel.
                                    existElementOfModelCompositionByName(
                                    nameComponent);
215                             if (nElem > 0) {
216                                 nameComponent = nameComponent + nElem;
217                             }
218                             String name = setComponentNameByDialog(
                                    nameComponent);
219                             ((TextAnnotation) shape).setTextString(name)
                                    ;
220                             parameter.setNameComponent(name);
221                         }
222                     }
223                 }
224                 String id = "" + parameter.getPath() + parameter.
                        getNameComponent();
225                 parameter.setId(id);
226                 if (DEBUG) {
227                     System.out.println("<<<>>>ID parameter: " + id);
228                 }
229 
230                 //Configuración del icono de visualización en el dise o
231                 CoordinateSystem iconCoordinate = completeIcon.
                        getCoordinateSystem();
232                 double scale = SCALE_VIEW_ICON * iconCoordinate.
                        getInitialScale();
233                 double widthIcon = scale * iconCoordinate.getExtent().
                        getWidth();
234                 double heightIcon = scale * iconCoordinate.getExtent().
                        getHeight();
235                 Node icon = completeIcon.getIcon();
```

```
236                 icon.getTransforms().add(new Scale(scale, scale));
237                 dragNode = new DraggableNode(icon);
238                 dragNode.setId(id);
239                 dragNode.setName(parameter.getNameComponent());
240                 dragNode.setPrefSize(widthIcon + 4, heightIcon + 4);
241                 dragNode.setLayoutX(event.getX() - (dragNode.
                        getPrefWidth() / 2));
242                 dragNode.setLayoutY(event.getY() - (dragNode.
                        getPrefHeight() / 2));
243                 dragNode.setStyle("-fx-background-color: transparent; ")
                        ;
244                 designPane.getChildren().add(dragNode);
245
246                 // Modificación de la visualización del componente
247                 Placement placement = parameter.getPlacement();
248                 Transformation transformation = placement.
                        getTransformation();
249                 Point2D origin = transformFromSystemToModelicaCoordinate
                        (event.getX(), event.getY());
250                 transformation.setOrigin(origin);
251                 double leftWidth = -widthIcon / (2 * SCALE_VIEW_ICON);
252                 double topHeight = -heightIcon / (2 * SCALE_VIEW_ICON);
253                 transformation.setExtent(new Extent(leftWidth, topHeight
                        , -leftWidth, -topHeight));
254                 parameter.setPlacement(placement);
255                 dragNode.setPlacement(placement);
256                 if (DEBUG) {
257                     System.out.println("origin: " + origin.toString() +
                            " width: " + widthIcon / (2 * SCALE_VIEW_ICON) +
                             " height: " + heightIcon / (2 * SCALE_VIEW_ICON
                            ));
258                 }
259                 List<ModelicaConnector> connectors = iconManager.
                        getConnectors();
260                 for (ModelicaConnector connector : connectors) {
261                     //connector.setId(id + connector.getType());//
                            vincular el id con el de su padre
262                     //Test: cambiando el id para a adie el nombre del
                            connector
263                     connector.setId(id + connector.getType()+"::"+
                            connector.getName());//vincular el id con el de
                            su padre
264                     connector.setParent(parameter.getNameComponent());//
                             estableciendo los padres
265                 }
266                 dragNode.setConnectors(connectors);
267                 this.rootModel.addElementOfModelComposition(parameter);
268                 success = true;
269             }
270             event.setDropCompleted(success);
271             event.consume();
272         });
273
274         designPane.setOnMouseClicked(event -> {
275             Node node = (Node) event.getTarget();
276             if (DEBUG) {
277                 System.out.println("node: " + node.toString() + "\
                        nparent: " + node.getParent());
278             }
279             if (node instanceof DraggableNode) {
280                 Pane selectedNode = (Pane) node;
```

```
281                     setSelectedComponent(selectedNode);
282                 } else if (node != null && node.getParent() instanceof
                        IconAnnotation) {
283                     Pane selectedNode = (Pane) node.getParent().getParent();
284                     setSelectedComponent(selectedNode);
285                 } else if (node instanceof Polygon) {
286                     Pane selectedNode = (Pane) node.getParent().getParent().
                            getParent();
287                     setSelectedComponent(selectedNode);
288                 } else {
289                     setSelectedComponent(null);
290                 }
291 
292                 if (node instanceof Line) {// seleccionar la linea de conexi
                        ón
293                     setSelectedLineConnection((Line) node);
294                 } else {
295                     setSelectedLineConnection(null);
296                 }
297                 // Con dos clic abre las propiedades del elemento
                        seleccionado (1 click)
298                 if (event.getClickCount() == 2 && event.getButton() ==
                        MouseButton.PRIMARY && selectedPane != null) {
299                     String id = selectedPane.getId();
300                     if (DEBUG) {
301                         System.out.println("<<<>>>ID parameter: " + id);
302                     }
303                     ModelicaParameter modelicaParameters = rootModel.
                            getElementOfModelCompositionById(id);
304                     propertiesController.setParameters(modelicaParameters);
305                     propertiesController.showParamiter(modelicaParameters);
306                 }
307 
308                 if (event.getTarget() instanceof Shape) {// Evento para las
                        conecciones entre componentes
309                     Shape tempShape = (Shape) event.getTarget();
310                     boolean isConnector = isConnector(tempShape);
311                     if (isConnector && currentLineToConnect == null &&
                            isDrawingLine == false &&
                            currentConnectIconAnnotation != tempShape) {
312                         countClicks++;
313                         if (DEBUG) {
314                             System.out.println(">>>> Clicked " + countClicks
                                    );
315                         }
316                         currentConnectIconAnnotation = tempShape;
317                         currentLineToConnect = new Line();
318                         currentLineToConnect.setMouseTransparent(true);//
                                Importante, si no siempre el target es la linea
319                         currentLineToConnect.setStroke(Color.BLACK);
320                         startX = event.getX();
321                         startY = event.getY();
322                         currentLineToConnect.setStartX(startX);
323                         currentLineToConnect.setStartY(startY);
324                         currentLineToConnect.setEndX(event.getX());
325                         currentLineToConnect.setEndY(event.getY());
326                         designPane.getChildren().add(currentLineToConnect);
327                         DraggableNode tempDragNode =
                                getDraggableNodeContainer(tempShape);
328                         bindLineToShape(tempDragNode, "init", event.getX(),
                                event.getY()); //Para que se mueva la conexión
```

```
                    junto al componente
329                 isDrawingLine = true;
330                 String routeType = getRouteConnectorByShape(
                        tempShape);
331                 //ModelicaConnector selectedConnector = tempDragNode
                        .getConnectorByType(routeType);
332                 //Test: cambiando la forma de obtener el conector
333
334                 String connectorName = getNameConnectorByShape(
                        tempShape);
335                 ModelicaConnector selectedConnector = tempDragNode.
                        getConnectorByName(connectorName);
336
337                 if (DEBUG) {
338                     System.out.println("--> connector: " + routeType
                            );
339                     System.out.println("-- Model: " +
                            selectedConnector.getParent() + "." +
                            selectedConnector.getName());
340                     System.out.println("Position: " + event.getX() +
                             "," + event.getY() + " -> layoutDragNode: "
                             + tempDragNode.getLayoutX() + "," +
                            tempDragNode.getLayoutY());
341                 }
342                 Random random = new Random();// Generar un número
                        aleatorio
343                 int randomNumber = random.nextInt(1000); // Número
                        aleatorio entre 0 y 999
344                 currentLineToConnect.setId("lineConnection::" +
                        routeType + "::" + selectedConnector.getParent()
                         + "." + selectedConnector.getName() + "::" +
                        randomNumber);
345                 if (modelicaConnection == null) {
346                     modelicaConnection = new ModelicaConnection(
                            currentLineToConnect);
347                     modelicaConnection.setId(currentLineToConnect.
                            getId()); // para vincular la linea con la
                            lineaAnnotation
348                     modelicaConnection.setLineConnection(new
                            LineAnnotation());
349                     modelicaConnection.getLineConnection().addPoint(
                            new Point2D(currentLineToConnect.getStartX()
                             - tempDragNode.getLayoutX(),
                            currentLineToConnect.getStartY() -
                            tempDragNode.getLayoutY()));
350                 }
351                 if (countClicks == 1) {
352                     modelicaConnection.setFirstConnector(
                            selectedConnector);
353                 } else if (countClicks == 2) {
354                     modelicaConnection.setSecondConnector(
                            selectedConnector);
355                     modelicaConnection.getLineConnection().addPoint(
                            new Point2D(currentLineToConnect.getStartX()
                            , currentLineToConnect.getStartY()));
356                     rootModel.addConnection(modelicaConnection);
357                     modelicaConnection = null;
358                     countClicks = 0;
359                 }
360             }
361         }
```

```
362             event.consume();
363         });
364 
365         designPane.requestFocus();// Asignar el foco al Pane para que
                pueda recibir eventos de teclado
366 
367         designScrollPane.setOnKeyPressed(event -> {//Eventos de teclado
368             if (DEBUG) {
369                 System.out.println("key: " + event.getCode() + "-> " +
                        event.getCharacter());
370             }
371             if (event.getCode() == KeyCode.DELETE) {
372                 removeComponentSelected();
373             }
374             if (event.getCode() == KeyCode.ESCAPE) { //Cancelar la
                    conexión
375                 if (currentLineToConnect != null) {
376                     designPane.getChildren().remove(currentLineToConnect
                            );
377                     currentLineToConnect = null;
378                     currentConnectIconAnnotation = null;
379                     isDrawingLine = false;
380                 }
381             }
382             event.consume();
383         });
384 
385         designPane.setOnMouseMoved(event -> {//dibujar la linea de
                conexión al mover el ratón
386             if (currentLineToConnect != null && isDrawingLine == true) {
387                 currentLineToConnect.setEndX(event.getX());
388                 currentLineToConnect.setEndY(event.getY());
389             }
390             event.consume();
391         });
392 
393         designPane.setOnMouseReleased(event -> {
394             if (event.getTarget() instanceof Shape &&
                    currentLineToConnect != null && isDrawingLine == true) {
395                 Shape tempShape = (Shape) event.getTarget();
396                 boolean isConnector = isConnector(tempShape);
397                 if (isConnector) {
398                     if (DEBUG) {
399                         System.out.println(">>>> Released");
400                     }
401                     currentConnectIconAnnotation = tempShape;
402                     DraggableNode tempDragNode =
                            getDraggableNodeContainer(tempShape);
403                     bindLineToShape(tempDragNode, "end", event.getX(),
                            event.getY());
404                     currentLineToConnect.setMouseTransparent(false);//
                            Importante vuelve a ser visible
405                     isDrawingLine = false;
406                     currentLineToConnect = null;
407                     currentConnectIconAnnotation = null;
408                 }
409             }
410             event.consume();
411         });
412 
413     }
```

```
414
415     /**
416      * *
417      * Determina si un Shape pertenece a un conector.
418      *
419      * @param shape
420      * @return true si el Shape pertenece a un conector
421      */
422     private boolean isConnector(Shape shape) {
423         Shape tempShape = shape;
424         /* pruebas*/
425         if (tempShape.getParent() instanceof Group){
426             IconAnnotation tempIcoAnnotation = (IconAnnotation)
                    tempShape.getParent().getParent();
427            return tempIcoAnnotation.isConnector(shape);
428         }
429         //end pruebas
430         if (tempShape.getParent() instanceof IconAnnotation) {
431             IconAnnotation tempIcoAnnotation = (IconAnnotation)
                    tempShape.getParent();
432             ShapeAnnotation shapeFinded = tempIcoAnnotation.
                    getShapeAnnotationByShape(tempShape);
433             if (shapeFinded.getId().contains("connector")) {
434                 return true;
435             }
436         }
437         return false;
438     }
439
440     /**
441      * *
442      * Obtiene la ruta donde se encuentra el código del Shape.
443      *
444      * @param shape
445      * @return ruta del código del Shape, null si no encuentra
446      */
447     private String getRouteConnectorByShape(Shape shape) {
448         Shape tempShape = shape;
449         IconAnnotation tempIcoAnnotation = getIconAnnotationByShape(
                tempShape);
450         //if (tempShape.getParent() instanceof IconAnnotation) {
451         if(tempIcoAnnotation != null)
452         {
453             //IconAnnotation tempIcoAnnotation = (IconAnnotation)
                    tempShape.getParent();
454             ShapeAnnotation shapeFinded = tempIcoAnnotation.
                    getShapeAnnotationByShape(tempShape);
455             if (tempIcoAnnotation.getShapeAnnotationByShape(tempShape).
                    getId().contains("connector")) {
456                 String completedID = tempIcoAnnotation.
                        getShapeAnnotationByShape(tempShape).getId();
457                 String components[] = completedID.split("::");
458                 if (components.length > 3) {
459                     return components[3];
460                 }
461             }
462         }
463         return null;
464     }
465
466     private String getNameConnectorByShape(Shape shape) {
```

```
467         Shape tempShape = shape;
468         IconAnnotation tempIcoAnnotation = getIconAnnotationByShape(
                tempShape);
469         //if (tempShape.getParent() instanceof IconAnnotation) {
470         if(tempIcoAnnotation != null)
471         {
472             //IconAnnotation tempIcoAnnotation = (IconAnnotation)
                    tempShape.getParent();
473             ShapeAnnotation shapeFinded = tempIcoAnnotation.
                    getShapeAnnotationByShape(tempShape);
474             if (tempIcoAnnotation.getShapeAnnotationByShape(tempShape).
                    getId().contains("connector")) {
475                 String completedID = tempIcoAnnotation.
                        getShapeAnnotationByShape(tempShape).getId();
476                 String components[] = completedID.split("::");
477                 if (components.length > 3) {
478                     return components[2];
479                 }
480             }
481         }
482         return null;
483     }
484 
485     private IconAnnotation getIconAnnotationByShape(Shape shape){
486         Shape tempShape = shape;
487         if (tempShape.getParent() instanceof Group){
488             IconAnnotation tempIcoAnnotation = (IconAnnotation)
                    tempShape.getParent().getParent();
489            return tempIcoAnnotation;
490         }
491 
492         if (tempShape.getParent() instanceof IconAnnotation) {
493             IconAnnotation tempIcoAnnotation = (IconAnnotation)
                    tempShape.getParent();
494             return tempIcoAnnotation;
495         }
496         return null;
497     }
498 
499     /**
500      * Permite que las conexiones se muevan junto con el DraggableNode
                al que
501      * estan conectadas.
502      *
503      * @param selectedShape Elemento que se esta moviendo
504      * @param type para especificar si se trata del inicio de la conexió
                n (init)
505      * o el final(end)
506      * @param x posición horizontal del inicio o fin de la linea
507      * @param y posición vertical del inicio o fin de la linea
508      */
509     private void bindLineToShape(DraggableNode selectedShape, String
            type, double x, double y) {
510         if (type.contains("init")) {
511             currentLineToConnect.startXProperty().bind(selectedShape.
                    layoutXProperty().add(x - selectedShape.getLayoutX()));
512             currentLineToConnect.startYProperty().bind(selectedShape.
                    layoutYProperty().add(y - selectedShape.getLayoutY()));
513 
514         } else {
```

```
515             currentLineToConnect.endXProperty().bind(selectedShape.
                    layoutXProperty().add(x - selectedShape.getLayoutX()));
516             currentLineToConnect.endYProperty().bind(selectedShape.
                    layoutYProperty().add(y - selectedShape.getLayoutY()));
517         }
518     }
519
520     /**
521      * *
522      * Devuelve el DraggableNode que contiene al node.
523      *
524      * @param node que se quiere buscar su contenedor DraggableNode
525      * @return contenedor del node de tipo DraggableNode
526      */
527     private DraggableNode getDraggableNodeContainer(Node node) {
528         if (node == null) {
529             return null;
530         }
531         if (node instanceof DraggableNode) {
532             return (DraggableNode) node;
533         } else if (node.getParent() instanceof DraggableNode) {
534             return (DraggableNode) node.getParent();
535         } else if (node.getParent().getParent() instanceof DraggableNode
                ) {
536             return (DraggableNode) node.getParent().getParent();
537         }else if (node.getParent().getParent().getParent() instanceof
                DraggableNode) {
538             return (DraggableNode) node.getParent().getParent().
                    getParent();
539         }
540         return null;
541     }
542
543     private String setComponentNameByDialog(String initialName) {
544         TextInputDialog dialog = new TextInputDialog(initialName);
545         dialog.setTitle("Nuevo componente");
546         dialog.setHeaderText("Ingresa el nombre del componente");
547         dialog.setContentText("Nombre:");
548         dialog.showAndWait();
549         return dialog.getResult();
550     }
551
552     private Point2D transformFromSystemToModelicaCoordinate(double x,
            double y) {
553         double targetWidth = 200;
554         double targetHeight = 200;
555         double localX = (x - (designPane.getWidth() / 2)) * targetWidth
                / designPane.getWidth();
556         double localY = -(y - (designPane.getHeight() / 2)) *
                targetHeight / designPane.getHeight();
557         return new Point2D(localX, localY);
558     }
559
560     private Point2D transformFromModelicaToSystemCoordinate(double
            localX, double LocalY) {
561         double targetWidth = 200;
562         double targetHeight = 200;
563         double x = (localX * designPane.getWidth() / targetWidth) + (
                designPane.getWidth() / 2);
564         double y = (-LocalY * designPane.getWidth() / targetHeight) + (
                designPane.getWidth() / 2);
```

```
565         return new Point2D(x, y);
566     }
567 
568     private void setSelectedLineConnection(Line line) {
569         if (selectedLine != null) {
570             selectedLine.setStyle(""); // Deshacer el resaltado de la
                    linea previamente seleccionada
571         }
572         selectedLine = line;
573         if (selectedLine != null) {
574             selectedLine.setStyle(
575                     "-fx-stroke: red; "
576                     + "-fx-stroke-width: 4;"
577             );
578         }
579     }
580 
581     private void setSelectedComponent(Pane pane) {
582         if (selectedPane != null) {
583             selectedPane.setStyle(""); // Deshacer el resaltado del Pane
                     previamente seleccionado
584         }
585         selectedPane = pane;
586         if (selectedPane != null) {
587             selectedPane.setStyle(
588                     "-fx-border-color: red; "
589                     + "-fx-border-width: 2;"
590                     + "-fx-effect: dropshadow(gaussian, red, 10, 0, 0,
                         0);" /* Sombras (tipo, color, radio, offsetX,
                         offsetY, spread) */
591             ); // Resaltar el Pane seleccionado
592         }
593     }
594 
595     private void removeComponentSelected() {
596         if (selectedLine != null) {
597             rootModel.removeModelicaConnectionByLineObject(selectedLine)
                    ;
598             designPane.getChildren().remove(selectedLine);
599         } else if (selectedPane != null) {
600             String id = selectedPane.getId();
601             designPane.getChildren().remove(selectedPane);
602             List<Line> linesToRemove = rootModel.
                    removeModelicaConnectionById(id);
603             for (Line line : linesToRemove) {
604                 designPane.getChildren().remove(line);
605             }
606             rootModel.removeElementOfModelCompositionById(id);
607         }
608 
609     }
610 
611     private void configureTreeViewEvents() {
612         // Configurar celdas personalizadas para mostrar los iconos
613         fluidTreeView.setCellFactory((TreeView<NodeItemCode> treeView)
                -> {
614             TreeCell<NodeItemCode> cell = new TreeCell<>() {
615                 @Override // rellenar los item del treeView
616                 protected void updateItem(NodeItemCode item, boolean
                        empty) {
617                     super.updateItem(item, empty);
```

```
618                     if (empty || item == null) {
619                         setText(null);
620                         setGraphic(null);
621                     } else {
622                         setText(item.getName());
623                         setGraphic(null);
624                         if (item.getIconGraphic() != null) {
625                             Pane iconPane = (Pane) item.getIconGraphic()
                                    ;
626                             //Escalar el pane del icono
627                             double targetWidth = 30;
628                             double targetHeight = 30;
629                             double widthOriginal = iconPane.getPrefWidth
                                    ();
630                             double heightOriginal = iconPane.
                                    getPrefHeight();
631                             double scaleX = targetWidth / widthOriginal;
632                             double scaleY = targetHeight /
                                    heightOriginal;
633                             iconPane.setScaleX(scaleX);
634                             iconPane.setScaleY(scaleY);
635                             iconPane.setTranslateX(-widthOriginal / 2 +
                                    scaleX * widthOriginal / 2);
636                             iconPane.setTranslateY(-heightOriginal / 2 +
                                     scaleY * heightOriginal / 2);
637                             Pane paneWrapp = new Pane(); // Wrapp para
                                    conseguir alinearlo correctamente
638                             paneWrapp.getChildren().add(iconPane);
639                             paneWrapp.setStyle("-fx-background-color:
                                    transparent;" + "-fx-border-color:
                                    transparent;");
640                             paneWrapp.setPrefSize(targetWidth,
                                    targetHeight);
641                             setGraphic(paneWrapp);
642                         }
643                         // Personaliza la apariencia del TreeItem
644                         //setFont(Font.font("Arial", FontWeight.BOLD,
                                14));
645                         //setBackground(new Background(new
                                BackgroundFill(Color.CORAL, CornerRadii.
                                EMPTY, Insets.EMPTY)));
646                     }
647                 }
648             };
649             cell.setOnMouseClicked(event -> {// mouse click en la celda
650                 if (event != null) {
651                     if (DEBUG) {
652                         System.out.println("click: " + event.getSource()
                                );
653                     }
654                 }
655             });
656
657             cell.setOnDragDetected((MouseEvent event) -> {// deteccción
                    de Drag en los items
658                 if (!cell.isEmpty()) {
659                     Dragboard dragboard = cell.startDragAndDrop(
                            TransferMode.ANY);
660                     ClipboardContent content = new ClipboardContent();
661                     content.putString(cell.getItem().getRoute()); //Paso
                             la ruta en el clipboard
```

```
662                     dragboard.setContent(content);
663                     event.consume();
664                 }
665             });
666             return cell;
667         });
668     }
669 
670     /**
671      * *
672      * Construye el árbol de los componentes de la libreria.
673      *
674      * @param modelicaLibPath Ruta de la libreria.
675      */
676     private void makeTreeView(String modelicaLibPath) {
677         File rootDirectory = new File(modelicaLibPath);
678         NodeItemCode rootNode = new NodeItemCode("Modelica");
679         rootNode.setRoute("Modelica");
680         TreeItem<NodeItemCode> rootItem = this.makeTreeFromDirectory(
                rootNode, rootDirectory);
681         rootItem.setExpanded(true);
682         rootItem = makeAllRouteToTreeViewComponent(rootItem);
683         fluidTreeView.setRoot(rootItem);
684         makeIcon(rootItem);//generar los iconos
685         TreeItem<NodeItemCode> fluidItem = getTreeItemByRute(rootItem, "
                Modelica.Fluid");
686         if (fluidItem != null) {
687             fluidItem.setExpanded(true);// expandir rama fluid
688         }
689 
690     }
691 
692     /**
693      * *
694      * Crea ramas del árbol a partir de carpetas.
695      *
696      * @param rootNodeItem La rama principal del árbol.
697      * @param file contiene la ruta de la carpeta.
698      * @return Rama a partir de la carpeta.
699      */
700     private TreeItem<NodeItemCode> makeTreeFromDirectory(NodeItemCode
            rootNodeItem, File file) {
701         TreeItem<NodeItemCode> item = new TreeItem<>(rootNodeItem);
702         File[] children = file.listFiles();
703         if (children != null) {
704             for (File child : children) {
705                 if (child.isDirectory()) {
706                     if (child.getName().equals("Resources")) { //omitir
                            la carpeta de recursos
707                         continue;
708                     }
709                     NodeItemCode directoryNode = new NodeItemCode(child.
                            getName());
710                     item.getChildren().add(makeTreeFromDirectory(
                            directoryNode, child));
711 
712                 } else {
713                     if (!this.hasExtension(child, ".mo")) {
714                         continue; // Omitir ficheros que no sean .mo
715                     }
```

```
716                     TreeItem<NodeItemCode> rootItemFile = this.
                            makeTreeFromModelicaFile(child);
717                     item.getChildren().add(rootItemFile);
718                 }
719             }
720         }
721         return item;
722     }
723 
724     /**
725      * *
726      * Crea árbol de componentes a partir de un fichero modelica.
727      *
728      * @param file Ruta del fichero
729      * @return Rama generada por los componentes del fichero.
730      */
731     private TreeItem<NodeItemCode> makeTreeFromModelicaFile(File file) {
732         modelicaAnalizer = new ModelicaAnalizer(file.getAbsolutePath());
733         if (modelicaAnalizer.isIsModelicaFile()) {
734             modelicaAnalizer.analize();
735             String rootName = modelicaAnalizer.getNameOfRootNode();
736             String startMark = "::";
737             String typeComponent = modelicaAnalizer.getTypeComponent();
738             typeComponent = startMark + typeComponent;
739             TreeItem<NodeItemCode> rootItemFile = modelicaAnalizer.
                    createTreeItem(typeComponent, rootName);
740             return rootItemFile;
741         } else {
742             return null;
743         }
744 
745     }
746 
747     /**
748      * *
749      * Permite obtener todos los iconos de forma recursiva para todo el
            árbol de
750      * componentes.
751      *
752      * @param rootTreeItem Rama principal
753      * @return Rama con su iconos en cada subrama
754      */
755     private TreeItem<NodeItemCode> makeIcon(TreeItem<NodeItemCode>
            rootTreeItem) {
756         //List<String> codeList = rootTreeItem.getValue().getCode();
757         String nameComponent = rootTreeItem.getValue().getName();
758         String routeStr = rootTreeItem.getValue().getRoute();
759         try {
760             IconManager iconManager = new IconManager(fluidTreeView.
                    getRoot(), rootTreeItem.getValue(), rootPath);
761             IconAnnotation completeIcon = iconManager.
                    getCompleteIconAnnotation();
762             if (completeIcon != null) {
763                 rootTreeItem.getValue().setIconGraphic(completeIcon.
                        getIcon());
764             } else {
765                 if (DEBUG) {
766                     System.out.println("NO HAY ICONO --->: " + routeStr)
                            ;
767                 }
768             }
```

```
769         } catch (Exception e) {
770             if (DEBUG) {
771                 System.out.println("ERROR --->: " + nameComponent + "\
                        tRoute: " + routeStr);
772             }
773         }
774         for (TreeItem item : rootTreeItem.getChildren()) {
775             makeIcon(item);
776         }
777         return rootTreeItem;
778     }
779 
780     /**
781      * *
782      * Busca el icono de una herencia.
783      *
784      * @param rootTreeItem Rama raiz.
785      * @param icon El icono al que se va combinar el icono extraido de
               la
786      * herencia.
787      * @param extendLine Declacación de la extensión que se va analizar.
788      * @return El icono con los elementos combinados de la herencia.
789      */
790     public IconAnnotation generateExtendIcon(TreeItem<NodeItemCode>
            rootTreeItem, IconAnnotation icon, String extendLine) {
791         String routeToFind = extendLine.replace("extends", "").replace("
                ;", "").strip();
792         NodeItemCode nodeExtend = getNodeByRute(rootTreeItem,
                routeToFind);
793         if (nodeExtend != null) {
794             List<String> codeList = nodeExtend.getCode();
795             String nameComponent = nodeExtend.getName();
796             String typeComponent = nodeExtend.getType();
797             CodeAnalizer codeAnalizer = new CodeAnalizer(codeList,
                    rootPath);
798             codeAnalizer.setComponentName(nameComponent);
799             codeAnalizer.setComponentType(typeComponent);
800             codeAnalizer.analize();
801             IconAnnotation iconExtended = codeAnalizer.getIconPane();
802             if (iconExtended != null) {
803                 icon.getChildren().add(iconExtended.getIcon());
804             }
805             Map<String, List<String>> declatationMap = codeAnalizer.
                    getDeclarations();
806             if (!declatationMap.get("extends").isEmpty()) {
807                 for (String line : declatationMap.get("extends")) {
808                     if (DEBUG) {
809                         System.out.println("\tExtends : " + line);
810                     }
811                     icon.getChildren().add(generateExtendIcon(
                            rootTreeItem, icon, line));
812                 }
813             }
814         }
815         return icon;
816     }
817 
818     /**
819      * *
820      * Crea de manera recursiva la ruta completa para cada elemento del
               árbol de
```

```
821         * componentes.
822         *
823         * @param rootTreeItem Raiz del árbol.
824         * @return Arbol completo con sus rutas correspondientes.
825         */
826        private TreeItem<NodeItemCode> makeAllRouteToTreeViewComponent(
               TreeItem<NodeItemCode> rootTreeItem) {
827            String name = rootTreeItem.getValue().getName();
828            if (rootTreeItem.getParent() != null) {
829                String parentRoute = rootTreeItem.getParent().getValue().
                       getRoute();
830                rootTreeItem.getValue().setRoute(parentRoute + "." + name);
831            }
832            for (TreeItem item : rootTreeItem.getChildren()) {
833                makeAllRouteToTreeViewComponent(item);
834            }
835            return rootTreeItem;
836        }
837
838        /**
839         * *
840         * Obtiene el nodo(componente) correspondiente a una ruta.
841         *
842         * @param root Arbol raiz en el que se va buscar.
843         * @param route Ruta del elemento.
844         * @return Nodo(componente) buscado o null si no se encuentra.
845         */
846        private NodeItemCode getNodeByRute(TreeItem<NodeItemCode> root,
               String route) {
847            if (root.getValue().getRoute().equals(route)) {
848                return root.getValue();
849            }
850            for (TreeItem<NodeItemCode> child : root.getChildren()) {
851                NodeItemCode foundNode = getNodeByRute(child, route);
852                if (foundNode != null) {
853                    return foundNode; // Se ha encontrado el nodo en un hijo
854                }
855            }
856            return null;
857        }
858
859        /**
860         * Obtiene el subarbol a partir de una ruta.
861         *
862         * @param root Arbol raiz en el que se va buscar.
863         * @param route Ruta del elemento.
864         * @return Subarbol a partir de la ruta.
865         */
866        private TreeItem<NodeItemCode> getTreeItemByRute(TreeItem<
               NodeItemCode> root, String route) {
867            if (root.getValue().getRoute().equals(route)) {
868                return root;
869            }
870            for (TreeItem<NodeItemCode> child : root.getChildren()) {
871                TreeItem<NodeItemCode> foundNode = getTreeItemByRute(child,
                       route);
872                if (foundNode != null) {
873                    return foundNode; // Se ha encontrado el nodo en un hijo
874                }
875            }
876            return null;
```

```
877     }
878
879     private void selectInitialTabToShow(Tab tab) {
880         if (tab != null) {
881             SingleSelectionModel<Tab> selectionModel = internalTabPane.
                    getSelectionModel();
882             selectionModel.select(tab);
883         }
884     }
885
886     //Permite comprobar si el fichero tiene una determinada extensión
887     private boolean hasExtension(File file, String extension) {
888         String fileName = file.getName();
889         return fileName.endsWith(extension);
890     }
891
892     /**
893      * Configuración inicial de la interfaz gráfica.
894      */
895     private void initialGUIConfigure() {
896         Image openImage = readImageFromName("open.png");
897         ImageView iconImageView = new ImageView(openImage);
898         openButton.setGraphic(iconImageView);
899         newButton.setGraphic(new ImageView(readImageFromName("new.png"))
                );
900         saveButton.setGraphic(new ImageView(readImageFromName("save.png"
                )));
901     }
902
903     private Image readImageFromName(String name) {
904         Image iconImage = new Image(getClass().getResourceAsStream(name)
                );
905         return iconImage;
906     }
907
908     @FXML
909     private void newButtonAction(ActionEvent event) {
910         if (!designPane.getChildren().isEmpty()) {
911             createDeleteConfirmationDialog();
912         } else {
913             generateNewRootModel();
914         }
915
916     }
917
918     private void createDeleteConfirmationDialog() {
919         Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
920         alert.setTitle("Nuevo modelo");
921         alert.setHeaderText(" Est ás seguro de crear un nuevo Model?");
922         alert.setContentText("Cualquier cambio no guardado se perderá.")
                ;
923         ButtonType buttonTypeYes = new ButtonType("Aceptar");
924         ButtonType buttonTypeNo = new ButtonType("Cancelar");
925         alert.getButtonTypes().setAll(buttonTypeYes, buttonTypeNo);
926         alert.showAndWait().ifPresent(buttonType -> {
927             if (buttonType == buttonTypeYes) {
928                 clearCurrentModel();
929                 generateNewRootModel();
930             }
931         });
932     }
```

```
933
934     private void clearCurrentModel(){
935         this.rootModel.clear();
936         this.designPane.getChildren().clear();
937         selectInitialTabToShow(tabDesign);
938         selectedPane = null;
939         currentLineToConnect = null;
940         currentConnectIconAnnotation = null;
941         modelicaConnection = null;
942         isDrawingLine = false;
943         countClicks = 0;
944     }
945
946     private void generateNewRootModel() {
947         TextInputDialog dialog = new TextInputDialog("Mymodel");
948         dialog.setTitle("Crear nuevo modelo");
949         dialog.setHeaderText("Ingrese el nombre de su modelo");
950         dialog.setContentText("Nombre:");
951         dialog.showAndWait();
952         String nameModel = dialog.getResult();
953         rootModel = new Model(nameModel);
954         updateStatusMessageApp();
955     }
956
957     private void updateStatusMessageApp() {
958         if (rootModel.getName() == null) {
959             tabDiagram.setText("Diagram default");
960         } else {
961             tabDiagram.setText("Diagram " + rootModel.getName());
962         }
963         if (rootModel.getAbsolutePath() == null) {
964             textModelSavedPath.setText("File: (not saved)");
965         } else {
966             textModelSavedPath.setText("File saved at: " + rootModel.
                    getAbsolutePath());
967         }
968         selectInitialTabToShow(tabDesign);
969     }
970
971     @FXML
972     private void saveAsButtonAction(ActionEvent event) {
973         onCodeViewAction(event); //update textArea
974         File modelicaFile = showAndGetFileToSaveDialog();
975         if (modelicaFile != null) {
976             rootModel.setAbsolutePath(modelicaFile.getAbsolutePath());
977         } else {
978             return;
979         }
980         String content = rootCodeArea.getText();
981         File fileToSave = new File(rootModel.getAbsolutePath());
982         boolean status = saveModelicaFile(fileToSave, content);
983         if (status == false) {
984             showAlertErrorToSaveDialog();
985         }
986         updateStatusMessageApp();
987     }
988
989     @FXML
990     private void saveButtonAction(ActionEvent event) {
991         if (rootModel == null || rootCodeArea == null) {
992             return;
```

```
993         }
994         String absolutePath = rootModel.getAbsolutePath();
995         if (absolutePath == null) {
996             File modelicaFile = showAndGetFileToSaveDialog();
997             if (modelicaFile != null) {
998                 rootModel.setAbsolutePath(modelicaFile.getAbsolutePath()
                        );
999             } else {
1000                return;
1001            }
1002        }
1003        onCodeViewAction(event);
1004        String content = rootCodeArea.getText();
1005        File fileToSave = new File(rootModel.getAbsolutePath());
1006        boolean status = saveModelicaFile(fileToSave, content);
1007        if (status == false) {
1008            showAlertErrorToSaveDialog();
1009        }
1010        updateStatusMessageApp();
1011    }
1012
1013    private File showAndGetFileToSaveDialog() {
1014        FileChooser fileChooser = new FileChooser();
1015        fileChooser.setTitle("Guardar " + rootModel.getName());
1016        fileChooser.getExtensionFilters().add(new FileChooser.
                ExtensionFilter("Modelica File", "*.mo"));
1017        fileChooser.setInitialFileName(rootModel.getName());
1018        File modelicaFile = fileChooser.showSaveDialog(null);
1019        return modelicaFile;
1020    }
1021
1022    private void showAlertErrorToSaveDialog() {
1023        Alert alert = new Alert(Alert.AlertType.WARNING);
1024        alert.setTitle("Error al guardar el fichero");
1025        alert.setHeaderText("No ha sido posible guardar el fichero " +
                rootModel.getName() + "\nRuta: " + rootModel.getAbsolutePath
                ());
1026        alert.setContentText("Intentelo guardar en otra ubicación.");
1027        ButtonType buttonTypeYes = new ButtonType("Aceptar");
1028        alert.getButtonTypes().setAll(buttonTypeYes);
1029        alert.showAndWait();
1030        rootModel.setAbsolutePath(null);
1031    }
1032
1033    private boolean saveModelicaFile(File modelicaFile, String content)
            {
1034        try (FileWriter fileWriter = new FileWriter(modelicaFile)) {
1035            fileWriter.write(content);
1036        } catch (IOException e) {
1037            return false;
1038        }
1039        return true;
1040    }
1041
1042    @FXML
1043    /**
1044     * *
1045     * Generación del code Modelica en el textArea
1046     */
1047    private void onCodeViewAction(Event event) {
1048        if (designPane == null || rootModel == null) {
```

```
1049            return;
1050        }
1051        for (Node node : this.designPane.getChildren()) {
1052            if (node instanceof Line) {
1053                Line tempLine = (Line) node;
1054                String lineId = tempLine.getId();
1055                double startLineX = tempLine.getStartX();
1056                double startLineY = tempLine.getStartY();
1057                double endLineX = tempLine.getEndX();
1058                double endLineY = tempLine.getEndY();
1059                Point2D startPoint =
                        transformFromSystemToModelicaCoordinate(startLineX,
                        startLineY);
1060                Point2D endPoint =
                        transformFromSystemToModelicaCoordinate(endLineX,
                        endLineY);
1061                List<Point2D> points = new ArrayList<>();
1062                points.add(startPoint);
1063                points.add(endPoint);
1064                if (rootModel.getModelicaConnectionById(lineId) != null)
                        {
1065                    rootModel.getModelicaConnectionById(lineId).
                            getLineConnection().setPoints(points);
1066                    if (DEBUG) {
1067                        System.out.println("line: " + tempLine.toString
                                ());
1068                        System.out.println("annotation: " + rootModel.
                                getModelicaConnectionById(lineId).
                                getCodeString());
1069                    }
1070                }
1071
1072            }
1073
1074        }
1075
1076        String textCode = "model " + this.rootModel.getName() + "\n";
1077        for (ModelicaParameter parameter : rootModel.getAllCompositions
                ()) {
1078            textCode += parameter.getCodeString() + "\n";
1079        }
1080        textCode += "equation\n";
1081        for (ModelicaConnection connection : rootModel.getAllConnections
                ()) {
1082            if (connection.getFirstConnector().isIsArray()) {
1083                connection.getFirstConnector().setIndexArray(connection.
                        getFirstConnector().getIndexArray() + 1);
1084            }
1085            if (connection.getSecondConnector().isIsArray()) {
1086                connection.getSecondConnector().setIndexArray(connection
                        .getSecondConnector().getIndexArray() + 1);
1087            }
1088            textCode += connection.getCodeString() + "\n";
1089        }
1090        for (ModelicaConnection connection : rootModel.getAllConnections
                ()) {
1091            //reiniciar indexArrayConnector
1092            if (connection.getFirstConnector().isIsArray()) {
1093                connection.getFirstConnector().setIndexArray(0);
1094            }
1095            if (connection.getSecondConnector().isIsArray()) {
```

```
1096                connection.getSecondConnector().setIndexArray(0);
1097            }
1098        }
1099        textCode += "\nend " + this.rootModel.getName() + ";";
1100        this.rootCodeArea.setText(textCode);
1101    }
1102
1103    @FXML
1104    private void openButtonAction(ActionEvent event) {
1105        if (!designPane.getChildren().isEmpty()) {
1106            Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
1107            alert.setTitle("Abrir modelo");
1108            alert.setHeaderText(" Est ás seguro que desea abrir un nuevo
                 Model?");
1109            alert.setContentText("Cualquier cambio no guardado se perder
                á.");
1110            ButtonType buttonTypeYes = new ButtonType("Aceptar");
1111            ButtonType buttonTypeNo = new ButtonType("Cancelar");
1112            alert.getButtonTypes().setAll(buttonTypeYes, buttonTypeNo);
1113            alert.showAndWait().ifPresent(buttonType -> {
1114                if (buttonType == buttonTypeYes) {// Si el usuario elige
                     "Aceptar", eliminar el modelo actual
1115                    clearCurrentModel();
1116                    openRootModel();
1117                }
1118            });
1119        } else {
1120            openRootModel();
1121        }
1122    }
1123
1124    private void openRootModel() {
1125        FileChooser fileChooser = new FileChooser();
1126        fileChooser.setTitle("Abrir Modelo");
1127        fileChooser.getExtensionFilters().addAll(
1128                new FileChooser.ExtensionFilter("Archivos Modelica", "*.
                    mo")
1129        //new FileChooser.ExtensionFilter("Todos los archivos", "*.*")
1130        );
1131        // Mostrar el cuadro de diálogo para obtener el archivo que se
            desea abrir
1132        File selectedFile = fileChooser.showOpenDialog(null);
1133
1134        if (selectedFile != null) {
1135            if (DEBUG) {
1136                System.out.println("Archivo seleccionado: " +
                        selectedFile.getAbsolutePath());
1137            }
1138            TreeItem<NodeItemCode> treeFile = makeTreeFromModelicaFile(
                    selectedFile);
1139            // crear la clase parameters
1140            ModelManager basicModelicaManager = new ModelManager(
                    fluidTreeView.getRoot(), treeFile.getValue(), rootPath);
1141            Map<String, List<String>> declarationsModelMap =
                    basicModelicaManager.getAllDeclarationsMap();
1142
1143            // Lectura de todos los componentes definidos en el bloque
                de declaraciones
1144            for (String declarationLine : declarationsModelMap.get("
                    variables")) {
```

```
1145                List<String> typeAndName = basicModelicaManager.
                        getTypeAndNameComponentByLine(declarationLine);
1146                String typeComponet = typeAndName.get(0);
1147                String nameComponet = typeAndName.get(1);
1148                NodeItemCode nodeItem = this.getNodeByRute(fluidTreeView
                        .getRoot(), typeComponet);
1149                if (nodeItem == null) {
1150                    continue; //El componente no tiene un código
                            asociado
1151                }
1152                IconManager iconManager = new IconManager(fluidTreeView.
                        getRoot(), nodeItem, rootPath);
1153                IconAnnotation completeIcon = iconManager.
                        getCompleteIconAnnotation();
1154
1155                //Configurar el nombre del componente (visualización)
1156                for (ShapeAnnotation shape : completeIcon.getShapes()) {
1157                    if (shape instanceof TextAnnotation) {
1158                        String textString = ((TextAnnotation) shape).
                                getTextString();
1159                        if (textString.contains("%name")) {
1160                            ((TextAnnotation) shape).setTextString(
                                    nameComponet);
1161                        }
1162                    }
1163                }
1164                // Crear la clase parameters que representa cada uno de
                        los parametros de cada componente
1165                ModelManager modelicaManager = new ModelManager(
                        fluidTreeView.getRoot(), nodeItem, rootPath);
1166                ModelicaParameter parameter = modelicaManager.
                        getAllParameterModel();
1167                Map<String, String> redefinitionsMap =
                        basicModelicaManager.getRedefinitionsByLineString(
                        declarationLine);
1168                parameter.setRedefinicions(redefinitionsMap);
1169                parameter.setNameComponent(nameComponet);
1170                // actualizar los valores de las redefiniciones de cada
                        componentModel
1171                for (Map.Entry<String, String> entry : redefinitionsMap.
                        entrySet()) {
1172                    String key = entry.getKey();
1173                    if (key.contains(" ")) {// adaptar para:
                            remplaceable package medium
1174                        String[] tempKeys = key.split("\\s");
1175                        if (tempKeys.length > 0) {
1176                            key = tempKeys[tempKeys.length - 1];
1177                        }
1178                    }
1179                    String value = entry.getValue();
1180                    ComponentModel component = parameter.
                            getComponentByName(key);
1181                    if (component != null) {
1182                        component.setValue(value);
1183                    }
1184                }
1185                //generate ID
1186                String id = "" + parameter.getPath() + parameter.
                        getNameComponent();
1187                parameter.setId(id);
1188                //Configuración del icono
```

```
1189                CoordinateSystem iconCoordinate = completeIcon.
                        getCoordinateSystem();
1190                double scale = SCALE_VIEW_ICON * iconCoordinate.
                        getInitialScale();
1191                double widthIcon = scale * iconCoordinate.getExtent().
                        getWidth();
1192                double heightIcon = scale * iconCoordinate.getExtent().
                        getHeight();
1193                Node completIcon = completeIcon.getIcon();
1194                completIcon.getTransforms().add(new Scale(scale, scale))
                        ;
1195                DraggableNode dragNode = new DraggableNode(completIcon);
1196                dragNode.setId(id);
1197                dragNode.setName(parameter.getNameComponent());
1198                dragNode.setPrefSize(widthIcon + 4, heightIcon + 4);
1199                dragNode.setStyle("-fx-background-color: transparent; ")
                        ;
1200                designPane.getChildren().add(dragNode);
1201
1202                //Crear las trasformaciones (placement)
1203                Placement placement = modelicaManager.
                        getPlacementByLineString(declarationLine);
1204                Transformation transformation = placement.
                        getTransformation();
1205                Point2D origin = placement.getTransformation().getOrigin
                        ();
1206                Point2D originSys =
                        transformFromModelicaToSystemCoordinate(origin.getX
                        (), origin.getY());
1207                dragNode.setLayoutX(originSys.getX() - (dragNode.
                        getPrefWidth() / 2));
1208                dragNode.setLayoutY(originSys.getY() - (dragNode.
                        getPrefHeight() / 2));
1209                if (DEBUG) {
1210                    System.out.println(parameter.getNameComponent() + "
                            -> local:" + origin + "\nSys:" + originSys);
1211                }
1212                double leftWidth = -widthIcon / (2 * SCALE_VIEW_ICON);
1213                double topHeight = -heightIcon / (2 * SCALE_VIEW_ICON);
1214                transformation.setExtent(new Extent(leftWidth, topHeight
                        , -leftWidth, -topHeight));
1215                parameter.setPlacement(placement);
1216                dragNode.setPlacement(placement);
1217                List<ModelicaConnector> connectors = iconManager.
                        getConnectors();
1218                for (ModelicaConnector connector : connectors) {
1219                    connector.setId(id + connector.getType());//para
                            vincular el conector con su componente padre
1220                    connector.setParent(parameter.getNameComponent());//
                             estableciendo los padres
1221                }
1222                dragNode.setConnectors(connectors);
1223                this.rootModel.addElementOfModelComposition(parameter);
1224            }
1225            //Lectura de las conecciones
1226            for (String connectionStr : declarationsModelMap.get("
                    equations")) {
1227                if (DEBUG) {
1228                    System.out.println("Connection: " + connectionStr);
1229                }
```

```
1230                LineAnnotation lineAnnotation = basicModelicaManager.
                        getLineAnnotationOfConnectionByLineString(
                        connectionStr);
1231                if (lineAnnotation != null) {
1232                    Point2D initLine = lineAnnotation.getPoints().get(0)
                            ;
1233                    Point2D endLine = lineAnnotation.getPoints().get(
                            lineAnnotation.getPoints().size() - 1);
1234                    Point2D initLineSys =
                            transformFromModelicaToSystemCoordinate(initLine
                            .getX(), initLine.getY());
1235                    Point2D endLineSys =
                            transformFromModelicaToSystemCoordinate(endLine.
                            getX(), endLine.getY());
1236                    Line lineConnect = new Line(initLineSys.getX(),
                            initLineSys.getY(), endLineSys.getX(),
                            endLineSys.getY());
1237                    designPane.getChildren().add(lineConnect);
1238
1239                    Map<String, String> connectionsMap =
                            basicModelicaManager.
                            getConnectionMapByLineString(connectionStr);
1240                    String firstcomponentName = connectionsMap.get("
                            firstComponentName");
1241                    String firstConnectorName = connectionsMap.get("
                            firstConnectorName");
1242                    String secondcomponentName = connectionsMap.get("
                            secondComponentName");
1243                    String secondConnectorName = connectionsMap.get("
                            secondConnectorName");
1244
1245                    // Buscar el draggableNode de las conexiones
1246                    currentLineToConnect = lineConnect;
1247                    if (modelicaConnection == null) {
1248                        modelicaConnection = new ModelicaConnection(
                                currentLineToConnect);
1249                        modelicaConnection.setLineConnection(new
                                LineAnnotation());
1250                        DraggableNode firstDN = findDraggableNodeByName(
                                firstcomponentName);
1251                        ModelicaConnector fistConnector = firstDN.
                                getConnectorByName(firstConnectorName);
1252
1253                        Random random = new Random();
1254                        int randomNumber = random.nextInt(1000); // Nú
                                mero aleatorio entre 0 startYLine 999
1255                        currentLineToConnect.setId("lineConnection::" +
                                fistConnector.getType() + "::" + firstDN.
                                getName() + "." + firstConnectorName + "::"
                                + randomNumber);
1256                        modelicaConnection.setId(currentLineToConnect.
                                getId()); // para vincular la linea con la
                                lineaAnnotation
1257                        modelicaConnection.getLineConnection().addPoint(
                                new Point2D(currentLineToConnect.getStartX()
                                , currentLineToConnect.getStartY()));
1258                        modelicaConnection.setFirstConnector(
                                fistConnector);
1259                        double startXLine = lineConnect.getStartX();
1260                        double startYLine = lineConnect.getStartY();
```

```
1261                        bindLineToShape(firstDN, "init", startXLine,
                                startYLine);
1262                        if (DEBUG) {
1263                            System.out.println("Position: " +
                                    lineConnect.getStartX() + "," +
                                    lineConnect.getStartY() + " ->
                                    layoutDragNode: " + firstDN.getLayoutX()
                                     + "," + firstDN.getLayoutY());
1264                        }
1265                        // Final de la conexión
1266                        DraggableNode endDN = findDraggableNodeByName(
                                secondcomponentName);
1267                        ModelicaConnector secondConnector = endDN.
                                getConnectorByName(secondConnectorName);
1268                        if (DEBUG) {
1269                            System.out.println("\tend connect at: " +
                                    secondConnectorName);
1270                        }
1271                        modelicaConnection.setSecondConnector(
                                secondConnector);
1272                        modelicaConnection.getLineConnection().addPoint(
                                new Point2D(currentLineToConnect.getEndX(),
                                currentLineToConnect.getEndY()));
1273                        rootModel.addConnection(modelicaConnection);
1274                        double endXLine = lineConnect.getEndX();
1275                        double endYLine = lineConnect.getEndY();
1276                        bindLineToShape(endDN, "end", endXLine, endYLine
                                );
1277                        if (DEBUG) {
1278                            System.out.println("Position: " +
                                    lineConnect.getStartX() + "," +
                                    lineConnect.getStartY() + " ->
                                    layoutDragNode: " + endDN.getLayoutX() +
                                     "," + endDN.getLayoutY());
1279                        }
1280                    }
1281                    modelicaConnection = null; //reinicio la linea de
                            conexión
1282                }
1283            }
1284            currentLineToConnect = null; //reinicio la linea de conexión
1285        } else {
1286            if (DEBUG) {
1287                System.out.println("Ningún archivo seleccionado para
                        abrir.");
1288            }
1289        }
1290    }
1291
1292    private DraggableNode findDraggableNodeByName(String name) {
1293        for (Node node : designPane.getChildren()) {
1294            if (node instanceof DraggableNode) {
1295                DraggableNode dn = (DraggableNode) node;
1296                if (dn.getName().equals(name)) {
1297                    return dn;
1298                }
1299            }
1300        }
1301        return null;
1302    }
1303
```

```
1304        @FXML
1305        private void deleteModelItemAction(ActionEvent event) {
1306            if (!designPane.getChildren().isEmpty()) {
1307                Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
1308                alert.setTitle("Eliminar Modelo");
1309                alert.setHeaderText(" Est ás seguro que desea eliminar el
                        actual Modelo?");
1310                alert.setContentText("Se eleminará de forma permanente
                        cualquier cambio no guardado.");
1311                // Configurar los botones del diálogo (Aceptar startYLine
                        Cancelar)
1312                ButtonType buttonTypeYes = new ButtonType("Aceptar");
1313                ButtonType buttonTypeNo = new ButtonType("Cancelar");
1314                alert.getButtonTypes().setAll(buttonTypeYes, buttonTypeNo);
1315                // Mostrar el diálogo startYLine esperar a que el usuario
                        elija una opción
1316                alert.showAndWait().ifPresent(buttonType -> {
1317                    if (buttonType == buttonTypeYes) {
1318                        // Si el usuario elige "Aceptar", eliminar el modelo
                                actual
1319                        clearCurrentModel();
1320                    }
1321                });
1322            }
1323        }
1324
1325        @FXML
1326        private void onAboutItemAction(ActionEvent event) {
1327            showAboutWindow();
1328        }
1329
1330        private void showAboutWindow() {
1331            Alert alert = new Alert(AlertType.INFORMATION);
1332            alert.setTitle("Acerca de");
1333            alert.setHeaderText("FluidEditor ");
1334            alert.setContentText("Versión 1.0\nAutor: Jackson F. Reyes
                    Bermeo");
1335            alert.showAndWait();
1336        }
1337
1338        public void confirmClose(Event event) {
1339            Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
1340            alert.setTitle("Confirmar salida");
1341            alert.setHeaderText(" Est á seguro de que desea salir?");
1342
1343            ButtonType buttonTypeSi = new ButtonType("Sí");
1344            ButtonType buttonTypeNo = new ButtonType("No");
1345            alert.getButtonTypes().setAll(buttonTypeSi, buttonTypeNo);
1346
1347            alert.showAndWait().ifPresent(buttonType -> {
1348                if (buttonType == buttonTypeSi) {
1349                    System.exit(0);
1350                } else {
1351                    event.consume();
1352                }
1353            });
1354        }
1355    }
```

Código B.4: Implementación del controlador principal.

B-2.2. Código del controlador de visualización de parámetros: PropertiesViewController.java

```
1  package com.fluideditor.controller;
2
3  import com.fluideditor.model.modelica.ComponentModel;
4  import com.fluideditor.model.modelica.Dialog;
5  import com.fluideditor.model.modelica.ModelicaParameter;
6  import java.util.ArrayList;
7  import java.util.List;
8  import javafx.event.ActionEvent;
9  import javafx.fxml.FXML;
10 import javafx.scene.Node;
11 import javafx.scene.control.Button;
12 import javafx.scene.control.ScrollPane;
13 import javafx.scene.control.Tab;
14 import javafx.scene.control.TabPane;
15 import javafx.scene.control.TextField;
16 import javafx.scene.control.TitledPane;
17 import javafx.scene.layout.HBox;
18 import javafx.scene.layout.Pane;
19 import javafx.scene.layout.Priority;
20 import javafx.scene.layout.StackPane;
21 import javafx.scene.layout.VBox;
22 import javafx.scene.text.Font;
23 import javafx.scene.text.Text;
24 import javafx.stage.Stage;
25
26 /**
27  * FXML Controlador de PropertiesView.
28  *
29  * @author Jackson F. Reyes Bermeo
30  */
31 public class PropertiesViewController {
32
33     @FXML
34     private Button saveParametersBtn;
35     @FXML
36     private Button cancelParameters;
37     @FXML
38     private TabPane propertiesTabPane;
39
40     private ModelicaParameter modelicaParameter;
41
42     /**
43      * Initializes the controller class.
44      */
45     public void initialize() {
46         modelicaParameter = new ModelicaParameter();
47     }
48
49     public void setParameters(ModelicaParameter parameters) {
50         this.modelicaParameter = parameters;
51     }
52
53     @FXML
54     private void saveParameters(ActionEvent event) {
55         System.out.println("Guardando parametros ...");
56         updateParameters();
```

```
57          Stage stage = (Stage) ((Button) event.getSource()).getScene().
                getWindow();
58          stage.close();
59      }
60
61      /**
62       * Actualiza los parametros editados en la ventana y marca las
63       * redefiniciones que se mostraran en el código Modelica.
64       */
65      private void updateParameters() {
66          List<TextField> allTextFields = findAllTextFields(
                propertiesTabPane);
67          for (TextField txtField : allTextFields) {
68              ComponentModel component = modelicaParameter.
                    getComponentByName(txtField.getId());
69              if (component.getValue() == null) {
70                  component.setValue("");
71              }
72              if (txtField.getText() == null) {
73                  txtField.setText("");
74              }
75              if (!component.getValue().strip().equals(txtField.getText().
                    strip())) { // si hay cambio actualizo datos en el
                    modelo y genero la redefinicion
76                  System.out.println(txtField.getId() + "-> old:" +
                        component.getValue() + " new:" + txtField.getText())
                        ;
77                  component.setValue(txtField.getText().strip());
78                  if (component.getPrefix().contains("replaceable")) {
79                      String prefix = component.getPrefix().replace("
                            replaceable", "redeclare");
80                      String name = prefix + " " + component.getType() + "
                             " + component.getName();
81                      modelicaParameter.addRedefiniton(name, component.
                            getValue());
82                  } else {
83                      modelicaParameter.addRedefiniton(component.getName()
                            , component.getValue());
84                  }
85              }
86          }
87
88      }
89
90      private List<TextField> findAllTextFields(TabPane tabPane) {
91          List<TextField> allTextFields = new ArrayList<>();
92          for (Tab tab : tabPane.getTabs()) {
93              ScrollPane scroll = (ScrollPane) tab.getContent();
94              VBox tabContent = (VBox) scroll.getContent();
95              List<TextField> textFields = findTextFields(tabContent);
96              allTextFields.addAll(textFields);
97          }
98          return allTextFields;
99      }
100
101     private List<TextField> findTextFields(Pane container) {
102         List<TextField> textFields = new ArrayList<>();
103         for (Node node : container.getChildren()) {
104             if (node instanceof TextField) {
105                 textFields.add((TextField) node);
106             } else if (node instanceof TitledPane) {
```

```
107                 VBox vBox = (VBox) ((TitledPane) node).getContent();
108                 List<TextField> result = findTextFields(vBox);
109                 textFields.addAll(result);
110             } else if (node instanceof HBox) {
111                 List<TextField> result = findTextFields((Pane) node);
112                 textFields.addAll(result);
113             }
114         }
115         return textFields;
116     }
117 
118     @FXML
119     private void cancelParameters(ActionEvent event) {
120         Stage stage = (Stage) ((Button) event.getSource()).getScene().
                getWindow();
121         event.consume();
122         stage.close();
123     }
124 
125     private void clearAndResetView(ModelicaParameter modelicaParameter)
            {
126         propertiesTabPane.getTabs().clear();
127         // Elementos que se muestran por defecto
128         String componentName = modelicaParameter.getNameComponent();
129         String comment = modelicaParameter.getComment();
130         String path = modelicaParameter.getPath();
131         Tab defaultTab = new Tab("General");
132         TitledPane pane1 = new TitledPane("Component", new VBox(new Text
                ("name: " + componentName), new Text("comment: ")));
133         pane1.setPadding(new javafx.geometry.Insets(5, 10, 5, 10));
134         pane1.setMinWidth(600);
135         TitledPane pane2 = new TitledPane("Class", new VBox(new Text("
                path: " + path), new Text("comment: " + comment)));
136         pane2.setPadding(new javafx.geometry.Insets(5, 10, 5, 10));
137         ScrollPane scrollPane = new ScrollPane();
138         scrollPane.setFitToWidth(true);
139         scrollPane.setFitToHeight(true);
140         scrollPane.setStyle("fx-background-color:white;");
141         VBox vbox = new VBox(pane1, pane2);
142         vbox.setStyle("-fx-background-color:white;");
143         scrollPane.setContent(vbox);
144         defaultTab.setContent(scrollPane);
145         propertiesTabPane.getTabs().add(defaultTab);
146     }
147 
148     public void showParamiter(ModelicaParameter modelicaParameter) {
149         clearAndResetView(modelicaParameter);
150         if (modelicaParameter != null) {
151             for (ComponentModel componentModel : modelicaParameter.
                    getComponents()) {
152                 if (componentModel.getPrefix() != null && componentModel
                        .getPrefix().contains("final")) {
153                     continue; // no mostrasmos los parametros que son
                            finales
154                 }
155                 Dialog dialog = componentModel.getDialog();
156                 String tabTitle = dialog.getTab();
157                 String groupName = dialog.getGroup();
158                 Tab newTab = buscarOCrearTab(propertiesTabPane, tabTitle
                        );
```

```
159                 TitledPane newTitlePane = buscarOCrearTitledPane(newTab,
                        groupName);
160                 // Obtener el contenido actual del TitledPane
161                 VBox contenidoActual = (VBox) newTitlePane.getContent();
162                 // Agregar los nuevos elementos al contenido actual
163                 HBox horizontalContent = new HBox();
164                 horizontalContent.setSpacing(5);
165                 Text textName = new Text(componentModel.getName());
166                 StackPane textWidthFixed = new StackPane(textName);
167                 textWidthFixed.setMinWidth(120);
168                 TextField textValue = new TextField(componentModel.
                        getValue());
169                 textValue.setId(componentModel.getName());
170                 textValue.setMinWidth(250);
171                 Text textComment = new Text(componentModel.getComment())
                        ;
172                 textComment.setFont(new Font(10));
173                 StackPane commentWidthFixe = new StackPane(textComment);
174                 commentWidthFixe.setMinWidth(150);
175                 HBox.setHgrow(textValue, Priority.ALWAYS);
176                 horizontalContent.getChildren().addAll(textWidthFixed,
                        textValue, commentWidthFixe);
177                 contenidoActual.getChildren().add(horizontalContent);
178                 // Establecer el contenido actualizado en el TitledPane
179                 newTitlePane.setContent(contenidoActual);
180             }
181         }
182         Stage stage = (Stage) this.propertiesTabPane.getScene().
                getWindow();
183         stage.showAndWait();
184     }
185
186     /**
187      * Busca una pesta a por su título o crear una nueva pesta a si no
               existe
188      **/
189     private Tab buscarOCrearTab(TabPane tabPane, String title) {
190         for (Tab tab : tabPane.getTabs()) {
191             if (tab.getText().equals(title)) {
192                 return tab; // Pesta a encontrada
193             }
194         }
195         // Si la pesta a no existe, la creamos y la agregamos al
                TabPane
196         Tab nuevaTab = new Tab(title);
197         VBox vbox = new VBox();
198         vbox.setFillWidth(true);
199         vbox.setStyle("-fx-background-color:white;");
200         ScrollPane scrollPane = new ScrollPane(vbox);
201         scrollPane.setFitToWidth(true);
202         scrollPane.setFitToHeight(true);
203         scrollPane.setStyle("-fx-background-color:red;");
204         nuevaTab.setContent(scrollPane);
205         tabPane.getTabs().add(nuevaTab);
206         return nuevaTab;
207     }
208
209     // Método para buscar un TitledPane por su título o crear uno nuevo
            si no existe
210     private TitledPane buscarOCrearTitledPane(Tab tab, String
            tituloBuscado) {
```

```
211         VBox vbox = null;
212         if (tab.getContent() instanceof ScrollPane) {
213             ScrollPane scrollPane = (ScrollPane) tab.getContent();
214             if (scrollPane.getContent() instanceof VBox) {
215                 vbox = (VBox) scrollPane.getContent();
216             }
217         }
218         for (Node node : vbox.getChildren()) {
219             if (node instanceof TitledPane) {
220                 TitledPane titledPane = (TitledPane) node;
221                 if (titledPane.getText().equals(tituloBuscado)) {
222                     return titledPane; // TitledPane encontrado
223                 }
224             }
225         }
226         // Si el TitledPane no existe, lo creamos y lo agregamos al VBox
227         TitledPane nuevoTitledPane = new TitledPane(tituloBuscado, new
                VBox());
228         nuevoTitledPane.setPadding(new javafx.geometry.Insets(5, 10, 5,
                10));
229         vbox.getChildren().add(nuevoTitledPane);
230         return nuevoTitledPane;
231     }
232 }
```

Código B.5: Implementación del controlador de visualización de parámetros.

B-3. Implementación de los modelos

B-3.1. Implementación de la librería gráfica

Código de la clase encargada de analizar texto escrito en lenguaje Modelica: CodeAnalizer.java

```
1  package com.fluideditor.model.icon;
2
3  import com.fluideditor.model.modelica.ComponentModel;
4  import com.fluideditor.model.modelica.Dialog;
5  import com.fluideditor.model.modelica.ModelicaConnector;
6  import com.fluideditor.model.modelica.ModelicaParameter;
7  import java.util.ArrayList;
8  import java.util.Arrays;
9  import java.util.HashMap;
10 import java.util.List;
11 import java.util.Map;
12 import java.util.regex.Matcher;
13 import java.util.regex.Pattern;
14 import javafx.geometry.Point2D;
15 import javafx.scene.paint.Color;
16
17 /**
18  * Analizador de código Modelica.
19  *
20  * @author Jackson F. Reyes Bermeo
```

```
21  */
22 public class CodeAnalizer {
23
24     private String within;
25     private final String rootPath;
26     private String componentType;
27     private String componentName;
28     private Placement placement;
29     private final List<String> codeList;
30     private final Map<String, List<String>> declarations;
31     private Map<String, List<String>> completeDeclarations;
32     private String annotation;
33     private final Map<String, List<String>> iconMap;
34     private Map<String, String> propertiesPattern;
35     private final boolean DEBUG = false;
36
37     public CodeAnalizer(List<String> codeList, String rootPath) {
38         this.rootPath = rootPath;
39         this.codeList = codeList;
40         this.declarations = new HashMap<>();
41         this.completeDeclarations = new HashMap<>();
42         this.iconMap = new HashMap<>();
43         createMapPatternToExtractGraphicPrimitives();
44     }
45
46     private void createMapPatternToExtractGraphicPrimitives() {
47         // Patrones para extraer las propiedades que contiene cada
               primitiva
48         propertiesPattern = new HashMap<>();
49         propertiesPattern.put("origin", "origin=(\\{.*?\\})"); //
50         propertiesPattern.put("extent", "extent=(\\{\\{.*?\\}\\})");//
51         propertiesPattern.put("lineColor", "lineColor=(\\{.*?\\})"); //
52         propertiesPattern.put("fillColor", "fillColor=(\\{.*?\\})"); //
53         propertiesPattern.put("fillPattern", "fillPattern=(\\w+.\\w+)");
               //
54         propertiesPattern.put("pattern", "pattern=(\\w+.\\w+)"); //
55         propertiesPattern.put("lineThickness", "\\blineThickness\\s?=\\s
               ?(-?[\\d+\\.]+)"); //
56         propertiesPattern.put("thickness", "\\bthickness\\s?=\\s?(-?[\\d
               +\\.]+)"); //
57         propertiesPattern.put("rotation", "\\brotation=(-?[\\d+\\.]+)");
               //
58         propertiesPattern.put("radius", "\\bradius\\s?=\\s?(-?[\\d
               +\\.]+)"); //
59         propertiesPattern.put("startAngle", "\\bstartAngle\\s?=\\s
               ?(-?[\\d+\\.]+)"); //
60         propertiesPattern.put("endAngle", "\\bendAngle\\s?=\\s?(-?[\\d
               +\\.]+)"); //
61         propertiesPattern.put("color", "color=(\\{.*?\\})"); //
62         propertiesPattern.put("textColor", "textColor=(\\{.*?\\})"); //
63         propertiesPattern.put("textString", "textString\\s*=\\s
               *(\".*?\")"); //
64         propertiesPattern.put("fontName", "fontName\\s*=\\s*\"(.*?)\"");
               //
65         propertiesPattern.put("fontSize", "\\bfontSize\\s?=\\s?(-?[\\d
               +\\.]+)");//points\s*=\s*\{\{.*?\}\}
66         propertiesPattern.put("points", "points\\s*=\\s*\\{\\{.*?\\}\\}"
               );
67         propertiesPattern.put("horizontalAlignment", "
               horizontalAlignment\\s*=\\s*\\w+.(\\w+)");
```

```
68          propertiesPattern.put("textStyle", "(textStyle\\s*=\\s
                *\\{.*?\\})");
69          propertiesPattern.put("fileName", "fileName\\s*=\\s*\\\"(.*)\\\"
                ");
70      }
71
72      /**
73       *
74       * @return un Map que contiene cada linea de instrucciones modelica.
75       */
76      public Map<String, List<String>> getAllDeclarations() {
77          return completeDeclarations;
78      }
79
80      public void setCompleteDeclarations(Map<String, List<String>>
            completedDeclarations) {
81          completeDeclarations = completedDeclarations;
82      }
83
84      public String getComponetType() {
85          return componentType;
86      }
87
88      public void setComponentType(String componentType) {
89          if (!componentType.contains("::")) {//marca de se al
90              this.componentType = "::" + componentType;
91          } else {
92              this.componentType = componentType;
93          }
94      }
95
96      public String getComponentName() {
97          return componentName;
98      }
99
100     public void setComponentName(String componentName) {
101         this.componentName = componentName;
102     }
103
104     /**
105      * Extrae within, annotation, y componentes de las declaraciones (
             incluido
106      * el icono). *
107      */
108     public void analize() {
109         if (codeList != null || !codeList.isEmpty()) {
110             this.within = extractWithin();
111             this.annotation = extractAnnotation(); //Extrae una linea
                    que contiene la anotación
112             this.extractDeclarationsToMap();
113         }
114         if (null != annotation) {
115             this.extractIconToMap();
116         }
117     }
118
119     /**
120      * Extrae los parametros que contiene el modelo de modelica,
             incluido los
121      * heredados.
122      *
```

```
123         * @return ModelicaParameter.
124         */
125        public ModelicaParameter getCompletedModelicaParameters() {
126            ModelicaParameter parameters = new ModelicaParameter();
127            String comment;
128            Pattern pattern = Pattern.compile("\\\"([^\\\"]+)\\\"");
129            Matcher matcher;
130            if (getWithin().isEmpty()) {
131                comment = codeList.get(0);
132                if (!comment.contains("\"")) {
133                    comment = codeList.get(1);
134                }
135            } else {
136                comment = codeList.get(1);
137                if (!comment.contains("\\\"")) {
138                    comment = codeList.get(2);
139                }
140            }
141            matcher = pattern.matcher(comment);
142            if (matcher.find()) {
143                comment = matcher.group(1);
144                parameters.setComment(comment.strip());
145            }
146
147            parameters.setNameComponent(this.componentName);
148            parameters.setComponents(getAllCompletedComponents()); //Extrae
                todos los parametros
149
150            String patternStr = "defaultComponentName\\s?=\\s?\\\"([^\\\"]+)
                \\\"";
151            pattern = Pattern.compile(patternStr);
152            matcher = pattern.matcher(this.annotation);
153            String defaulComponentName;
154            if (matcher.find()) {
155                defaulComponentName = matcher.group(1);
156                parameters.setDefaultComponentName(defaulComponentName.strip
                    ());
157            } else {
158                parameters.setDefaultComponentName(this.getComponentName().
                    toLowerCase());//nombre de la clase
159
160            }
161
162            patternStr = "defaultComponentPrefixes\\s?=\\s?\\\"([^\\\"]+)
                \\\"";
163            pattern = Pattern.compile(patternStr);
164            matcher = pattern.matcher(this.annotation);
165            String defaulComponentPrefixes;
166            if (matcher.find()) {
167                defaulComponentPrefixes = matcher.group(1);
168                parameters.setDefaultComponentPrefix(defaulComponentPrefixes
                    .strip());
169            }
170
171            patternStr = "missingInnerMessage\\s?=\\s?\\\"([^\\\"]+)\\\"";
172            pattern = Pattern.compile(patternStr);
173            matcher = pattern.matcher(this.annotation);
174            String missingInnerMessage;
175            if (matcher.find()) {
176                missingInnerMessage = matcher.group(1);
```

```
177             parameters.setMissingInnerMessage(missingInnerMessage.strip
                    ());
178         }
179         return parameters;
180     }
181 
182     public ModelicaParameter getModelicaParameters() {
183         ModelicaParameter parameters = new ModelicaParameter();
184         String comment;
185         Pattern pattern = Pattern.compile("\\\"([^\\\"]+)\\\"");
186         Matcher matcher;
187         if (getWithin().isEmpty()) {
188             comment = codeList.get(0);
189             if (!comment.contains("\"")) {
190                 comment = codeList.get(1);
191             }
192         } else {
193             comment = codeList.get(1);
194             if (!comment.contains("\\\"")) {
195                 comment = codeList.get(2);
196             }
197         }
198         matcher = pattern.matcher(comment);
199         if (matcher.find()) {
200             comment = matcher.group(1);
201         }
202 
203         parameters.setNameComponent(this.componentName);
204         parameters.setComment(comment);
205         parameters.setComponents(getAllLocalComponents());
206 
207         String patternStr = "defaultComponentName\\s?=\\s?\\\"([^\\\"]+)
                \\\"";
208         pattern = Pattern.compile(patternStr);
209         matcher = pattern.matcher(this.annotation);
210         String defaulComponentName;
211         if (matcher.find()) {
212             defaulComponentName = matcher.group(1);
213             parameters.setDefaultComponentName(defaulComponentName);
214         } else {
215             parameters.setDefaultComponentName(this.getComponentName().
                    toLowerCase());//nombre de la clase
216 
217         }
218 
219         patternStr = "defaultComponentPrefixes\\s?=\\s?\\\"([^\\\"]+)
                \\\"";
220         pattern = Pattern.compile(patternStr);
221         matcher = pattern.matcher(this.annotation);
222         String defaulComponentPrefixes;
223         if (matcher.find()) {
224             defaulComponentPrefixes = matcher.group(1);
225             parameters.setDefaultComponentPrefix(defaulComponentPrefixes
                    );
226         }
227 
228         patternStr = "missingInnerMessage\\s?=\\s?\\\"([^\\\"]+)\\\"";
229         pattern = Pattern.compile(patternStr);
230         matcher = pattern.matcher(this.annotation);
231         String missingInnerMessage;
232         if (matcher.find()) {
```

```
233             missingInnerMessage = matcher.group(1);
234             parameters.setMissingInnerMessage(missingInnerMessage);
235         }
236         return parameters;
237     }
238 
239     public List<ComponentModel> getAllCompletedComponents() {
240         List<ComponentModel> components = new ArrayList<>();
241         for (String line : this.completeDeclarations.get("parameters"))
                {
242             components.add(extractComponentByLine(line));
243         }
244         //extraer los replaceable como componentes
245         for (String line : this.completeDeclarations.get("replaceable"))
                {
246             components.add(extractComponentByLine(line));
247         }
248         //extraer los variables que tengan dialogo: en pruebas
249         for (String line : this.completeDeclarations.get("variables")) {
250             if(line.contains("Dialog")){
251                 components.add(extractComponentByLine(line));
252             }
253         }
254 
255         //Test: Eliminar repetidos
256         List<ComponentModel> uniqueComponents = new ArrayList<>();
257         for(ComponentModel componentModel:components){
258             if(findComponentModelByName(uniqueComponents, componentModel
                    .getName())==null){
259                 uniqueComponents.add(componentModel);
260             }
261         }
262 
263         return uniqueComponents;
264     }
265 
266     private ComponentModel findComponentModelByName(List<ComponentModel>
             componentModelList ,String name){
267         for(ComponentModel componentModel:componentModelList){
268             if(componentModel.getName().equals(name)){
269                 return componentModel;
270             }
271         }
272         return null;
273     }
274 
275     public List<ComponentModel> getAllLocalComponents() {
276 
277         List<ComponentModel> components = new ArrayList<>();
278         for (String line : this.declarations.get("parameters")) {
279             components.add(extractComponentByLine(line));
280         }
281         return components;
282     }
283 
284     public Map<String, String> getRedefinitionsByLineString(String line)
             {
285         String annotationLine = extractElementsByParentesis(line, "
                annotation");
286         if (annotationLine != null) {
```

```
287             line = line.substring(0, line.indexOf("annotation"));//
                    eliminar la annotation
288             line = line.replace(";", "").strip();//eliminar ;
289         }
290         // extraer las redefiniciones
291         String redefinitionsStr = line.substring(line.indexOf("(") + 1);
292         if (redefinitionsStr.length() > 0) {
293             redefinitionsStr = redefinitionsStr.strip();
294         }
295 
296         HashMap<String, String> redefinitions = new HashMap<>();
297         Pattern pattern;
298         Matcher matcher;
299         //test: eliminar , en las redefiniciones
300         if(redefinitionsStr.contains("(")){
301             pattern = Pattern.compile("(\\(.*?\\))");
302             matcher = pattern.matcher(redefinitionsStr);
303             while (matcher.find()) {
304                 String oldValue = matcher.group(1);
305                 String newValue = oldValue.replaceAll(",", ";");
306                 newValue = newValue.replaceAll("\\(", "<<");
307                 newValue = newValue.replaceAll("\\)", ">>");
308                 redefinitionsStr = redefinitionsStr.replace(oldValue,
                        newValue);
309             }
310 
311         }
312         pattern = Pattern.compile("([^,=\\s]+|\\w+\\s\\w+\\s\\w+)\\s?=\\
                s?(\\{[^}]*\\}|[^,\\s\\)]+)");
313         matcher = pattern.matcher(redefinitionsStr);
314 
315         while (matcher.find()) {
316             String key = matcher.group(1);
317             String value = matcher.group(2);
318             value = value.replaceAll(";", ","); // volver a poner ,
319             value = value.replaceAll("<<", "("); // volver a poner (
320             value = value.replaceAll(">>", ")"); // volver a poner )
321             redefinitions.put(key, value);
322         }
323         return redefinitions;
324     }
325 
326     public List<String> getTypeAndNameComponentByLine(String line) {
327         String annotationLine = extractElementsByParentesis(line, "
                annotation");
328         if (annotationLine != null) {
329             line = line.replace(annotationLine, "");//eliminar la
                    annotation
330         }
331         line = line.replaceAll("\\(.*\\)", ""); // eliminar la
                redefinición
332         line = line.replaceAll(";", ""); // eliminar ;
333         String[] tempElements = line.split("\\s");
334         String typeComponent = null;
335         String nameComponent = null;
336         List<String> typeAndName = new ArrayList<>();
337         if (tempElements.length > 1) {
338             typeComponent = tempElements[tempElements.length - 2].strip
                    ();
339             typeAndName.add(typeComponent);
```

```
340             nameComponent = tempElements[tempElements.length - 1].strip
                    ();
341             typeAndName.add(nameComponent);
342         }
343         return typeAndName;
344     }
345 
346     public ModelicaConnector getConnectorComponentByLine(String line) {
347         ModelicaConnector connectorComponent = new ModelicaConnector();
348         line = line.replace(";", "").strip();
349         //eliminar la annotation
350         String annotationLine = extractElementsByParentesis(line, "
                annotation");
351         if (annotationLine != null) {
352             String contentAnnotation = annotationLine.replace("
                    annotation", "").strip();
353             line = line.replace(contentAnnotation, "").strip();
354             line = line.replace("annotation", "").strip();
355         }
356         //extraer el comentario
357         int indexComment = line.indexOf(" \"");
358         String comment;
359         if (indexComment > 0) {
360             comment = line.substring(indexComment);
361             connectorComponent.setComment(comment);
362             line = line.replace(comment, "");
363         }
364         // buscar y extraer redeclaraciones
365         if (line.contains("redeclare")) {
366             String redeclarationStr = line.substring(line.indexOf("("));
367             connectorComponent.setRedeclaration(redeclarationStr);
368             line = line.replace(redeclarationStr, "");
369         }
370 
371 
372         //extraer el nombre y el tipo
373         String propitiesStr[] = line.split("\\s");
374         if (propitiesStr.length > 1) {
375             String nameComponet = propitiesStr[1];
376             //verificar si es un array
377             if (nameComponet.contains("[")) {
378                 String indexName = nameComponet.substring(nameComponet.
                        indexOf("[") + 1, nameComponet.indexOf("]"));
379                 connectorComponent.setIndexName(indexName);
380                 connectorComponent.setIsArray(true);
381             }
382             connectorComponent.setName(nameComponet);
383             String typeComponent = propitiesStr[0];
384             connectorComponent.setType(typeComponent);
385             String prefix = line.replace(" " + nameComponet, "").replace
                    (typeComponent, "");
386             connectorComponent.setPrefix(prefix);
387         }
388         return connectorComponent;
389     }
390 
391     private ComponentModel extractComponentByLine(String line) {
392         line = line.replace(";", "");
393         line = line.strip();
394         ComponentModel componentModel = new ComponentModel();
395         Dialog newDialog = new Dialog();
```

```
396         String annotationLine = extractElementsByParentesis(line, "
                annotation");
397         String dialog = null;
398         if (annotationLine != null) {
399             String contentAnnotation = annotationLine.replace("
                    annotation", "").strip();
400             line = line.replace(contentAnnotation, "").strip();
401             line = line.replace("annotation", "").strip();
402             dialog = extractElementsByParentesis(annotationLine, "Dialog
                    ");
403         }
404         //int indexComment = line.indexOf("\"");
405         int indexComment = line.indexOf(" \"");
406         String comment;
407         if (indexComment > 0) {
408             comment = line.substring(indexComment);
409             componentModel.setComment(comment.strip());
410             line = line.replace(comment, "");
411         }
412 
413         line = line.replaceAll("\\(.*\\)", ""); // eliminar la
                redefinición
414         if (line.contains("=")) {
415             String value = line.substring(line.indexOf("=") + 1);
416             line = line.replace("=" + value, "");
417             componentModel.setValue(value.strip());
418         }
419 
420         String propitiesStr[] = line.split("\\s");
421         if (propitiesStr.length > 1) {
422             String nameComponet = propitiesStr[propitiesStr.length - 1];
423             componentModel.setName(nameComponet.strip());
424             String typeComponent = propitiesStr[propitiesStr.length -
                    2];
425             componentModel.setType(typeComponent.strip());
426             //String prefix = line.replace(nameComponet, "").replace(
                    typeComponent, "");
427             String prefix = line.replace(typeComponent, "").replace(
                    nameComponet, "");
428             componentModel.setPrefix(prefix.strip());
429         }
430 
431         if (dialog != null) {
432             dialog = dialog.substring(7, dialog.length() - 1);
433             String propities[] = dialog.split(",");
434             for (String propertie : propities) {
435                 String[] values = propertie.split("=");
436                 String newValue = values[1].strip();
437                 newValue = newValue.replaceAll("\\\"", "");
438                 if (propertie.contains("tab")) {
439                     newDialog.setTab(newValue);
440                 } else if (propertie.contains("group")) {
441                     newDialog.setGroup(newValue);
442                 } else if (propertie.contains("enable")) {
443                     boolean newValueBool = Boolean.parseBoolean(newValue
                            );
444                     newDialog.setEnable(newValueBool);
445                 } else if (propertie.contains("ShowStartAttribute")) {
446                     boolean newValueBool = Boolean.parseBoolean(newValue
                            );
447                     newDialog.setShowStartAttribute(newValueBool);
```

```
448                 } else if (propertie.contains("colorSelector")) {
449                     boolean newValueBool = Boolean.parseBoolean(newValue
                            );
450                     newDialog.setColorSelector(newValueBool);
451                 } else if (propertie.contains("groupImage")) {
452                     newDialog.setGroupImage(newValue);
453                 } else if (propertie.contains("connectorSizing")) {
454                     boolean newValueBool = Boolean.parseBoolean(newValue
                            );
455                     newDialog.setConnectorSizing(newValueBool);
456                 }
457             }
458         }
459         componentModel.setDialog(newDialog);
460         return componentModel;
461     }
462 
463     public String getWithin() {
464         return within;
465     }
466 
467     private String extractWithin() {
468         String withinString = codeList.get(0);
469         if (withinString.contains("within")) {
470             withinString = withinString.replace("\\s?within\\s", "").
                    replace(";", "");
471         } else {
472             withinString = "";
473         }
474         return withinString;
475     }
476 
477     public Map<String, String> getConnectionMapByLineString(String line)
             {
478         String connectionStr = extractElementsByParentesis(line, "
                connect");
479         Map<String, String> connectionMap = new HashMap<>();
480         if (connectionStr != null) {
481             connectionStr = connectionStr.replace("connect", "");
482             connectionStr = connectionStr.replaceAll("[\\(\\)]", "");
483             String[] tempValues = connectionStr.split(",");
484 
485             String[] tempValueFirst = tempValues[0].split("\\.");
486             String[] tempValueEnd = tempValues[1].split("\\.");
487             connectionMap.put("firstComponentName", tempValueFirst[0].
                    strip());
488             connectionMap.put("secondComponentName", tempValueEnd[0].
                    strip());
489             String firstConnectorName = tempValueFirst[1];
490             String secondConnectorName = tempValueEnd[1];
491             if (firstConnectorName.contains("[")) {
492                 firstConnectorName = firstConnectorName.substring(0,
                        firstConnectorName.indexOf("["));
493             }
494             if (secondConnectorName.contains("[")) {
495                 secondConnectorName = secondConnectorName.substring(0,
                        secondConnectorName.indexOf("["));
496             }
497             connectionMap.put("firstConnectorName", firstConnectorName);
498             connectionMap.put("secondConnectorName", secondConnectorName
                    );
```

```
499         }
500         return connectionMap;
501     }
502 
503     public LineAnnotation getLineAnnotationOfConnectionByLineString(
            String lineConnection) {
504         return extractConnectionLine(lineConnection);
505     }
506 
507     private LineAnnotation extractConnectionLine(String line) {
508         String lineConnection = extractElementsByParentesis(line, "Line"
                );
509         if (lineConnection == null) {
510             return null;
511         }
512         Pattern pattern;
513         Matcher matcher;
514         LineAnnotation shapeAnnotation = new LineAnnotation();
515         if (lineConnection.startsWith("Line")) {
516             if (lineConnection.contains("origin")) {
517                 pattern = Pattern.compile(propertiesPattern.get("origin"
                        ));
518                 matcher = pattern.matcher(lineConnection);
519                 String origin = matcher.find() ? matcher.group(1) : "";
520                 shapeAnnotation.setOrigin(this.extractOrigin(origin));
521             }
522 
523             if (lineConnection.contains("color")) {
524                 pattern = Pattern.compile(propertiesPattern.get("color")
                        );
525                 matcher = pattern.matcher(lineConnection);
526                 String color = matcher.find() ? matcher.group(1) : "";
527                 shapeAnnotation.setColor(this.extractColor(color));
528             }
529 
530             if (lineConnection.contains("pattern")) {
531                 pattern = Pattern.compile(propertiesPattern.get("pattern
                        "));
532                 matcher = pattern.matcher(lineConnection);
533                 String linePattern = matcher.find() ? matcher.group(1) :
                         "";
534                 shapeAnnotation.setPattern(this.extractStringProperty(
                        linePattern, "LinePattern"));
535             }
536 
537             if (lineConnection.contains("thickness")) {
538                 pattern = Pattern.compile(propertiesPattern.get("
                        thickness"));
539                 matcher = pattern.matcher(lineConnection);
540                 String thickness = matcher.find() ? matcher.group(1) : "
                        ";
541                 shapeAnnotation.setThickness(this.extractDoubleProperty(
                        thickness));
542             }
543 
544             if (lineConnection.contains("rotation")) {
545                 pattern = Pattern.compile(propertiesPattern.get("
                        rotation"));
546                 matcher = pattern.matcher(lineConnection);
547                 String rotation = matcher.find() ? matcher.group(1) : ""
                        ;
```

```
548                 shapeAnnotation.setRotation(this.extractDoubleProperty(
                        rotation));
549             }
550 
551             if (lineConnection.contains("arrow")) {
552                 pattern = Pattern.compile(propertiesPattern.get("
                        rotation"));
553                 matcher = pattern.matcher(lineConnection);
554                 String rotation = matcher.find() ? matcher.group(1) : ""
                        ;
555                 //shapeAnnotation.setStartArrows(this.
                        extractDoubleProperty(item));
556             }
557 
558             if (lineConnection.contains("arrowSize")) {
559                 pattern = Pattern.compile(propertiesPattern.get("
                        arrowSize"));
560                 matcher = pattern.matcher(lineConnection);
561                 String arrowSize = matcher.find() ? matcher.group(1) : "
                        ";
562                 shapeAnnotation.setArrowSize(this.extractDoubleProperty(
                        arrowSize));
563             }
564 
565             if (lineConnection.contains("points")) {
566                 pattern = Pattern.compile(propertiesPattern.get("points"
                        ));
567                 matcher = pattern.matcher(lineConnection);
568                 String points = matcher.find() ? matcher.group() : "";
569                 shapeAnnotation.setPoints(this.extractPoints(points));
570             }
571         }
572         return shapeAnnotation;
573     }
574 
575     public Placement getPlacement(String annotation) {
576         this.extractPlacement(annotation);
577         return placement;
578     }
579 
580     private void extractPlacement(String line) {
581         placement = new Placement();
582         //extract visible
583         String placementStr = extractElementsByParentesis(line, "
                Placement");
584         Pattern pattern = Pattern.compile("\\bvisible\\s?=\\s?(\\w+)");
585         Matcher matcher = pattern.matcher(placementStr.replace("\\s", ""
                ));
586         String visibleStr = null;
587         if (matcher.find()) {
588             visibleStr = matcher.group(1);
589             boolean visible = Boolean.parseBoolean(visibleStr);
590             placement.setVisible(visible);
591         }
592 
593         Transformation transformation = new Transformation();
594         String transformationStr = extractElementsByParentesis(line, "
                transformation");
595         pattern = Pattern.compile("origin=(\\{.*?\\})");
596         matcher = pattern.matcher(transformationStr.replace("\\s", ""));
597         if (matcher.find()) {
```

```
598             String originStr = matcher.group(1);
599             transformation.setOrigin(extractOrigin(originStr));
600         }
601 
602         pattern = Pattern.compile("extent=(\\{\\{.*?\\}\\})");
603         matcher = pattern.matcher(transformationStr.replace("\\s", ""));
604         if (matcher.find()) {
605             String extentStr = matcher.group(1);
606             transformation.setExtent(extractExtent(extentStr));
607         }
608 
609         pattern = Pattern.compile("\\brotation=(-?[\\d+\\.]+)");
610         matcher = pattern.matcher(transformationStr.replace("\\s", ""));
611         if (matcher.find()) {
612             String rotationStr = matcher.group(1);
613             transformation.setRotation(extractDoubleProperty(rotationStr
                    ));
614         }
615         placement.setTransformation(transformation);
616     }
617 
618     public Map<String, List<String>> getDeclarations() {
619         return declarations;
620     }
621 
622     /**
623      * Permite extraer las declaraciones: importaciones, herencias,
             parametros.
624      *
625      */
626     public void extractDeclarationsToMap() {
627         List<String> extendsList = new ArrayList<>();
628         List<String> parameterList = new ArrayList<>();
629         List<String> importList = new ArrayList<>();
630         List<String> variableList = new ArrayList<>();
631         List<String> equationList = new ArrayList<>();
632         List<String> protectedList = new ArrayList<>();
633         List<String> replaceableList = new ArrayList<>();
634         boolean startDeclaration = false;
635         boolean startEquation = false;
636         boolean startProtected = false;
637         boolean firstLine = false; //primera linea de declaraciones
638         Matcher matcher;
639         for (String line : codeList) {
640             line = line.strip();
641             if (line.startsWith("replaceable")) {
642                 replaceableList.add(line);
643                 continue;
644             }
645             if (line.startsWith("annotation")) {
646                 continue;// no se toman en cuenta las anotaciones.
647             }
648             String startComponent = this.componentType + " " + this.
                    componentName;
649             if (line.startsWith(startComponent)) {
650                 startDeclaration = true;
651                 firstLine = true;
652                 continue;
653             }
654             if (firstLine) {
655                 String regex = "(\".*?\")";
```

```
656                 Pattern pattern = Pattern.compile(regex);
657                 matcher = pattern.matcher(line);
658                 String extrectedComment = "";
659                 if (matcher.find()) {
660                     extrectedComment = matcher.group(1);
661                 }
662                 line = line.replace(extrectedComment, "").strip();//
                        elimina comentario del componente
663                 firstLine = false;
664             }
665             if (line.startsWith("protected")) {
666                 startProtected = true;
667                 line = line.replace("protected", "").strip();
668             }
669             if (line.startsWith("::equation") || line.startsWith("::
                    initial equation")) {
670                 startDeclaration = false;
671                 startProtected = false;
672                 startEquation = true;
673                 continue;
674             }
675             String endComponent = "end" + " " + this.componentName + ";"
                    ;
676             if (line.startsWith(endComponent)) {
677                 startDeclaration = false;
678                 startProtected = false;
679                 startEquation = false;
680             }
681
682             if (startProtected) {
683                 protectedList.add(line);
684             } else if (startDeclaration) {
685                 line = line.strip();
686                 if (line.contains("parameter")) {
687                     parameterList.add(line);
688                 } else if (line.contains("import")) {
689                     importList.add(line);
690                 } else if (line.contains("extends")) {
691                     extendsList.add(line);
692                 } else {
693                     variableList.add(line);
694                 }
695             }
696             if (startEquation) {
697                 equationList.add(line);
698             }
699         }
700
701         declarations.put("imports", importList);
702         declarations.put("extends", extendsList);
703         declarations.put("parameters", parameterList);
704         declarations.put("protected", protectedList);
705         declarations.put("replaceable", replaceableList);
706         declarations.put("variables", variableList);
707         declarations.put("equations", equationList);
708
709     }
710
711     /**
712      * Extrae y crea el Map de los elementos(primitivos) que contiene el
            icono.
```

```
713      *
714      */
715     public void extractIconToMap() {
716         Pattern pattern;
717         Matcher matcher;
718         String iconStr = extractElementsByParentesis(annotation, "icon")
                ;
719         if (iconStr != null) {
720             String sysCoordStr = extractElementsByParentesis(iconStr, "
                    coordinateSystem");
721             if (sysCoordStr != null) {
722                 //extraer las componentes de coordinateSystem
723                 pattern = Pattern.compile("\\bpreserveAspectRatio=(\\w+)
                        ");
724                 matcher = pattern.matcher(sysCoordStr.replace("\\s", "")
                        );
725                 String preserveAspectRatio = null;
726                 List<String> sysCoordComponentes = new ArrayList<>();
727                 if (matcher.find()) {
728                     preserveAspectRatio = matcher.group();
729                     sysCoordComponentes.add(preserveAspectRatio);
730                 }
731                 pattern = Pattern.compile("extent=\\{\\{[-+]?\\d+(\\.\\d
                        +)?,[-+]?\\d+(\\.\\d+)?\\},\\{[-+]?\\d+(\\.\\d+)
                        ?,[-+]?\\d+(\\.\\d+)?\\}\\}");
732                 matcher = pattern.matcher(sysCoordStr.replace("\\s", "")
                        );
733                 String extentStr = null;
734                 if (matcher.find()) {
735                     extentStr = matcher.group();
736                     sysCoordComponentes.add(extentStr);
737                 }
738                 //Extraer la escala.
739                 pattern = Pattern.compile("\\binitialScale\\s*=\\s*\\d
                        +(\\.\\d+)?");
740                 matcher = pattern.matcher(sysCoordStr.replace("\\s", "")
                        );
741                 String scaleStr = null;
742                 if (matcher.find()) {
743                     scaleStr = matcher.group();
744                     sysCoordComponentes.add(scaleStr);
745                 }
746                 iconMap.put("coordinateSystem", sysCoordComponentes);
747             }
748             //extraer componentes del icono sin dynamicSelect
749             pattern = Pattern.compile("(Rectangle|Ellipse|Line|Polygon|
                    Text|Bitmap)\\([^\\(\\)]+\\)");
750             matcher = pattern.matcher(iconStr);
751             List<String> iconComponentes = new ArrayList<>();
752             while (matcher.find()) {
753                 String componente = matcher.group();
754                 iconComponentes.add(componente);
755             }
756             iconMap.put("icon", iconComponentes);
757             //Extraer icono con DynamicSelect
758             pattern = Pattern.compile("(Rectangle|Ellipse|Polygon|Line|
                    Text|Bitmap)\\([^()]*DynamicSelect\\s*\\(([^\\)]+\\)\\s
                    *[,\\}]*)");
759             matcher = pattern.matcher(iconStr);
760             while (matcher.find()) {
761                 int indexStart = matcher.start();
```

```
762                 String typeComponent = matcher.group(1);
763                 String componentToFind = iconStr.substring(indexStart);
764                 String componente = extractElementsByParentesis(
                        componentToFind, typeComponent);
765                 if (componente.contains("DynamicSelect")) { //tratar los
                        DynamicSselect:
766                     Pattern patternAux = Pattern.compile("(\\w+\\s*)=\\s
                            *DynamicSelect\\s*\\(([^\\)]+\\)\\s*[,\\}]*)");
767                     Matcher matcherAux = patternAux.matcher(componente);
768                     while (matcherAux.find()) {
769                         String typeDynamicSelect = matcherAux.group(1);
770                         String allDynamicSelect = matcherAux.group();
771                         if (typeDynamicSelect.contains("extent")) {
772                             Pattern patternAux2 = Pattern.compile("
                                    (\\{\\{-?[\\d+\\.]+,-?[\\d
                                    +\\.]+\\},\\{-?[\\d+\\.]+,-?[\\d
                                    +\\.]+\\}\\})");
773                             Matcher matcherAux2 = patternAux2.matcher(
                                    allDynamicSelect);
774                             if (matcherAux2.find()) {
775                                 String extendNum = matcherAux2.group();
776                                 componente = componente.replace(
                                        allDynamicSelect, "extent=" +
                                        extendNum + ",");
777                             }
778                         } else if (typeDynamicSelect.contains("
                                textString")) {
779                             Pattern patternAux2 = Pattern.compile("
                                    (\\\".*\\\")");
780                             Matcher matcherAux2 = patternAux2.matcher(
                                    allDynamicSelect);
781                             if (matcherAux2.find()) {
782                                 String textString = matcherAux2.group();
783                                 componente = componente.replace(
                                        allDynamicSelect, "textString=" +
                                        textString);
784                             }
785                         }
786                     }
787                 }
788                 iconComponentes.add(componente);
789             }
790             iconMap.put("icon", iconComponentes);
791         }
792     }
793
794     /**
795      * Extrae elementos que tengan un nombre y elementos entre
            parentesis de un
796      * String.
797      *
798      * @param line String de donde se va extraer el elemento que tenta
            @param
799      * name.
800      * @param name Nombre del componente a extraer.
801      * @return String con elemento extraido.
802      */
803     private String extractElementsByParentesis(String line, String name)
            {
804         line = line.strip();
805         String patternStr = name + "\\s?\\(";
```

```
806         Pattern pattern = Pattern.compile(patternStr, Pattern.
                CASE_INSENSITIVE);
807         Matcher matcher = pattern.matcher(line);
808         if (matcher.find()) {
809             line = line.substring(matcher.end());
810             StringBuilder sb = new StringBuilder();
811             int parrentesisCount = 1;
812             for (char c : line.toCharArray()) {
813                 if (c == '(') {
814                     parrentesisCount++;
815                 } else if (c == ')') {
816                     parrentesisCount--;
817                 }
818                 if (parrentesisCount <= 0) {
819                     break;
820                 } else {
821                     sb.append(c);
822                 }
823             }
824             return name + "(" + sb.toString() + ")";
825         }
826         return null;
827     }
828 
829     /**
830      * Extrae la anotación del icono.
831      *
832      */
833     private String extractAnnotation() {
834         String line = "";
835         Pattern pattern = Pattern.compile("^\\bannotation\\(");
836         Matcher matcher;
837         for (int i = 0; i < codeList.size(); i++) {
838             line = codeList.get(i).strip().replaceAll("\\s", "");
839             matcher = pattern.matcher(line);
840 
841             if (matcher.find()) {
842                 return line;
843             }
844         }
845         return null;
846     }
847 
848     /**
849      * Devuelve la anotación en formato String.
850      * @return anotacion en formato String.     *
851      */
852     public String getAnnotation() {
853         return this.annotation;
854     }
855 
856     /**
857      * Icono extraido de las anotaciones.
858      *
859      * @return IconAnnotation.     *
860      */
861     public IconAnnotation getIconPane() {
862         if (annotation == null) {
863             return null;
864         }
865         return this.parseIconTree();
```

```
866     }
867
868     /**
869      * Genera un IconAnnotation(icono) a partir de cada elemento que
             conforma el
870      * icono (primitivas).     *
871      */
872     private IconAnnotation parseIconTree() {
873         if (iconMap.isEmpty()) {
874             return null;
875         }
876         Pattern pattern;
877         Matcher matcher;
878         //create icon
879         IconAnnotation icon = new IconAnnotation();
880         //extract coordinate system del icon
881         if (iconMap.get("coordinateSystem") != null) {
882             CoordinateSystem coordinateSystem = new CoordinateSystem();
883             for (String item : iconMap.get("coordinateSystem")) {
884                 if (item.contains("preserveAspectRatio")) {
885                     item = item.replace("preserveAspectRatio=", "");
886                     boolean value = Boolean.parseBoolean(item);
887                     coordinateSystem.setPreserveAspectRatio(value);
888                 } else if (item.contains("extent")) {
889                     coordinateSystem.setExent(this.extractExtent(item));
890                 } else if (item.contains("initialScale")) {
891                     item = item.replace("initialScale=", "");
892                     double value = Double.parseDouble(item);
893                     coordinateSystem.setInitialScale(value);
894                 }
895             }
896             icon.setCoordinateSystem(coordinateSystem);
897         }
898         // extract elements del icono
899         for (String item : iconMap.get("icon")) {
900             if (item.startsWith("Rectangle")) {
901                 RectangleAnnotation shapeAnnotation = new
                        RectangleAnnotation();
902                 FilledShape filledShape = new FilledShape();
903                 if (item.contains("origin")) {
904                     pattern = Pattern.compile(propertiesPattern.get("
                            origin"));
905                     matcher = pattern.matcher(item);
906                     String origin = matcher.find() ? matcher.group(1) :
                            "";
907                     shapeAnnotation.setOrigin(this.extractOrigin(origin)
                            );
908                 }
909
910                 if (item.contains("extent")) {
911                     pattern = Pattern.compile(propertiesPattern.get("
                            extent"));
912                     matcher = pattern.matcher(item);
913                     String extent = matcher.find() ? matcher.group(1) :
                            "";
914                     shapeAnnotation.setExtent(this.extractExtent(extent)
                            );
915                 }
916
917                 if (item.contains("lineColor")) {
```

```
918                     pattern = Pattern.compile(propertiesPattern.get("
                            lineColor"));
919                     matcher = pattern.matcher(item);
920                     String lineColor = matcher.find() ? matcher.group(1)
                             : "";
921                     filledShape.setLineColor(this.extractColor(lineColor
                            ));
922                 }
923
924                 if (item.contains("fillColor")) {
925                     pattern = Pattern.compile(propertiesPattern.get("
                            fillColor"));
926                     matcher = pattern.matcher(item);
927                     String fillColor = matcher.find() ? matcher.group(1)
                             : "";
928                     filledShape.setFillColor(this.extractColor(fillColor
                            ));
929                 }
930
931                 if (item.contains("fillPattern")) {
932                     pattern = Pattern.compile(propertiesPattern.get("
                            fillPattern"));
933                     matcher = pattern.matcher(item);
934                     String fillPattern = matcher.find() ? matcher.group
                            (1) : "";
935                     filledShape.setFillPattern(this.
                            extractStringProperty(fillPattern, "FillPattern"
                            ));
936                 }
937
938                 if (item.contains("pattern")) {
939                     pattern = Pattern.compile(propertiesPattern.get("
                            pattern"));
940                     matcher = pattern.matcher(item);
941                     String linePattern = matcher.find() ? matcher.group
                            (1) : "";
942                     filledShape.setPattern(this.extractStringProperty(
                            linePattern, "LinePattern"));
943                 }
944
945                 if (item.contains("lineThickness")) {
946                     pattern = Pattern.compile(propertiesPattern.get("
                            lineThickness"));
947                     matcher = pattern.matcher(item);
948                     String lineThickness = matcher.find() ? matcher.
                            group(1) : "";
949                     filledShape.setLineThickness(this.
                            extractDoubleProperty(lineThickness));
950                 }
951
952                 if (item.contains("rotation")) {
953                     pattern = Pattern.compile(propertiesPattern.get("
                            rotation"));
954                     matcher = pattern.matcher(item);
955                     String rotation = matcher.find() ? matcher.group(1)
                             : "";
956                     shapeAnnotation.setRotation(this.
                            extractDoubleProperty(rotation));
957                 }
958
959                 if (item.contains("radius")) {
```

```
960                     pattern = Pattern.compile(propertiesPattern.get("
                            radius"));
961                     matcher = pattern.matcher(item);
962                     String radius = matcher.find() ? matcher.group(1) :
                            "";
963                     shapeAnnotation.setRadius(this.extractDoubleProperty
                            (radius));
964                 }
965                 shapeAnnotation.setFilledShape(filledShape);
966                 icon.add(shapeAnnotation);
967             } else if (item.startsWith("Ellipse")) {
968                 EllipseAnnotation shapeAnnotation = new
                        EllipseAnnotation();
969                 FilledShape filledShape = new FilledShape();
970                 if (item.contains("origin")) {
971                     pattern = Pattern.compile(propertiesPattern.get("
                            origin"));
972                     matcher = pattern.matcher(item);
973                     String origin = matcher.find() ? matcher.group(1) :
                            "";
974                     shapeAnnotation.setOrigin(this.extractOrigin(origin)
                            );
975                 }
976
977                 if (item.contains("extent")) {
978                     pattern = Pattern.compile(propertiesPattern.get("
                            extent"));
979                     matcher = pattern.matcher(item);
980                     String extent = matcher.find() ? matcher.group(1) :
                            "";
981                     shapeAnnotation.setExtent(this.extractExtent(extent)
                            );
982                 }
983
984                 if (item.contains("lineColor")) {
985                     pattern = Pattern.compile(propertiesPattern.get("
                            lineColor"));
986                     matcher = pattern.matcher(item);
987                     String lineColor = matcher.find() ? matcher.group(1)
                             : "";
988                     filledShape.setLineColor(this.extractColor(lineColor
                            ));
989                 }
990
991                 if (item.contains("fillColor")) {
992                     pattern = Pattern.compile(propertiesPattern.get("
                            fillColor"));
993                     matcher = pattern.matcher(item);
994                     String fillColor = matcher.find() ? matcher.group(1)
                             : "";
995                     filledShape.setFillColor(this.extractColor(fillColor
                            ));
996                 }
997
998                 if (item.contains("fillPattern")) {
999                     pattern = Pattern.compile(propertiesPattern.get("
                            fillPattern"));
1000                    matcher = pattern.matcher(item);
1001                    String fillPattern = matcher.find() ? matcher.group
                            (1) : "";
```

```
1002                    filledShape.setFillPattern(this.
                            extractStringProperty(fillPattern, "FillPattern"
                            ));
1003                }
1004
1005                if (item.contains("pattern")) {
1006                    pattern = Pattern.compile(propertiesPattern.get("
                            pattern"));
1007                    matcher = pattern.matcher(item);
1008                    String linePattern = matcher.find() ? matcher.group
                            (1) : "";
1009                    filledShape.setPattern(this.extractStringProperty(
                            linePattern, "LinePattern"));
1010                }
1011
1012                if (item.contains("lineThickness")) {
1013                    pattern = Pattern.compile(propertiesPattern.get("
                            lineThickness"));
1014                    matcher = pattern.matcher(item);
1015                    String lineThickness = matcher.find() ? matcher.
                            group(1) : "";
1016                    filledShape.setLineThickness(this.
                            extractDoubleProperty(lineThickness));
1017                }
1018
1019                if (item.contains("rotation")) {
1020                    pattern = Pattern.compile(propertiesPattern.get("
                            rotation"));
1021                    matcher = pattern.matcher(item);
1022                    String rotation = matcher.find() ? matcher.group(1)
                            : "";
1023                    shapeAnnotation.setRotation(this.
                            extractDoubleProperty(rotation));
1024                }
1025
1026                if (item.contains("startAngle")) {
1027                    pattern = Pattern.compile(propertiesPattern.get("
                            startAngle"));
1028                    matcher = pattern.matcher(item);
1029                    String startAngle = matcher.find() ? matcher.group
                            (1) : "";
1030                    shapeAnnotation.setStartAngle(this.
                            extractDoubleProperty(startAngle));
1031                }
1032                if (item.contains("endAngle")) {
1033                    pattern = Pattern.compile(propertiesPattern.get("
                            endAngle"));
1034                    matcher = pattern.matcher(item);
1035                    String endAngle = matcher.find() ? matcher.group(1)
                            : "";
1036                    shapeAnnotation.setEndAngle(this.
                            extractDoubleProperty(endAngle));
1037                }
1038                shapeAnnotation.setFilledShape(filledShape);
1039                icon.add(shapeAnnotation);
1040            } else if (item.startsWith("Polygon")) {
1041                PolygonAnnotation shapeAnnotation = new
                        PolygonAnnotation();
1042                FilledShape filledShape = new FilledShape();
1043                if (item.contains("origin")) {
```

```
1044                    pattern = Pattern.compile(propertiesPattern.get("
                            origin"));
1045                    matcher = pattern.matcher(item);
1046                    String origin = matcher.find() ? matcher.group(1) :
                            "";
1047                    shapeAnnotation.setOrigin(this.extractOrigin(origin)
                            );
1048                }
1049
1050                if (item.contains("lineColor")) {
1051                    pattern = Pattern.compile(propertiesPattern.get("
                            lineColor"));
1052                    matcher = pattern.matcher(item);
1053                    String lineColor = matcher.find() ? matcher.group(1)
                             : "";
1054                    filledShape.setLineColor(this.extractColor(lineColor
                            ));
1055                }
1056
1057                if (item.contains("fillColor")) {
1058                    pattern = Pattern.compile(propertiesPattern.get("
                            fillColor"));
1059                    matcher = pattern.matcher(item);
1060                    String fillColor = matcher.find() ? matcher.group(1)
                             : "";
1061                    filledShape.setFillColor(this.extractColor(fillColor
                            ));
1062                }
1063
1064                if (item.contains("fillPattern")) {
1065                    pattern = Pattern.compile(propertiesPattern.get("
                            fillPattern"));
1066                    matcher = pattern.matcher(item);
1067                    String fillPattern = matcher.find() ? matcher.group
                            (1) : "";
1068                    filledShape.setFillPattern(this.
                            extractStringProperty(fillPattern, "FillPattern"
                            ));
1069                }
1070
1071                if (item.contains("pattern")) {
1072                    pattern = Pattern.compile(propertiesPattern.get("
                            pattern"));
1073                    matcher = pattern.matcher(item);
1074                    String linePattern = matcher.find() ? matcher.group
                            (1) : "";
1075                    filledShape.setPattern(this.extractStringProperty(
                            linePattern, "LinePattern"));
1076                }
1077
1078                if (item.contains("lineThickness")) {
1079                    pattern = Pattern.compile(propertiesPattern.get("
                            lineThickness"));
1080                    matcher = pattern.matcher(item);
1081                    String lineThickness = matcher.find() ? matcher.
                            group(1) : "";
1082                    filledShape.setLineThickness(this.
                            extractDoubleProperty(lineThickness));
1083                }
1084
1085                if (item.contains("rotation")) {
```

```
1086                    pattern = Pattern.compile(propertiesPattern.get("
                            rotation"));
1087                    matcher = pattern.matcher(item);
1088                    String rotation = matcher.find() ? matcher.group(1)
                            : "";
1089                    shapeAnnotation.setRotation(this.
                            extractDoubleProperty(rotation));
1090                }
1091
1092                if (item.contains("points")) {
1093                    pattern = Pattern.compile(propertiesPattern.get("
                            points"));
1094                    matcher = pattern.matcher(item);
1095                    String points = matcher.find() ? matcher.group() : "
                            ";
1096                    shapeAnnotation.setPoints(this.extractPoints(points)
                            );
1097                }
1098                if (item.contains("smooth")) {
1099                    shapeAnnotation.setSmooth(true);
1100                }
1101                shapeAnnotation.setFilledShape(filledShape);
1102                icon.add(shapeAnnotation);
1103
1104            } else if (item.startsWith("Line")) {
1105                LineAnnotation shapeAnnotation = new LineAnnotation();
1106                if (item.contains("origin")) {
1107                    pattern = Pattern.compile(propertiesPattern.get("
                            origin"));
1108                    matcher = pattern.matcher(item);
1109                    String origin = matcher.find() ? matcher.group(1) :
                            "";
1110                    shapeAnnotation.setOrigin(this.extractOrigin(origin)
                            );
1111                }
1112
1113                if (item.contains("color")) {
1114                    pattern = Pattern.compile(propertiesPattern.get("
                            color"));
1115                    matcher = pattern.matcher(item);
1116                    String color = matcher.find() ? matcher.group(1) : "
                            ";
1117                    shapeAnnotation.setColor(this.extractColor(color));
1118                }
1119
1120                if (item.contains("pattern")) {
1121                    pattern = Pattern.compile(propertiesPattern.get("
                            pattern"));
1122                    matcher = pattern.matcher(item);
1123                    String linePattern = matcher.find() ? matcher.group
                            (1) : "";
1124                    shapeAnnotation.setPattern(this.
                            extractStringProperty(linePattern, "LinePattern"
                            ));
1125                }
1126
1127                if (item.contains("thickness")) {
1128                    pattern = Pattern.compile(propertiesPattern.get("
                            thickness"));
1129                    matcher = pattern.matcher(item);
```

```
1130                    String thickness = matcher.find() ? matcher.group(1)
                            : "";
1131                    shapeAnnotation.setThickness(this.
                            extractDoubleProperty(thickness));
1132                }
1133
1134                if (item.contains("rotation")) {
1135                    pattern = Pattern.compile(propertiesPattern.get("
                            rotation"));
1136                    matcher = pattern.matcher(item);
1137                    String rotation = matcher.find() ? matcher.group(1)
                            : "";
1138                    shapeAnnotation.setRotation(this.
                            extractDoubleProperty(rotation));
1139                }
1140
1141                if (item.contains("arrow")) {
1142                    pattern = Pattern.compile(propertiesPattern.get("
                            rotation"));
1143                    matcher = pattern.matcher(item);
1144                    String rotation = matcher.find() ? matcher.group(1)
                            : "";
1145                    //shapeAnnotation.setStartArrows(this.
                            extractDoubleProperty(item));
1146                }
1147
1148                if (item.contains("arrowSize")) {
1149                    pattern = Pattern.compile(propertiesPattern.get("
                            arrowSize"));
1150                    matcher = pattern.matcher(item);
1151                    String arrowSize = matcher.find() ? matcher.group(1)
                            : "";
1152                    shapeAnnotation.setArrowSize(this.
                            extractDoubleProperty(arrowSize));
1153                }
1154
1155                if (item.contains("points")) {
1156                    pattern = Pattern.compile(propertiesPattern.get("
                            points"));
1157                    matcher = pattern.matcher(item);
1158                    String points = matcher.find() ? matcher.group() : "
                            ";
1159                    shapeAnnotation.setPoints(this.extractPoints(points)
                            );
1160                }
1161                icon.add(shapeAnnotation);
1162
1163            } else if (item.startsWith("Text")) {
1164                TextAnnotation shapeAnnotation = new TextAnnotation();
1165                if (item.contains("origin")) {
1166                    pattern = Pattern.compile(propertiesPattern.get("
                            origin"));
1167                    matcher = pattern.matcher(item);
1168                    String origin = matcher.find() ? matcher.group(1) :
                            "";
1169                    shapeAnnotation.setOrigin(this.extractOrigin(origin)
                            );
1170                }
1171
1172                if (item.contains("textString")) {
```

```
1173                    pattern = Pattern.compile(propertiesPattern.get("
                            textString"));
1174                    matcher = pattern.matcher(item);
1175                    String textString = matcher.find() ? matcher.group
                            (1) : "";
1176                    shapeAnnotation.setTextString(textString);
1177                }
1178
1179                if (item.contains("extent")) {
1180                    pattern = Pattern.compile(propertiesPattern.get("
                            extent"));
1181                    matcher = pattern.matcher(item);
1182                    String extent = matcher.find() ? matcher.group(1) :
                            "";
1183                    shapeAnnotation.setExtent(this.extractExtent(extent)
                            );
1184                }
1185
1186                if (item.contains("textColor")) {
1187                    pattern = Pattern.compile(propertiesPattern.get("
                            textColor"));
1188                    matcher = pattern.matcher(item);
1189                    String color = matcher.find() ? matcher.group(1) : "
                            ";
1190                    shapeAnnotation.setTextColor(this.extractColor(color
                            ));
1191                }
1192
1193                if (item.contains("fontSize")) {
1194                    pattern = Pattern.compile(propertiesPattern.get("
                            fontSize"));
1195                    matcher = pattern.matcher(item);
1196                    String fontSize = matcher.find() ? matcher.group(1)
                            : "";
1197                    shapeAnnotation.setFontSize(this.
                            extractDoubleProperty(fontSize));
1198                }
1199
1200                if (item.contains("fontName")) {
1201                    pattern = Pattern.compile(propertiesPattern.get("
                            fontName"));
1202                    matcher = pattern.matcher(item);
1203                    String fontName = matcher.find() ? matcher.group(1)
                            : "";
1204                    shapeAnnotation.setFontName(fontName);
1205                }
1206
1207                if (item.contains("horizontalAlignment")) {
1208                    pattern = Pattern.compile(propertiesPattern.get("
                            horizontalAlignment"));
1209                    matcher = pattern.matcher(item);
1210                    String horizontalAlignment = matcher.find() ?
                            matcher.group(1) : "";
1211                    shapeAnnotation.setHorizontalAlignament(
                            horizontalAlignment);
1212                }
1213
1214                if (item.contains("textStyle")) {
1215                    pattern = Pattern.compile(propertiesPattern.get("
                            textStyle"));
1216                    matcher = pattern.matcher(item);
```

```
1217                    String textStyle = matcher.find() ? matcher.group(1)
                            : "";
1218                    shapeAnnotation.setTextStyle(this.extractStyles(
                            textStyle));
1219                }
1220
1221                if (item.contains("rotation")) {
1222                    pattern = Pattern.compile(propertiesPattern.get("
                            rotation"));
1223                    matcher = pattern.matcher(item);
1224                    String rotation = matcher.find() ? matcher.group(1)
                            : "";
1225                    shapeAnnotation.setRotation(this.
                            extractDoubleProperty(rotation));
1226                }
1227                icon.add(shapeAnnotation);
1228            } else if (item.startsWith("Bitmap")) {
1229                BitmapAnnotation shapeAnnotation = new BitmapAnnotation
                        ();
1230                if (item.contains("origin")) {
1231                    pattern = Pattern.compile(propertiesPattern.get("
                            origin"));
1232                    matcher = pattern.matcher(item);
1233                    String origin = matcher.find() ? matcher.group(1) :
                            "";
1234                    shapeAnnotation.setOrigin(this.extractOrigin(origin)
                            );
1235                }
1236                if (item.contains("extent")) {
1237                    pattern = Pattern.compile(propertiesPattern.get("
                            extent"));
1238                    matcher = pattern.matcher(item);
1239                    String extent = matcher.find() ? matcher.group(1) :
                            "";
1240                    shapeAnnotation.setExtent(this.extractExtent(extent)
                            );
1241                }
1242                if (item.contains("rotation")) {
1243                    pattern = Pattern.compile(propertiesPattern.get("
                            rotation"));
1244                    matcher = pattern.matcher(item);
1245                    String rotation = matcher.find() ? matcher.group(1)
                            : "";
1246                    shapeAnnotation.setRotation(this.
                            extractDoubleProperty(rotation));
1247                }
1248                if (item.contains("fileName")) {
1249                    pattern = Pattern.compile(propertiesPattern.get("
                            fileName"));
1250                    matcher = pattern.matcher(item);
1251                    String fileName = matcher.find() ? matcher.group(1)
                            : "";
1252                    fileName = fileName.replace("modelica://", rootPath
                            + "\\lib\\");
1253                    fileName = fileName.replace("\\", "/");
1254                    fileName = fileName.replace("/", "\\");
1255                    shapeAnnotation.setFileName(fileName);
1256                    if (DEBUG) {
1257                        System.out.println("Ruta Bitmap: " + fileName);
1258                    }
1259                }
```

```
1260                icon.add(shapeAnnotation);
1261            }
1262        }
1263        return icon;
1264    }
1265
1266    /**
1267     * Extrae y crea un Point2D correspondiente al origen.
1268     *
1269     */
1270    private Point2D extractOrigin(String item) {
1271        // Patrón para extraer las coordenadas de los listPoints
1272        Pattern pattern = Pattern.compile("\\{(-?\\d+\\.?\\d*),(-?\\d
                +\\.?\\d*)\\}");
1273        Matcher matcher = pattern.matcher(item);
1274        try {
1275            if (matcher.find()) {
1276                double x = Double.parseDouble(matcher.group(1));
1277                double y = Double.parseDouble(matcher.group(2));
1278                return new Point2D(x, y);
1279            }
1280        } catch (NumberFormatException e) {
1281            if (DEBUG) {
1282                System.out.println("ERROR al intentar extraer origin de
                        " + item + "\n" + e.getMessage());
1283            }
1284        }
1285        return new Point2D(0, 0); //default
1286    }
1287
1288    private Color extractColor(String item) {
1289        Pattern pattern = Pattern.compile("\\{(-?\\d+),(-?\\d+),(-?\\d+)
                \\}");
1290        Matcher matcher = pattern.matcher(item);
1291        // Extraer los números en grupos
1292        if (matcher.find()) {
1293            int R = Integer.parseInt(matcher.group(1));
1294            int G = Integer.parseInt(matcher.group(2));
1295            int B = Integer.parseInt(matcher.group(3));
1296            return Color.rgb(R, G, B, 1);
1297        }
1298        return null;
1299    }
1300
1301    /**
1302     * Extrae cualquier elemento que tenga un nombre = valor.
1303     *
1304     * @param item linea String de la propiedad.
1305     * @param name nombre que se quiere extraer.
1306     * @return
1307     */
1308    private String extractStringProperty(String item, String name) {
1309        String valueStr = item.substring(item.indexOf("=") + 1);
1310        valueStr = valueStr.replace(name + ".", "");
1311        return valueStr;
1312    }
1313
1314    private List<Point2D> extractPoints(String item) {
1315        // Patrón para extraer las coordenadas de los listPoints
1316        //Pattern pattern = Pattern.compile("\\{(-?\\d+\\.?\\d*),(-?\\d
                +\\.?\\d*)\\}");
```

```
1317        Pattern pattern = Pattern.compile("\\{(-?\\d+\\.?\\d*)\\s?,\\s
                ?(-?\\d+\\.?\\d*)\\}");
1318        Matcher matcher = pattern.matcher(item);
1319        // Lista para almacenar los listPoints
1320        List<Point2D> listPoints = new ArrayList<>();
1321        // Extraer las coordenadas y crear objetos Point2D
1322        while (matcher.find()) {
1323            double x = Double.parseDouble(matcher.group(1));
1324            double y = Double.parseDouble(matcher.group(2));
1325            Point2D point = new Point2D(x, y);
1326            listPoints.add(point);
1327        }
1328        return listPoints;
1329    }
1330
1331    private double extractDoubleProperty(String item) {
1332        String valueStr = item.substring(item.indexOf("=") + 1);
1333        return Double.parseDouble(valueStr);
1334    }
1335
1336    private Extent extractExtent(String item) {
1337        // Patrón para extraer los números en 4 grupos
1338        Pattern pattern = Pattern.compile("\\{(-?[\\d+\\.]+),(-?[\\d
                +\\.]+)\\},\\{(-?[\\d+\\.]+),(-?[\\d+\\.]+)\\}");
1339        Matcher matcher = pattern.matcher(item);
1340        // Extraer los números en 4 grupos
1341        if (matcher.find()) {
1342            double x1 = Double.parseDouble(matcher.group(1));
1343            double y1 = Double.parseDouble(matcher.group(2));
1344            double x2 = Double.parseDouble(matcher.group(3));
1345            double y2 = Double.parseDouble(matcher.group(4));
1346            return new Extent(x1, y1, x2, y2);
1347        }
1348        return null;
1349    }
1350
1351    private List<String> extractStyles(String item) {
1352        List<String> styles = new ArrayList<>();
1353        String valueStr = item.substring(item.indexOf("=") + 1);
1354        valueStr = valueStr.replaceAll("\\{|\\}", "");
1355        String[] values = valueStr.split("[,]");
1356        styles.addAll(Arrays.asList(values));
1357        return styles;
1358    }
1359}
```

Código B.6: Implementación de la clase encargada de analizar texto escrito en lenguaje Modelica.

Código del contenedor del icono Drag and Drop: DraggableNode.java

```
1  package com.fluideditor.model.icon;
2
3  import com.fluideditor.model.modelica.ModelicaConnector;
4  import java.util.List;
5  import javafx.event.EventHandler;
6  import javafx.geometry.Point2D;
7  import javafx.scene.Node;
8  import javafx.scene.input.MouseEvent;
9  import javafx.scene.layout.StackPane;
10
11 public class DraggableNode extends StackPane {
12
13     private double widthContainer = 500; // deberia establecerlo cuando
           se crea el objeto
14     private double heightContainer = 500;
15     private double nodePositionX = 0;
16     private double nodePositionY = 0;
17     private double mousex = 0;
18     private double mousey = 0;
19     private boolean dragging = false;
20     private boolean moveToFront = true;
21     private Placement placement;
22     private List<ModelicaConnector> connectors;
23     private String name;
24
25     public DraggableNode(Node view) {
26         getChildren().add(view);
27         init();
28     }
29
30     public String getName() {
31         return name;
32     }
33
34     public void setName(String name) {
35         this.name = name;
36     }
37
38     public ModelicaConnector getConnectorByName(String name) {
39         for (ModelicaConnector connector : connectors) {
40             if (connector.getName().contains(name)) {
41                 return connector;
42             }
43         }
44         return null;
45     }
46
47     public ModelicaConnector getConnectorByType(String type) {
48         for (ModelicaConnector connector : connectors) {
49             if (connector.getType().equals(type)) {
50                 return connector;
51             }
52         }
53         return null;
54     }
55
56
57     public List<ModelicaConnector> getConnectors() {
58         return connectors;
```

```
59      }
60
61      public void setConnectors(List<ModelicaConnector> connectors) {
62          this.connectors = connectors;
63      }
64
65      public void add(Node node) {
66          getChildren().add(node);
67      }
68
69      public Placement getPlacement() {
70          return placement;
71      }
72
73      public void setPlacement(Placement placement) {
74          this.placement = placement;
75      }
76
77      public void setDragging(boolean dragging) {
78          this.dragging = dragging;
79      }
80
81      private void init() {
82          onMousePressedProperty().set((EventHandler<MouseEvent>) (
                MouseEvent event) -> {
83              mousex = event.getSceneX();
84              mousey = event.getSceneY();
85              nodePositionX = getLayoutX();
86              nodePositionY = getLayoutY();
87              if (isMoveToFront()) {
88                  toFront();
89              }
90              dragging = true;
91          });
92
93          // Evento para el Dragg del icono.
94          onMouseDraggedProperty().set((EventHandler<MouseEvent>) (
                MouseEvent event) -> {
95              if (isDragging()) {
96                  double offsetX = event.getSceneX() - mousex;
97                  double offsetY = event.getSceneY() - mousey;
98                  nodePositionX += offsetX;
99                  nodePositionY += offsetY;
100                 //limitaciones por donde se puede mover el elemento
101                 //if (nodePositionX >= 0 && nodePositionX <=
                        widthContent - getPrefWidth()) {
102                 setLayoutX(nodePositionX);
103                 //}
104                 //if (nodePositionY >= 0 && nodePositionY <=
                        heightContent -getPrefHeight()) {
105                 setLayoutY(nodePositionY);
106                 // }
107                 // actualizar la posición del ratón.
108                 mousex = event.getSceneX();
109                 mousey = event.getSceneY();
110                 updatePlacement(getLayoutX(), getLayoutY());
111             }
112             event.consume();
113         });
114
```

```
115         onMouseClickedProperty().set((EventHandler<MouseEvent>) (
                MouseEvent event) -> {
116             dragging = false;
117         });
118     }
119
120     /**
121      * Mantener actualizado la posición del icono.
122      *
123      * @param x Posición horizontal del icono.
124      * @param y Posición vertical del icono.
125      */
126     private void updatePlacement(double x, double y) {
127         //create placement
128         Transformation transformation = placement.getTransformation();
129         Point2D origin = transformFromSystemToModelicaCoordinate(x, y);
130         transformation.setOrigin(origin);
131
132     }
133
134     private Point2D transformFromSystemToModelicaCoordinate(double x,
            double y) {
135         double targetWidth = 200;
136         double targetHeight = 200;
137         double centerRectangleX = (getWidth() - 4) / 2;
138         double centerRectangleY = (getHeight() - 4) / 2;
139         double localX = (x - (widthContainer / 2) + centerRectangleX) *
                targetWidth / widthContainer;
140         double localY = -(y - (heightContainer / 2) + centerRectangleY)
                * targetHeight / heightContainer;
141         return new Point2D(localX, localY);
142     }
143
144     protected boolean isDragging() {
145         return dragging;
146     }
147
148     /**
149      * @param moveToFront poner en primer plano el icono que se mueve.
150      */
151     public void setMoveToFront(boolean moveToFront) {
152         this.moveToFront = moveToFront;
153     }
154
155     public boolean isMoveToFront() {
156         return moveToFront;
157     }
158
159     public void removeNode(Node node) {
160         getChildren().remove(node);
161     }
162
163     public double getWidthContent() {
164         return widthContainer;
165     }
166
167     public void setWidthContent(double widthContent) {
168         this.widthContainer = widthContent;
169     }
170
171     public double getHeightContent() {
```

```
172         return heightContainer;
173     }
174 
175     public void setHeightContent(double heightContent) {
176         this.heightContainer = heightContent;
177     }
178 
179 }
```

Código B.7: Implementación del contenedor del icono Drag and Drop.

Código del manejador de los iconos: IconManager.java

```
1  package com.fluideditor.model.icon;
2  
3  import com.fluideditor.model.modelica.ModelicaConnector;
4  import com.fluideditor.model.tree.NodeItemCode;
5  import java.util.ArrayList;
6  import java.util.List;
7  import java.util.Map;
8  import javafx.scene.Cursor;
9  import javafx.scene.Group;
10 import javafx.scene.control.TreeItem;
11 import javafx.scene.transform.Rotate;
12 import javafx.scene.transform.Scale;
13 import javafx.scene.transform.Translate;
14 
15 /**
16  * Gestor de los iconos de cada elemento de Modelica.
17  *
18  * @author Jackson F. Reyes Bermeo
19  */
20 public class IconManager {
21 
22     private final String rootPath;
23     private final TreeItem<NodeItemCode> rootTree;
24     private CodeAnalizer codeAnalizer;
25     private final IconAnnotation iconAnnotation;
26     private final NodeItemCode currentNodeCode;
27     private final List<ModelicaConnector> connectors;
28     private final boolean DEBUG = false;
29 
30     public IconManager(TreeItem<NodeItemCode> rootTree, NodeItemCode
           nodeItemCode, String rootPath) {
31         this.rootPath = rootPath;
32         this.rootTree = rootTree;
33         iconAnnotation = new IconAnnotation();
34         currentNodeCode = nodeItemCode;
35         connectors = new ArrayList<>();
36     }
37 
38     public List<ModelicaConnector> getConnectors() {
39         return connectors;
40     }
41 
42     public IconAnnotation getCompleteIconAnnotation() {
43         if (iconAnnotation.getShapes().isEmpty()) {
44             return makeCompleteIconAnnotation(currentNodeCode, null);
45         } else {
```

```
46              return iconAnnotation;
47          }
48
49      }
50
51      public IconAnnotation getPartialIconAnnotation(List<String> codeList
            ) {
52          codeAnalizer = new CodeAnalizer(codeList, rootPath);
53          codeAnalizer.setComponentName(currentNodeCode.getName());
54          codeAnalizer.setComponentType(currentNodeCode.getType());
55          codeAnalizer.analize();
56          return codeAnalizer.getIconPane();
57      }
58
59      /**
60       * Geenera de manera recursiva todos los shapes, los locales, los
               heredados
61       * y los de composición.
62       */
63      private IconAnnotation makeCompleteIconAnnotation(NodeItemCode
            nodeCodeItem, Placement placement) {
64          List<String> actualCodeList = nodeCodeItem.getCode();
65          String currentName = nodeCodeItem.getName();
66          String currentType = nodeCodeItem.getType();
67          String route = nodeCodeItem.getRoute();
68          codeAnalizer = new CodeAnalizer(actualCodeList, rootPath);
69          codeAnalizer.setComponentName(currentName);
70          codeAnalizer.setComponentType(currentType);
71          codeAnalizer.analize();
72          // Shapes actuales
73          IconAnnotation iconTemp = codeAnalizer.getIconPane();
74          if (iconTemp != null) {
75              // Poner eventos a los conectores
76              if (currentType.contains("connector")) {
77                  for (ShapeAnnotation shape_i : iconTemp.getShapes()) {
78                      if (shape_i == null) {
79                          continue;
80                      }
81                      shape_i.getShape().setOnMouseEntered(event -> {
82                          shape_i.getShape().setCursor(Cursor.CROSSHAIR);
83                          if (DEBUG) {
84                              System.out.println("--> Soy un conector " +
                                    currentName + "\nPosition: " + event.
                                    getX() + "," + event.getY());
85                          }
86                      });
87                      shape_i.getShape().setOnMouseExited(event -> {
88                          shape_i.getShape().setCursor(Cursor.DEFAULT);
89                      });
90
91                  }
92              }
93              //if (currentType.contains("package")) { // los shapes de
                    package van al fondo.
94              if (currentType.contains("package") || route.contains("
                    Blocks.Icons.")) { // los shapes de packagey Icons van
                    al fondo.
95                  for (ShapeAnnotation shape_i : iconTemp.getShapes()) {
96                      shape_i.setId(currentType + "::" + currentName + "::
                            " + route);/// ID de los shapes:
97                      iconAnnotation.addAt(0, shape_i);
```

```
98                  }
99              } else if (placement != null) { // hay que escalar la figura
100                 for (ShapeAnnotation shape_i : iconTemp.getShapes()) {
101                     double centerX = iconAnnotation.getCoordinateSystem
                            ().getExtent().getWidth() / 2;
102                     double centerY = iconAnnotation.getCoordinateSystem
                            ().getExtent().getHeight() / 2;
103                     double originShapeX = placement.getTransformation().
                            getOrigin().getX();
104                     double originShapeY = placement.getTransformation().
                            getOrigin().getY();
105                     // Obtener las coordenadas del centro del rectangle
106                     double leftX = Math.min(placement.getTransformation
                            ().getExtent().getStart().getX(), placement.
                            getTransformation().getExtent().getEnd().getX())
                            ;
107                     double topY = Math.max(placement.getTransformation()
                            .getExtent().getStart().getY(), placement.
                            getTransformation().getExtent().getEnd().getY())
                            ;
108                     double localX = centerX + leftX + originShapeX;
109                     double localY = centerY - topY - originShapeY;
110                     double targetWidth = placement.getTransformation().
                            getExtent().getWidth();
111                     double targetHeight = placement.getTransformation().
                            getExtent().getHeight();
112                     double widthOriginal = iconTemp.getCoordinateSystem
                            ().getExtent().getWidth();
113                     double heightOriginal = iconTemp.getCoordinateSystem
                            ().getExtent().getHeight();
114                     double scaleX = targetWidth / widthOriginal;
115                     double scaleY = targetHeight / heightOriginal;
116                     double rotation = placement.getTransformation().
                            getRotation();
117                     shape_i.getShape().getTransforms().add(new Translate
                            (localX, localY));
118                     if (shape_i.getShape() instanceof Group) { //
                            poligonos agrupados
119                         shape_i.getShape().getTransforms().add(new
                                Translate(40, 0));
120                         shape_i.getShape().getTransforms().add(new
                                Rotate(-rotation));
121                     }
122                     shape_i.getShape().getTransforms().add(new Scale(
                            scaleX, scaleY));
123                     //shape_i.setId(currentType + "::" + currentName +
                            "::" + route);
124                     //Test: poniendo el nombre del conector
125                     if(currentType.contains("connector")){
126                         shape_i.setId(currentType + "::" + connectors.
                                get(connectors.size()-1).getName() + "::" +
                                route);
127                     }else{
128                         shape_i.setId(currentType + "::" + currentName +
                                "::" + route);
129                     }
130
131
132                     iconAnnotation.addAt(iconAnnotation.getShapes().size
                            (), shape_i);
133                 }
```

```
134             } else {
135                 for (ShapeAnnotation shape_i : iconTemp.getShapes()) {
136                     shape_i.setId(currentType + "::" + currentName + "::
                            " + route);
137                     iconAnnotation.add(shape_i);
138                 }
139             }
140         }
141 
142         // Shapes obtenidos por herencia (extends)
143         Map<String, List<String>> mapDeclarations = codeAnalizer.
                getDeclarations();
144         for (String extendLine : mapDeclarations.get("extends")) {
145             String routeToFind = extendLine.replace("extends", "").
                    replace(";", "").strip();
146             routeToFind = routeToFind.replaceAll("(\\(.*\\))", "");//
                    Eliminar redefiniciones
147 
148             if (!routeToFind.contains("Modelica.")) {//No tiene ruta
                    completa, toma la misma del actual fichero
149                 String currentCodeName = nodeCodeItem.getName();
150                 String parentRoute = nodeCodeItem.getRoute();
151                 routeToFind = normalizePath(parentRoute, routeToFind,
                        currentCodeName);
152             }
153             NodeItemCode nodeExtend = getNodeByRute(rootTree,
                    routeToFind);
154             if (nodeExtend != null) {
155                 makeCompleteIconAnnotation(nodeExtend, null);
156             } else {
157                 if (DEBUG) {
158                     System.out.println("IconManager ---> No se encontro
                            la ruta del extend: node:" + nodeCodeItem.
                            getName() + "\troute:" + routeToFind);
159                 }
160             }
161 
162         }
163         // Shapes obtenidos de la composición.
164         for (String componentLine : mapDeclarations.get("variables")) {
165             String componentType = componentLine.split("\\s")[0];
166             String routeToFind = componentType;
167             String parentRoute = nodeCodeItem.getRoute();
168             String[] parentRouteNames = parentRoute.split("\\.");
169             String alternativeRoute = "Modelica.Fluid." + componentType;
                     // esto hay que mejorarlo
170             if (parentRouteNames.length > 1) {
171                 alternativeRoute = parentRouteNames[0] + "." +
                        parentRouteNames[1] + "." + componentType;
172             }
173             if (!routeToFind.contains("Modelica.")) {//No tiene ruta
                    completa, toma la misma del actual fichero
174                 String currentCodeName = nodeCodeItem.getName();
175                 routeToFind = normalizePath(parentRoute, routeToFind,
                        currentCodeName);
176             }
177             NodeItemCode nodeComponent = getNodeByRute(rootTree,
                    routeToFind);
178             if (nodeComponent == null) {// try rootPath
179                 nodeComponent = getNodeByRute(rootTree, alternativeRoute
                        );
```

```
180                 routeToFind = alternativeRoute; //update route;
181             }
182 
183             if (nodeComponent != null) {
184                 if (nodeComponent.getType().contains("connector")) {
185                     // hay que cambiar las dimensiones del icono usando
                            el Placement
186                     Placement placementComponent = codeAnalizer.
                            getPlacement(componentLine);
187                     ModelicaConnector connector = codeAnalizer.
                            getConnectorComponentByLine(componentLine);
188                     connector.setType(routeToFind);
189                     connectors.add(connector);
190                     makeCompleteIconAnnotation(nodeComponent,
                            placementComponent);
191                 }
192             }
193 
194         }
195         return iconAnnotation;
196     }
197 
198     private String normalizePath(String parentPath, String relativePath,
             String currentComponentName) {
199         String routeToFind = parentPath.replace(currentComponentName,
                relativePath);
200         String[] routesName = routeToFind.split("\\.");
201         String absolutePath = "";
202         String previousName = "";
203         for (String name : routesName) {//eliminar repeticiones de
                nombres en la ruta
204             if (!name.equals(previousName)) {
205                 absolutePath += name + ".";
206             }
207             previousName = name;
208         }
209         absolutePath = absolutePath.substring(0, absolutePath.length() -
                1);//eliminar el último punto
210         return absolutePath;
211     }
212 
213     private NodeItemCode getNodeByRute(TreeItem<NodeItemCode> root,
             String route) {
214         String routeStr = root.getValue().getRoute();
215         if (root.getValue().getRoute().contains(route)) {
216             return root.getValue();
217         }
218         for (TreeItem<NodeItemCode> child : root.getChildren()) {
219             NodeItemCode foundNode = getNodeByRute(child, route);
220             if (foundNode != null) {
221                 return foundNode; // Se ha encontrado el nodo en un hijo
222             }
223         }
224         return null;
225     }
226 }
```

Código B.8: Implementación del manejador de iconos.

Código de la clase que compone a un icono a partir de primitivas: IconAnnotation.java

```
1  package com.fluideditor.model.icon;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import javafx.scene.Group;
6  import javafx.scene.Node;
7  import javafx.scene.layout.Pane;
8  import javafx.scene.shape.Shape;
9
10 /**
11  * Icono compuesto de un conjunto de objetos primitivos.
12  *
13  * @author Jackson F. Reyes Bermeo
14  */
15 public class IconAnnotation extends Pane {
16
17     private Placement placement;
18     private final List<ShapeAnnotation> shapes;
19     private CoordinateSystem coordinateSystem;
20
21     public IconAnnotation() {
22         shapes = new ArrayList<>();
23         coordinateSystem = new CoordinateSystem();
24         coordinateSystem.setExent(new Extent(-100, -100, 100, 100)); //
               defoult see modelica doc
25     }
26
27     public Placement getPlacement() {
28         return placement;
29     }
30
31     public void setPlacement(Placement placement) {
32         this.placement = placement;
33     }
34
35     /**
36      * A adir un elemento de tipo ShapeAnnotation
37      *
38      * @param shape Objeto hijo de ShapeAnnotation.
39      */
40     public void add(ShapeAnnotation shape) {
41         shape.setCoordinateSystem(coordinateSystem); //comparte las
               coordenadas con los shapes
42         shapes.add(shape);
43     }
44
45     public void addAt(int index, ShapeAnnotation shape) {
46         shape.setCoordinateSystem(coordinateSystem); //comparte las
               coordenadas con los shapes
47         shapes.add(index, shape);
48     }
49
50     /**
51      * Devuelve el Icono compuesto de Shapes primitivos encapsulado en
            un Pane.
52      *
53      * @return Objeto de tipo Pane.
```

```
54          *
55          */
56         public Node getIcon() {
57             this.getChildren().clear();//eliminar todos los hijos previos
58             this.setPrefWidth(coordinateSystem.getExtent().getWidth());
59             this.setPrefHeight(coordinateSystem.getExtent().getHeight());
60             this.setStyle("-fx-background-color: transparent;"); //color de
                   fondo por defecto
61             // Recorremos todos los shapes(primitivas) para formar el icono
                   y lo a adimos al panel
62             for (ShapeAnnotation shape : shapes) {
63                 Node tempShape = shape.getShape();
64                 this.getChildren().add(tempShape);
65             }
66             return this;
67         }
68 
69         /**
70          * Devuelve el sistema de coordenadas del Icono (shape).
71          *
72          * @return CoordinateSystem.
73          *
74          */
75         public CoordinateSystem getCoordinateSystem() {
76             return coordinateSystem;
77         }
78 
79         public void setCoordinateSystem(CoordinateSystem coordinateSystem) {
80             this.coordinateSystem = coordinateSystem;
81 
82         }
83 
84         /**
85          * Devuelve la lista de cada uno de los componentes primitivas del
                 icono.
86          *
87          * @return Lista de primitivas gráficas.
88          *
89          */
90         public List<ShapeAnnotation> getShapes() {
91             return shapes;
92         }
93 
94         /**
95          * A adir una lista de primitivas gráficas.
96          *
97          * @param shapes Lista de tipo ShapeAnnotation.
98          */
99         public void addAll(List<ShapeAnnotation> shapes) {
100            for (ShapeAnnotation shape : shapes) {
101                this.add(shape);
102            }
103 
104        }
105 
106        /**
107         * Obtener un ShapeAnnotation a partir de un elemento Shape que lo
                compone.
108         *
109         * @param shape Shape especifico del ShapeAnnotation.
110         * @return ShapeAnnotation que contenga el @param shape.
```

```
111      */
112     public ShapeAnnotation getShapeAnnotationByShape(Shape shape) {
113         for (ShapeAnnotation shapeAnnotation : shapes) {
114             if (shapeAnnotation == null) {
115                 continue;
116             }
117             if (shapeAnnotation instanceof TextAnnotation) {
118                 continue; // los Text no tienen shapes, tienen stackpane
119             }
120 
121             if (shapeAnnotation.getShape()  instanceof Group) {
122                 Group tempGroup = (Group)shapeAnnotation.getShape();
123                 for(Node node:tempGroup.getChildren()){
124                     if(node == shape){
125                         return shapeAnnotation;
126                     }
127                 }
128             }
129 
130             if (shapeAnnotation.getShape() == shape) {
131                 return shapeAnnotation;
132             }
133         }
134         return null;//no encontrado
135     }
136 
137     //test
138     public boolean isConnector(Node shape){
139         for (ShapeAnnotation shapeAnnotation : shapes) {
140             if (shapeAnnotation == null) {
141                 continue;
142             }
143             if (shapeAnnotation instanceof TextAnnotation) {
144                 continue; // los Text no tienen shapes, tienen stackpane
145             }
146 
147             if (shapeAnnotation.getShape()  instanceof Group) {
148                 Group tempGroup = (Group)shapeAnnotation.getShape();
149                 for(Node node:tempGroup.getChildren()){
150                     if(node == shape){
151                         return shapeAnnotation.getId().contains("
                                connector");
152                     }
153                 }
154             }
155             if (shapeAnnotation.getShape() == shape) {
156                 return shapeAnnotation.getId().contains("connector");
157             }
158         }
159         return false;//no encontrado
160     }
161 }
```

Código B.9: Implementación de la clase que compone a un icono a partir de primitivas.

Código de la clase abstracta padre los gráficos primitivos: ShapeAnnotation.java

```
1  package com.fluideditor.model.icon;
2
3  import javafx.geometry.Point2D;
4  import javafx.scene.Node;
5
6  /**
7   * Clase Padre de todas las primitivas gráficas.
8   *
9   * @author Jackson F. Reyes Bermeo
10  */
11 public abstract class ShapeAnnotation {
12
13     protected boolean visible;
14     protected Point2D origin;
15     protected double rotation;
16     protected CoordinateSystem coordinateSystem;
17     protected String id;
18
19     public ShapeAnnotation() {
20         visible = true; //default
21         origin = new Point2D(0, 0); // default
22         rotation = 0; //default
23         coordinateSystem = new CoordinateSystem(new Extent(-100, -100,
               100, 100)); // default coordinate
24     }
25
26     public String getId() {
27         return id;
28     }
29
30     public void setId(String id) {
31         this.id = id;
32     }
33
34     public boolean getVisible() {
35         return this.visible;
36     }
37
38     public void setVisible(boolean visible) {
39         this.visible = visible;
40     }
41
42     public Point2D getOrigin() {
43         return origin;
44     }
45
46     public void setOrigin(Point2D origin) {
47         this.origin = origin;
48     }
49
50     public void setRotation(double rotation) {
51         this.rotation = rotation;
52     }
53
54     public double getRotation() {
55         return rotation;
56     }
57
58     /**
```

```
59      * Obtener un gráfico primitivo (Shape): Rectangulo, Ellipse,
            Polygon, Line,
60      * Text.
61      * @return Gráfico primitivo (Shape) de las clases hijas.
62      */
63     public abstract Node getShape();
64
65     public abstract Extent getExtent();
66
67     public void setCoordinateSystem(CoordinateSystem coordinateSystem) {
68         this.coordinateSystem = coordinateSystem;
69     }
70
71     public CoordinateSystem getCoordinateSystem() {
72         return coordinateSystem;
73     }
74 }
```

Código B.10: Implementación de la clase abstracta padre de los gráficos primitivos.

Código de la clase que representa un Rectángulo: RectangleAnnotation.java

```
1  package com.fluideditor.model.icon;
2
3  import javafx.scene.paint.Color;
4  import javafx.scene.shape.Rectangle;
5
6  /**
7   * Representa al rectangulo de Modelica.
8   *
9   * @author Jackson F. Reyes Bermeo
10  */
11 public class RectangleAnnotation extends ShapeAnnotation {
12
13     private final Rectangle rectangle;
14     private Extent extent;
15     private FilledShape filledShape;
16     private double radius;
17
18     public RectangleAnnotation() {
19         rectangle = new Rectangle();
20     }
21
22     public RectangleAnnotation(Extent exent) {
23         rectangle = new Rectangle();
24         this.extent = exent;
25     }
26
27     @Override
28     public Extent getExtent() {
29         return extent;
30     }
31
32     public void setExtent(Extent extent) {
33         this.extent = extent;
34     }
35
36     public FilledShape getFilledShape() {
37         return filledShape;
```

```
38      }
39
40      public void setFilledShape(FilledShape filledShape) {
41          this.filledShape = filledShape;
42      }
43
44      /**
45       * Configura los parámetros del rectangulo.
46       */
47      private void configure() {
48          if (extent != null) {
49              rectangle.setHeight(extent.getHeight());
50              rectangle.setWidth(extent.getWidth());
51              // origen
52              double xOrigin = origin.getX();
53              double yOrigin = origin.getY();
54              // Obtener las coordenadas del centro del Pane
55              double centerX = coordinateSystem.getExtent().getWidth() /
                    2;
56              double centerY = coordinateSystem.getExtent().getHeight() /
                    2;
57              // Obtener las coordenadas de los extremos del rectangulo
58              double leftX = Math.min(extent.getStart().getX(), extent.
                    getEnd().getX());
59              double topY = Math.max(extent.getStart().getY(), extent.
                    getEnd().getY());
60              // Desplazamiento desde el origen
61              double offsetX = centerX + leftX + xOrigin;
62              double offsetY = centerY - topY - yOrigin;
63              rectangle.setX(offsetX);
64              rectangle.setY(offsetY);
65          }
66          if (filledShape != null) {
67              rectangle.setFill(this.filledShape.getFillPaint());
68          } else {//color por defecto
69              rectangle.setFill(Color.TRANSPARENT);
70          }
71          if (filledShape.getLineColor() != null) {
72              rectangle.setStroke(filledShape.getLineColor());
73          }
74          if (filledShape.getLineThickness() > 0) {
75              rectangle.setStrokeWidth(this.filledShape.getLineThickness()
                    );
76              //rectangle.setStroke(filledShape.getLineThickness());
77          }
78          if (filledShape.getLinePattern() != null) {
79              rectangle.getStrokeDashArray().addAll(filledShape.
                    getLinePattern());
80          } else {
81              rectangle.setStrokeWidth(0);
82          }
83          rectangle.setRotate(-rotation);
84          rectangle.setArcHeight(radius);
85          rectangle.setArcWidth(radius);
86      }
87
88      public double getRadius() {
89          return radius;
90      }
91
92      public void setRadius(double radius) {
```

```
93          this.radius = radius;
94      }
95
96      /**
97       * Obtener el gráfico primitivo.
98       * @return Rectangle.
99       */
100     @Override
101     public Rectangle getShape() {
102         configure();
103         return rectangle;
104     }
105 }
```

Código B.11: Implementación de la clase que representa el gráfico primitivo de un Rectángulo.

Código de la clase que representa un Polígono: PolygonAnnotation.java

```
1  package com.fluideditor.model.icon;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import javafx.geometry.Point2D;
6  import javafx.scene.Group;
7  import javafx.scene.Node;
8  import javafx.scene.layout.StackPane;
9  import javafx.scene.paint.Color;
10 import javafx.scene.shape.CubicCurveTo;
11 import javafx.scene.shape.MoveTo;
12 import javafx.scene.shape.Path;
13 import javafx.scene.shape.PathElement;
14 import javafx.scene.shape.Polygon;
15 import javafx.scene.shape.QuadCurveTo;
16 import javafx.scene.shape.Shape;
17 import javafx.scene.transform.Rotate;
18 import javafx.scene.transform.Translate;
19
20 /**
21  * Representa al Shape primitivo del Polygono.
22  *
23  * @author Jackson F. Reyes Bermeo
24  */
25 public class PolygonAnnotation extends ShapeAnnotation {
26
27     private Shape polygon;
28     private final Group polygonGroup;
29     private final StackPane stackPolygon;
30     private FilledShape filledShape;
31     private List<Point2D> xyPoints;
32     //private boolean smooth = Smooth.None;
33     private boolean smooth = false;
34     private Color color = Color.BLACK;
35     private LinePattern pattern = LinePattern.Solid;
36     private double thickness = 1.2;
37     //private List<Arrowarrow> arrowArrow = {Arrow.None,Arrow.None};
38     private double arrowSize = 3;
39
```

```
40      public PolygonAnnotation() {
41          xyPoints = new ArrayList<>();
42          filledShape = new FilledShape();
43          stackPolygon = new StackPane();
44          polygonGroup = new Group();
45      }
46
47      public FilledShape getFilledShape() {
48          return filledShape;
49      }
50
51      public void setFilledShape(FilledShape filledShape) {
52          this.filledShape = filledShape;
53      }
54
55      public List<Point2D> getPoints() {
56          return xyPoints;
57      }
58
59      public void setPoints(List<Point2D> points) {
60          this.xyPoints = points;
61      }
62
63      public Color getColor() {
64          return color;
65      }
66
67      public void setColor(Color color) {
68          this.color = color;
69      }
70
71      public LinePattern getPattern() {
72          return pattern;
73      }
74
75      public void setPattern(LinePattern pattern) {
76          this.pattern = pattern;
77      }
78
79      public double getThickness() {
80          return thickness;
81      }
82
83      public void setThickness(double thickness) {
84          this.thickness = thickness;
85      }
86
87      public double getArrowSize() {
88          return arrowSize;
89      }
90
91      public void setArrowSize(double arrowSize) {
92          this.arrowSize = arrowSize;
93      }
94
95      @Override
96      public Node getShape() {
97          configure();
98          stackPolygon.getChildren().add(polygon);
99          polygonGroup.getChildren().add(polygon);
100         return polygonGroup;
```

```
101     }
102
103     public boolean isSmooth() {
104         return smooth;
105     }
106
107     public void setSmooth(boolean smooth) {
108         this.smooth = smooth;
109     }
110
111     /**
112      * Obtener una lista de coordenadas de una lista de puntos.
113      *
114      *
115      * @param pointList Lista de puntos Point2D.
116      * @return Lista de coordenadas.
117      */
118     private List<Double> getPointsWithLocalCoordinates(List<Point2D>
             pointList) {
119         // origen
120         double xOrigin = origin.getX();
121         double yOrigin = origin.getY();
122         // Obtener las coordenadas del centro del Pane
123         double centerX = coordinateSystem.getExtent().getWidth() / 2;
124         double centerY = coordinateSystem.getExtent().getHeight() / 2;
125         // Desplazamiento desde el origen
126         double offsetX = centerX + xOrigin;
127         double offsetY = centerY - yOrigin;
128         List<Double> coordinates = new ArrayList<>();
129         for (int i = 0; i < pointList.size(); i++) {
130             coordinates.add(pointList.get(i).getX() + offsetX);
131             coordinates.add(-pointList.get(i).getY() + offsetY);
132         }
133         return coordinates;
134     }
135
136     /**
137      * Interpolar un lista de puntos para obtener una forma de Bezier.
               Ver
138      * Funciones de Bezier
139      *
140      * @param points Puntos de los que se interpolan.
141      * @return Puntos interpolados.
142      */
143     private List<Double> interpolatePoints(List<Double> points) {
144         // Lista para almacenar los puntos interpolados
145         List<Double> interpolatedPoints = new ArrayList<>();
146         // Obtener la cantidad de puntos originales
147         int originalSize = points.size() / 2;
148         // Asegurarse de que haya suficientes puntos para interpolar
149         if (originalSize < 2) {
150             throw new IllegalArgumentException("Se necesitan al menos 2
                    puntos para interpolar.");
151         }
152         // Calcular la cantidad de puntos interpolados que se deben
                generar entre los puntos originales
153         int interpolatedSize = (originalSize - 1) * 100; // Ajusta la
                precisión según tus necesidades
154         // Interpolar los puntos
155         for (int i = 0; i < originalSize - 1; i++) {
156             double startX = points.get(i * 2);
```

```
157             double startY = points.get(i * 2 + 1);
158             double endX = points.get((i + 1) * 2);
159             double endY = points.get((i + 1) * 2 + 1);
160 
161             for (int j = 0; j < interpolatedSize; j++) {
162                 double t = (double) j / interpolatedSize;
163                 double x = (1 - t) * startX + t * endX;
164                 double y = (1 - t) * startY + t * endY;
165                 interpolatedPoints.add(x);
166                 interpolatedPoints.add(y);
167             }
168         }
169         // Agregar el último punto original
170         interpolatedPoints.add(points.get((originalSize - 1) * 2));
171         interpolatedPoints.add(points.get((originalSize - 1) * 2 + 1));
172         return interpolatedPoints;
173     }
174 
175     private Polygon convertFromPathToPolygon(Path bezierPath) {
176         // Obtener los elementos de la curva Bezier
177         List<PathElement> pathElements = bezierPath.getElements();
178         // Crear una lista para almacenar los puntos del polígono
179         List<Double> points = new ArrayList<>();
180         // Definir la precisión de la aproximación (número de segmentos)
181         int precision = 20;
182         // Recorrer los elementos de la curva Bezier
183         for (int i = 0; i < pathElements.size(); i++) {
184             PathElement element = pathElements.get(i);
185             if (element instanceof MoveTo) {
186                 MoveTo moveTo = (MoveTo) element;
187                 points.add(moveTo.getX());
188                 points.add(moveTo.getY());
189             } else if (element instanceof CubicCurveTo) {
190                 CubicCurveTo curveTo = (CubicCurveTo) element;
191                 // Calcular los puntos aproximados en la curva Bezier
192                 for (int j = 1; j <= precision; j++) {
193                     double t = (double) j / precision;
194                     double x = Math.pow(1 - t, 3) * curveTo.getControlX1
                            ()
195                             + 3 * Math.pow(1 - t, 2) * t * curveTo.
                                getControlX2()
196                             + 3 * (1 - t) * Math.pow(t, 2) * curveTo.
                                getX()
197                             + Math.pow(t, 3) * curveTo.getX();
198                     double y = Math.pow(1 - t, 3) * curveTo.getControlY1
                            ()
199                             + 3 * Math.pow(1 - t, 2) * t * curveTo.
                                getControlY2()
200                             + 3 * (1 - t) * Math.pow(t, 2) * curveTo.
                                getY()
201                             + Math.pow(t, 3) * curveTo.getY();
202                     points.add(x);
203                     points.add(y);
204                 }
205             }
206         }
207         // Crear el polígono a partir de los puntos
208         Polygon polygon = new Polygon();
209         polygon.getPoints().addAll(points);
210         // Establecer el estilo del polígono
211         polygon.setStroke(bezierPath.getStroke());
```

```
212         polygon.setStrokeWidth(bezierPath.getStrokeWidth());
213         return polygon;
214     }
215 
216     private void configure() {
217         if (smooth) {//bezier
218             List<Double> xyPoints = new ArrayList<>();
219             xyPoints.addAll(getPointsWithLocalCoordinates(this.xyPoints)
                    );
220             // Crear un objeto Path para construir el polígono suavizado
221             Path bezier = new Path();
222             // Crear el elemento MoveTo con el primer punto
223             MoveTo moveTo = new MoveTo(xyPoints.get(0), xyPoints.get(1))
                    ;
224             bezier.getElements().add(moveTo);
225             int pointCount = xyPoints.size();
226             if (pointCount >= 2 && pointCount % 2 == 0) {
227                 for (int i = 2; i < pointCount - 2; i += 2) {
228                     double endX = (xyPoints.get(i) + xyPoints.get(i + 2)
                            ) / 2;
229                     double endY = (xyPoints.get(i + 1) + xyPoints.get(i
                            + 3)) / 2;
230                     double controlX = xyPoints.get(i);
231                     double controlY = xyPoints.get(i + 1);
232                     QuadCurveTo curveTo = new QuadCurveTo(controlX,
                            controlY, endX, endY);
233                     bezier.getElements().add(curveTo);
234                 }
235             } else {
236                 System.out.println("La cantidad de puntos no es válida
                        para crear curvas de Bezier.");
237             }
238             polygon = new Path();// Convertir la curva Bezier a un polí
                    gono
239             polygon = bezier;
240         } else {
241             Polygon tempPol = new Polygon();
242             tempPol.getPoints().addAll(getPointsWithLocalCoordinates(
                    xyPoints));
243             polygon = tempPol;
244         }
245         if (filledShape.getFillColor() != null) {
246             polygon.setFill(this.filledShape.getFillPaint());
247         }
248         if (filledShape.getLineColor() != null) {
249             polygon.setStroke(filledShape.getLineColor());
250         }
251         if (filledShape.getLineThickness() > 0) {
252             polygon.setStrokeWidth(this.filledShape.getLineThickness());
253         }
254         if (filledShape.getLinePattern() != null) {
255             polygon.getStrokeDashArray().addAll(filledShape.
                    getLinePattern());
256         } else {
257             polygon.setStrokeWidth(0);
258         }
259         // Definir el punto de pivote personalizado para la operaciones
                de rotación.
260         double centerX = coordinateSystem.getExtent().getWidth() / 2;
261         double centerY = coordinateSystem.getExtent().getHeight() / 2;
262         double pivotX = centerX + origin.getX();
```

```
263         double pivotY = centerY - origin.getY();
264         // Calcular el desplazamiento para el punto de pivote
265         double offsetX = pivotX - polygon.getBoundsInLocal().getCenterX
                ();
266         double offsetY = pivotY - polygon.getBoundsInLocal().getCenterY
                ();
267         // Aplicar el desplazamiento al punto de pivote: el origen
268         Translate translate = new Translate(offsetX, offsetY);
269         // Aplicar la rotación al Polygon
270         Rotate rotate = new Rotate(-rotation, pivotX, pivotY);
271         // Crear un Group y agregar el polígono
272         polygonGroup.getChildren().add(polygon);
273         polygonGroup.getTransforms().add(rotate);
274     }
275
276     @Override
277     public Extent getExtent() {
278         throw new UnsupportedOperationException("Los poligonos no tienen
                Extent");
279     }
280
281 }
```

Código B.12: Implementación de la clase que representa el gráfico primitivo de un Polígono.

Código de la clase que representa una Elipse: EllipseAnnotation.java

```
1  package com.fluideditor.model.icon;
2
3  import javafx.scene.shape.Arc;
4  import javafx.scene.shape.ArcType;
5
6  /**
7   * Primitiva gráfica correspondiente a una elipse.
8   * @author Jackson F. Reyes Bermeo
9   */
10 public class EllipseAnnotation extends ShapeAnnotation {
11
12     private final Arc ellipse; //para poder representar la elipse en
            java
13     private FilledShape filledShape;
14     private Extent extent;
15     private double startAngle = 0;
16     private double endAngle = 360;
17
18     public EllipseAnnotation() {
19         ellipse = new Arc();
20     }
21
22     public void setExtent(Extent extent) {
23         this.extent = extent;
24     }
25
26     public double getStartAngle() {
27         return startAngle;
28     }
29
```

```
30      public void setStartAngle(double startAngle) {
31          this.startAngle = startAngle;
32      }
33
34      public double getEndAngle() {
35          return endAngle;
36      }
37
38      public void setEndAngle(double endAngle) {
39          this.endAngle = endAngle;
40      }
41
42      public void setFilledShape(FilledShape filledShape) {
43          this.filledShape = filledShape;
44      }
45
46      @Override
47      public Arc getShape() {
48          try {
49              configure();
50          } catch (Exception e) {
51              System.out.println(">>> Error al configurar la Ellipse : " +
                    this.toString());
52              return null;
53          }
54          return ellipse;
55      }
56
57      @Override
58      public Extent getExtent() {
59          return this.extent;
60      }
61
62      private void configure() {
63          if (extent != null) {//Arc arc = new Arc(CENTER_X, CENTER_Y,
                RADIUS_X, RADIUS_Y, START_ANGLE, ANGLE_LENGTH);
64              ellipse.setRadiusX(extent.getWidth() / 2);
65              ellipse.setRadiusY(extent.getHeight() / 2);
66              double xOrigin_local = origin.getX();
67              double yOrigin_local = origin.getY();
68              double xCenter_global = (coordinateSystem.getExtent().
                    getWidth() / 2);
69              double yCenter_global = (coordinateSystem.getExtent().
                    getHeight() / 2);
70              double xCenter = xCenter_global + extent.getXMed() +
                    xOrigin_local;
71              double yCenter = yCenter_global - extent.getYMed() -
                    yOrigin_local;
72              ellipse.setCenterX(xCenter);
73              ellipse.setCenterY(yCenter);
74          }
75          if (filledShape.getFillColor() != null) {
76              ellipse.setFill(this.filledShape.getFillPaint());
77          }
78          if (filledShape.getLineColor() != null) {
79              ellipse.setStroke(filledShape.getLineColor());
80          }
81
82          if (filledShape.getLineThickness() > 0) {
83              ellipse.setStrokeWidth(this.filledShape.getLineThickness());
84          }
```

```
85          if (filledShape.getLinePattern() != null) {
86              ellipse.getStrokeDashArray().addAll(filledShape.
                    getLinePattern());
87          } else {
88              ellipse.setStrokeWidth(0);
89          }
90
91          double initAngle = extent.getAngleCorrection() + startAngle;
92          double lengthAngle = Math.abs(startAngle - endAngle);
93          if (extent.isAntiHorary()) {
94              ellipse.setStartAngle(initAngle);
95              ellipse.setLength(lengthAngle);
96          } else {
97              ellipse.setStartAngle(-initAngle);
98              ellipse.setLength(-lengthAngle);
99          }
100         ellipse.setRotate(-rotation);
101         ellipse.setType(ArcType.ROUND);
102     }
103 }
```

Código B.13: Implementación de la clase que representa el gráfico primitivo de una Elipse.

Código de la clase que representa una Línea: LineAnnotation.java

```
1  package com.fluideditor.model.icon;
2
3  import java.text.DecimalFormat;
4  import java.text.DecimalFormatSymbols;
5  import java.util.ArrayList;
6  import java.util.List;
7  import java.util.Locale;
8  import javafx.geometry.Point2D;
9  import javafx.scene.paint.Color;
10 import javafx.scene.shape.Polyline;
11 import javafx.scene.shape.StrokeType;
12
13 /**
14  * Representa una linea primitiva.
15  *
16  * @author Jackson F. Reyes Bermeo
17  */
18 public class LineAnnotation extends ShapeAnnotation {
19
20     private List<Point2D> points;
21     private Color color = Color.rgb(0, 127, 255);
22     private LinePattern pattern = LinePattern.Solid;
23     private double thickness = 0.75;
24     //private List<Arrowarrow> arrowArrow = {Arrow.None,Arrow.None};
25     private double arrowSize = 3;
26     //private boolean smooth = Smooth.None;
27     private List<Double> linePattern; // Patrón de línea punteada
28     private final Polyline polyLine;
29
30     public LineAnnotation() {
31         points = new ArrayList<>();
32         linePattern = new ArrayList<>();
33         polyLine = new Polyline();
34     }
```

```
35
36      public List<Point2D> getPoints() {
37          return points;
38      }
39
40      public void addPoint(Point2D point) {
41          this.points.add(point);
42      }
43
44      public void setPoints(List<Point2D> points) {
45          this.points = points;
46      }
47
48      public double getArrowSize() {
49          return arrowSize;
50      }
51
52      public void setArrowSize(double arrowSize) {
53          this.arrowSize = arrowSize;
54      }
55
56      public double getThickness() {
57          return thickness;
58      }
59
60      public void setThickness(double thickness) {
61          this.thickness = thickness;
62      }
63
64      public void setColor(Color color) {
65          this.color = color;
66      }
67
68      public LinePattern getPattern() {
69          return pattern;
70      }
71
72      public void setPattern(String pattern) {
73          this.pattern = LinePattern.valueOf(pattern);
74          configureLinePattern();
75      }
76
77      public List<Double> getLinePattern() {
78          return linePattern;
79      }
80
81      /**
82       * Configura el tipo de linea.
83       */
84      private void configureLinePattern() {
85          switch (pattern) { //None , Solid , Dash , Dot , DashDot ,
                DashDotDot
86              case None:
87                  this.linePattern = null;
88                  break;
89              case Solid:
90                  this.linePattern.add(1.0);
91                  break;
92              case Dash:
93                  this.linePattern.add(10.0);
94                  this.linePattern.add(10.0);
```

```
95                  break;
96              case Dot:
97                  this.linePattern.add(1.0);
98                  this.linePattern.add(10.0);
99                  break;
100             case DashDot:
101                 this.linePattern.add(15.0);
102                 this.linePattern.add(10.0);
103                 this.linePattern.add(3.0);
104                 this.linePattern.add(10.0);
105                 break;
106             case DashDotDot:
107                 this.linePattern.add(15.0);
108                 this.linePattern.add(10.0);
109                 this.linePattern.add(3.0);
110                 this.linePattern.add(10.0);
111                 this.linePattern.add(3.0);
112                 this.linePattern.add(10.0);
113                 break;
114         }
115     }
116
117     /**
118      * Método para obtener las coordenadas a partir de una lista de
            puntos.
119      *
120      *
121      * @param pointList Lista de puntos Point2D
122      * @return Lista de puntos Double.
123      */
124     private List<Double> getCoordinates(List<Point2D> pointList) {
125         // origen
126         double xOrigin = origin.getX();
127         double yOrigin = origin.getY();
128         // Obtener las coordenadas del centro del Pane
129         double centerX = coordinateSystem.getExtent().getWidth() / 2;
130         double centerY = coordinateSystem.getExtent().getHeight() / 2;
131         // Desplazamiento desde el origen
132         double offsetX = centerX + xOrigin;
133         double offsetY = centerY - yOrigin;
134         List<Double> coordinates = new ArrayList<>();
135         for (int i = 0; i < pointList.size(); i++) {
136             coordinates.add(pointList.get(i).getX() + offsetX);
137             coordinates.add(-pointList.get(i).getY() + offsetY);
138         }
139         return coordinates;
140     }
141
142     private void configure() {
143         polyLine.getPoints().clear(); //reiniciar la lista
144         polyLine.getPoints().addAll(getCoordinates(points));
145         polyLine.setStroke(color);
146         polyLine.setStrokeWidth(thickness);
147         if (getPattern() != null) {
148             polyLine.getStrokeDashArray().addAll(getLinePattern());
149         } else {
150             polyLine.setStrokeWidth(0);
151         }
152         polyLine.setRotate(-rotation);
153         polyLine.setStrokeType(StrokeType.CENTERED);
154     }
```

```
155
156     @Override
157     public Polyline getShape() {
158         configure();
159         return polyLine;
160     }
161
162     @Override
163     public Extent getExtent() {
164         throw new UnsupportedOperationException("Polyline no tiene
                Extent");
165     }
166
167     @Override
168     public String toString() {
169         return getCodeString();
170     }
171
172     public String getCodeString() {
173         String textCode = "";
174         if (points != null) {
175             textCode += "Line(";
176             textCode += "points={";
177             for (Point2D point : points) {
178                 // Crear un objeto DecimalFormat para redondear el nú
                        mero a dos decimales
179                 DecimalFormat decimalFormat = new DecimalFormat("#.##",
                        new DecimalFormatSymbols(Locale.US));
180                 String xPoint = decimalFormat.format(point.getX());
181                 String yPoint = decimalFormat.format(point.getY());
182                 textCode += "{" + xPoint + "," + yPoint + "},";
183             }
184             textCode += "},";
185             if (pattern != null) {
186                 textCode += "pattern=LinePattern." + pattern.toString()
                        + ",";
187             }
188             if (color != null) {
189                 int redColor = (int) (color.getRed() * 255);
190                 int greenColor = (int) (color.getRed() * 255);
191                 int blueColor = (int) (color.getRed() * 255);
192                 textCode += "color={" + redColor + "," + greenColor + ",
                        " + blueColor + "}";
193             }
194             textCode += ")";
195         }
196         textCode = textCode.replace(",)", ")");
197         textCode = textCode.replace("},}", "}}");
198         return textCode;
199     }
200 }
```

Código B.14: Implementación de la clase que representa el gráfico primitivo de una Línea.

Código de la clase que representa un Texto: TextAnnotation.java

```
1  package com.fluideditor.model.icon;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import javafx.geometry.Pos;
6  import javafx.scene.Node;
7  import javafx.scene.layout.StackPane;
8  import javafx.scene.paint.Color;
9  import javafx.scene.text.Font;
10 import javafx.scene.text.FontPosture;
11 import javafx.scene.text.FontWeight;
12 import javafx.scene.text.Text;
13
14 /**
15  * Representa a los textos de los iconos.
16  * @author Jackson F. Reyes Bermeo
17  */
18 public class TextAnnotation extends ShapeAnnotation {
19
20     private FilledShape filledShape;
21     private Extent extent;
22     private String textString;
23     private double fontSize = 0;//d
24     private String fontName;
25     private List<String> textStyle;
26     private Color textColor = Color.BLACK;
27     private String horizontalAlignament = "TextAlignament.Center";
28     private final Text text;
29     private final StackPane textContent;
30
31     public TextAnnotation() {
32         filledShape = new FilledShape();
33         textStyle = new ArrayList<>();
34         this.text = new Text();
35         textContent = new StackPane();
36     }
37
38     public void setExtent(Extent extent) {
39         this.extent = extent;
40     }
41
42     public void setHorizontalAlignament(String horizontalAlignament) {
43         this.horizontalAlignament = horizontalAlignament;
44     }
45
46     public FilledShape getFilledShape() {
47         return filledShape;
48     }
49
50     public void setFilledShape(FilledShape filledShape) {
51         this.filledShape = filledShape;
52     }
53
54     public String getTextString() {
55         return textString;
56     }
57
58     public void setTextString(String textString) {
59         this.textString = textString.replaceAll("\"", "");
```

```
60      }
61
62      public double getFontSize() {
63          return fontSize;
64      }
65
66      public void setFontSize(double fontSize) {
67          this.fontSize = fontSize;
68      }
69
70      public String getFontName() {
71          return fontName;
72      }
73
74      public void setFontName(String fontName) {
75          this.fontName = fontName;
76      }
77
78      // Devuelve una lista de estilos del texto
79      public List<String> getTextStyle() {
80          return textStyle;
81      }
82
83      public void setTextStyle(List<String> textStyle) {
84          this.textStyle = textStyle;
85      }
86
87      public Color getTextColor() {
88          return textColor;
89      }
90
91      public void setTextColor(Color textColor) {
92          this.textColor = textColor;
93      }
94
95      @Override
96      public Node getShape() {
97          configure();
98          return textContent;
99      }
100
101     private void makeStyle() {
102         text.setFill(textColor);
103         FontWeight fw = FontWeight.NORMAL;//default
104         FontPosture fp = FontPosture.REGULAR;//default
105         for (String value : textStyle) {
106             if (value.contains("Bold")) {
107                 fw = FontWeight.BOLD;
108             } else if (value.contains("Italic")) {
109                 fp = FontPosture.ITALIC;
110             } else if (value.contains("UnderLine")) {
111                 text.setUnderline(true);
112             }
113         }
114         text.setFont(Font.font(fontName, fw, fp, fontSize));
115         Pos alignment;
116         switch (horizontalAlignament) {
117             case "Left":
118                 alignment = Pos.CENTER_LEFT;
119                 break;
120             case "Center":
```

```
121                 alignment = Pos.CENTER;
122                 break;
123             case "Right":
124                 alignment = Pos.CENTER_RIGHT;
125                 break;
126             default:
127                 alignment = Pos.CENTER;
128                 break;
129         }
130         textContent.setAlignment(alignment);
131         text.setStyle("-fx-background-color: red;");
132 
133     }
134 
135     private double calculateFontSizeToFitText(Text text, double
            containerWidth, double containerHeight) {
136         double fontSize = text.getFont().getSize();
137         double textWidth = 0;
138         double textHeight = 0;
139         while (textWidth < containerWidth && textHeight <
                containerHeight) {
140             fontSize++;
141             text.setFont(Font.font(text.getFont().getFamily(), fontSize)
                    );
142             textWidth = text.getLayoutBounds().getWidth();
143             textHeight = text.getLayoutBounds().getHeight();
144         }
145         return fontSize - 1;
146     }
147 
148     private void configure() {
149         text.setText(textString);
150         if (this.fontSize <= 0) {//compute fontSize;
151             this.fontSize = calculateFontSizeToFitText(text, extent.
                    getWidth(), extent.getHeight());
152         }
153         // origen
154         double xOrigin = origin.getX();
155         double yOrigin = origin.getY();
156         // Obtener las coordenadas del centro del Pane
157         double centerX = coordinateSystem.getExtent().getWidth() / 2;
158         double centerY = coordinateSystem.getExtent().getHeight() / 2;
159         // Obtener las coordenadas del centro del rectangle
160         double leftX = Math.min(extent.getStart().getX(), extent.getEnd
                ().getX());
161         double topY = Math.max(extent.getStart().getY(), extent.getEnd()
                .getY());
162         // Desplazamiento desde el origen
163         double offsetX = centerX + leftX + xOrigin;
164         double offsetY = centerY - topY - yOrigin;
165         textContent.setPrefWidth(extent.getWidth());
166         textContent.setPrefHeight(extent.getHeight());
167         textContent.setStyle("-fx-background-color: transparent;");
168         textContent.setLayoutX(offsetX);
169         textContent.setLayoutY(offsetY);
170         textContent.getChildren().add(text);
171         textContent.setRotate(rotation);
172         // Establecer propiedades del texto
173         makeStyle();
174     }
175 
```

```
176     @Override
177     public Extent getExtent() {
178         return extent;
179     }
180 }
```

Código B.15: Implementación de la clase que representa la primitiva Texto.

Código de la clase que representa un Bitmap: BitmapAnnotation.java

```
1  package com.fluideditor.model.icon;
2
3  import javafx.scene.Node;
4  import javafx.scene.image.Image;
5  import javafx.scene.image.ImageView;
6
7  /**
8   * Primitiva gráfica para un Bitmap.
9   *
10  * @author Jackson F. Reyes Bermeo
11  */
12 public class BitmapAnnotation extends ShapeAnnotation {
13
14     private Extent extent;
15     private String fileName;
16     private final ImageView imageView;
17
18     public BitmapAnnotation() {
19         imageView = new ImageView();
20     }
21
22     public String getFileName() {
23         return fileName;
24     }
25
26     public void setFileName(String fileName) {
27         this.fileName = fileName;
28     }
29
30     @Override
31     public Node getShape() {
32         configure();
33         return imageView;
34     }
35
36     private void configure() {
37         Image bitmapImage = null;
38         if (fileName != null) {
39             bitmapImage = new Image("file:" + fileName);
40         }
41         if (bitmapImage != null) {
42             imageView.setImage(bitmapImage);
43             imageView.setFitHeight(extent.getHeight());
44             imageView.setFitWidth(extent.getWidth());
45             // origen
46             double xOrigin = origin.getX();
47             double yOrigin = origin.getY();
48             // Obtener las coordenadas del centro del Pane
```

```
49              double centerX = coordinateSystem.getExtent().getWidth() /
                    2;
50              double centerY = coordinateSystem.getExtent().getHeight() /
                    2;
51              // Obtener las coordenadas del centro del rectangle
52              double leftX = Math.min(extent.getStart().getX(), extent.
                    getEnd().getX());
53              double topY = Math.max(extent.getStart().getY(), extent.
                    getEnd().getY());
54              // Desplazamiento desde el origen
55              double offsetX = centerX + leftX + xOrigin;
56              double offsetY = centerY - topY - yOrigin;
57              imageView.setX(offsetX);
58              imageView.setY(offsetY);
59              imageView.setRotate(-rotation);
60          }
61      }
62
63      @Override
64      public Extent getExtent() {
65          return extent;
66      }
67
68      public void setExtent(Extent extent) {
69          this.extent = extent;
70      }
71  }
```

Código B.16: Implementación de la clase que representa una primitiva de un Bitmap.

Código de la clase que representa el sistema de coordenadas: CoordinateSystem.java

```
1   package com.fluideditor.model.icon;
2
3   /**
4    * Sistema de coordenadas de Modelica.
5    *
6    * @author Jackson F. Reyes Bermeo
7    */
8   public class CoordinateSystem {
9
10      private Extent extent;
11      private boolean preserveAspectRatio = true;
12      private double initialScale;
13
14      public CoordinateSystem() {
15          initialScale = 0.1;
16      }
17
18      public CoordinateSystem(Extent extent) {
19          this.extent = extent;
20      }
21
22      public double getInitialScale() {
23          return initialScale;
24      }
25
```

```
26      public void setInitialScale(double initialScale) {
27          this.initialScale = initialScale;
28      }
29
30      public Extent getExtent() {
31          return extent;
32      }
33
34      public void setExent(Extent exent) {
35          this.extent = exent;
36      }
37
38      public boolean isPreserveAspectRatio() {
39          return preserveAspectRatio;
40      }
41
42      public void setPreserveAspectRatio(boolean preserveAspectRatio) {
43          this.preserveAspectRatio = preserveAspectRatio;
44      }
45
46  }
```

Código B.17: Implementación de la clase que modela el sistema de coordenadas.

Código de la clase que representa los Extent de Modelica: Extent.java

```
1   package com.fluideditor.model.icon;
2
3   import javafx.geometry.Point2D;
4
5   /**
6    * Representa al Extent de Modelica.
7    *
8    * @author Jackson F. Reyes Bermeo
9    */
10  public class Extent {
11
12      private final Point2D start;
13      private final Point2D end;
14
15      public Extent(double x1, double y1, double x2, double y2) {
16          start = new Point2D(x1, y1);
17          end = new Point2D(x2, y2);
18      }
19
20      public String getCodeString() {
21          String code = "extent={{" + start.getX() + "," + start.getY() +
                "},{" + end.getX() + "," + end.getY() + "}}";
22          return code;
23      }
24
25      public Point2D getStart() {
26          return start;
27      }
28
29      public Point2D getEnd() {
30          return end;
31      }
32
```

```
33      public double getHeight() {
34          return Math.abs(end.getY() - start.getY());
35      }
36
37      public double getWidth() {
38          return Math.abs(end.getX() - start.getX());
39      }
40
41      public double getXMed() {
42          return (end.getX() + start.getX()) / 2;
43      }
44
45      public double getYMed() {
46          return (end.getY() + start.getY()) / 2;
47      }
48
49      public boolean isAntiHorary() {
50          return (start.getX() * start.getY()) > 0;
51      }
52
53      public double getAngleCorrection() {
54          if (start.getX() < end.getX()) {
55              return 0.0;
56          } else {
57              return 180;
58          }
59      }
60  }
```

Código B.18: Implementación de la clase que representa los Extent de Modelica.

Código de la clase que representa el FilledShape de Modelica: FilledShape.java

```
1   package com.fluideditor.model.icon;
2
3   import java.util.ArrayList;
4   import java.util.List;
5   import javafx.scene.canvas.Canvas;
6   import javafx.scene.canvas.GraphicsContext;
7   import javafx.scene.paint.Color;
8   import javafx.scene.paint.CycleMethod;
9   import javafx.scene.paint.ImagePattern;
10  import javafx.scene.paint.LinearGradient;
11  import javafx.scene.paint.Paint;
12  import javafx.scene.paint.RadialGradient;
13  import javafx.scene.paint.Stop;
14
15  /**
16   * Representa al FilledSahpe de Modelica.
17   *
18   * @author Jackson F. Reyes Bermeo
19   */
20  public class FilledShape {
21
22      private Color lineColor = Color.BLACK;
23      private Color fillColor = Color.BLACK;
24      private LinePattern pattern = LinePattern.Solid;
25      private FillPattern fillPattern = FillPattern.None;
26      private double lineThickness = 0.5;
```

```
27      private Paint fillPaint; //relleno
28      private List<Double> linePattern; // Patrón de línea punteada
29
30      public FilledShape() {
31          fillPaint = null;
32          linePattern = new ArrayList<>();
33      }
34
35      public Color getLineColor() {
36          return lineColor;
37      }
38
39      public void setLineColor(Color lineColor) {
40          this.lineColor = lineColor;
41      }
42
43      public Color getFillColor() {
44          return fillColor;
45      }
46
47      public void setFillColor(Color fillColor) {
48          this.fillColor = fillColor;
49          configureFillPattern();
50      }
51
52      public FillPattern getFillPattern() {
53          return fillPattern;
54      }
55
56      public void setFillPattern(String fillPattern) {
57          this.fillPattern = FillPattern.valueOf(fillPattern);
58          configureFillPattern();
59      }
60
61      /**
62       * Obtener una imagen de un patron de fondo.
63       *
64       * @param typeRegilla Tipo de rejilla.
65       * @return Patron tipo Imagen.
66       */
67      private ImagePattern getImagePattern(FillPattern typeRegilla) {
68          double canvasWidth = 100;
69          double canvasHeight = 100;
70          Canvas canvas = new Canvas(canvasWidth, canvasHeight);
71          GraphicsContext gc = canvas.getGraphicsContext2D();
72          gc.setLineWidth(0.2);
73          gc.setStroke(lineColor);
74          gc.setFill(fillColor);
75          gc.fillRect(0, 0, canvasWidth, canvasHeight);
76          int gridSize = 100 / 16; // Tama o de la rejilla
77          switch (typeRegilla) {
78              case Horizontal:
79                  // Dibujar líneas horizontales
80                  for (double y = 0; y <= canvasHeight; y += gridSize) {
81                      gc.strokeLine(0, y, canvasWidth, y);
82                  }
83                  gc.rotate(0);
84                  break;
85              case Vertical:
86                  // Dibujar líneas verticales
87                  for (double x = 0; x <= canvasWidth; x += gridSize) {
```

```
88                      gc.strokeLine(x, 0, x, canvasHeight);
89                  }
90                  break;
91              case Cross:
92                  // Dibujar líneas verticales
93                  for (double x = 0; x <= canvasWidth; x += gridSize) {
94                      gc.strokeLine(x, 0, x, canvasHeight);
95                  }
96                  // Dibujar líneas horizontales
97                  for (double y = 0; y <= canvasHeight; y += gridSize) {
98                      gc.strokeLine(0, y, canvasWidth, y);
99                  }
100                 break;
101             case Forward:
102                 // Dibujar líneas \\\
103                 for (double x = -canvasWidth; x <= canvasWidth; x +=
                        gridSize) {
104                     gc.strokeLine(x, 0, x + canvasWidth, canvasHeight);
105                 }
106                 //gc.rotate(65);
107                 break;
108             case Backward:
109                 // Dibujar líneas ///
110                 for (double x = 0; x <= 2 * canvasWidth; x += gridSize)
                        {
111                     gc.strokeLine(x, 0, x - canvasWidth, canvasHeight);
112                 }
113                 //gc.rotate(-45);
114                 break;
115             case CrossDiag:
116                 // Dibujar líneas \\\
117                 for (double x = -canvasWidth; x <= canvasWidth; x +=
                        gridSize) {
118                     gc.strokeLine(x, 0, x + canvasWidth, canvasHeight);
119                 }
120                 // Dibujar líneas ///
121                 for (double x = 0; x <= 2 * canvasWidth; x += gridSize)
                        {
122                     gc.strokeLine(x, 0, x - canvasWidth, canvasHeight);
123                 }
124                 break;
125         }
126         ImagePattern imagePattern = new ImagePattern(canvas.snapshot(
                null, null));
127         return imagePattern;
128     }
129
130     /**
131      * Seleccionar el fillPatern definido.
132      */
133     private void configureFillPattern() {
134         switch (fillPattern) {
135             case None:
136                 fillPaint = null;
137                 break;
138             case Horizontal:
139                 fillPaint = getImagePattern(fillPattern);
140                 //System.out.println("Horizontal");
141                 break;
142             case Vertical:
143                 fillPaint = getImagePattern(fillPattern);
```

```
144                 //System.out.println("Vertical");
145                 break;
146             case Cross:
147                 fillPaint = getImagePattern(fillPattern);
148                 //System.out.println("Cross");
149                 break;
150             case Forward:
151                 fillPaint = getImagePattern(fillPattern);
152                 //System.out.println("Forward");
153                 break;
154             case Backward:
155                 fillPaint = getImagePattern(fillPattern);
156                 //System.out.println("Backward");
157                 break;
158             case CrossDiag:
159                 fillPaint = getImagePattern(fillPattern);
160                 System.out.println("CrossDiag");
161                 break;
162             case HorizontalCylinder:
163                 fillPaint = new LinearGradient(0.5, 0, 0.5, 1, true,
                        CycleMethod.NO_CYCLE, new Stop(0, lineColor), new
                        Stop(0.5, fillColor), new Stop(1, lineColor));
164
165                 break;
166             case VerticalCylinder:
167                 // Crear un objeto LinearGradient con los colores RGB
168                 fillPaint = new LinearGradient(0, 0.5, 1, 0.5, true,
                        CycleMethod.NO_CYCLE, new Stop(0, lineColor), new
                        Stop(0.5, fillColor), new Stop(1, lineColor));
169                 break;
170             case Sphere:
171                 double centerX = 0.5;
172                 double centerY = 0.5;
173                 double radius = 1.0;
174                 boolean proportional = true;
175                 fillPaint = new RadialGradient(0, 0, centerX, centerY,
                        radius, proportional, CycleMethod.NO_CYCLE,
176                             new Stop(0, fillColor), new Stop(1, lineColor));
177                 break;
178             case Solid:
179                 fillPaint = fillColor;
180                 break;
181             default:
182
183         }
184     }
185
186     public void setPattern(String pattern) {
187         this.pattern = LinePattern.valueOf(pattern);
188         configureLinePattern();
189     }
190
191     private void configureLinePattern() {
192         switch (pattern) { //None , Solid , Dash , Dot , DashDot ,
                DashDotDot
193             case None:
194                 this.linePattern = null;
195                 break;
196             case Solid:
197                 this.linePattern.add(1.0);
198                 break;
```

```
199             case Dash:
200                 this.linePattern.add(10.0);
201                 this.linePattern.add(10.0);
202                 break;
203             case Dot:
204                 this.linePattern.add(1.0);
205                 this.linePattern.add(10.0);
206                 break;
207             case DashDot:
208                 this.linePattern.add(15.0);
209                 this.linePattern.add(10.0);
210                 this.linePattern.add(3.0);
211                 this.linePattern.add(10.0);
212                 break;
213             case DashDotDot:
214                 this.linePattern.add(15.0);
215                 this.linePattern.add(10.0);
216                 this.linePattern.add(3.0);
217                 this.linePattern.add(10.0);
218                 this.linePattern.add(3.0);
219                 this.linePattern.add(10.0);
220                 break;
221 
222         }
223     }
224 
225     public void setLineThickness(double lineThickness) {
226         this.lineThickness = lineThickness;
227     }
228 
229     public List<Double> getLinePattern() {
230         return linePattern;
231     }
232 
233     public double getLineThickness() {
234         return lineThickness;
235     }
236 
237     public Paint getFillPaint() {
238         return fillPaint;
239     }
240 
241     public void setFillPaint(Paint fillPaint) {
242         this.fillPaint = fillPaint;
243     }
244 
245 }
```

Código B.19: Implementación de la clase que representa el FilledShape de Modelica.

Código del enumerado con los patrones de rellenos de Modelica: FillPattern.java

```
package com.fluideditor.model.icon;

/**
 * Los diferentes tipos de patrones de fondo.
 * @author Jackson F. Reyes Bermeo
 */
public enum FillPattern {
    None ,
    Solid ,
    Horizontal ,
    Vertical ,
    Cross ,
    Forward ,
    Backward ,
    CrossDiag ,
    HorizontalCylinder ,
    VerticalCylinder ,
    Sphere
}
```

Código B.20: Implementación del enumerado con los patrones de rellenos de Modelica.

Código del enumerado con los patrones de línea de Modelica: LinePattern.java

```
package com.fluideditor.model.icon;

/**
 * Enumerados de los patrones de Linea.
 * @author Jackson F. Reyes Bermeo
 */
public enum LinePattern {
    None , Solid , Dash , Dot , DashDot , DashDotDot
}
```

Código B.21: Implementación del enumerado con los patrones de línea de Modelica.

Código de la clase que representa los desplazamientos de los iconos en el área de diseño: Placement.java

```
1  package com.fluideditor.model.icon;
2
3  /**
4   * Representa el Placement de Modelica.
5   *
6   * @author Jackson F. Reyes Bermeo
7   */
8  public class Placement {
9
10     boolean visible = true;
11     Transformation transformation;
12     boolean iconVisible;
13     Transformation icoTransformation;
14
15     public Placement() {
16         transformation = new Transformation();
17         icoTransformation = new Transformation();
18     }
19
20     public String getCodeString() {
21         String code = "Placement(visible=" + visible + ",";
22         code += transformation.getCodeString() + ")";
23         return code;
24     }
25
26     public boolean isVisible() {
27         return visible;
28     }
29
30     public void setVisible(boolean visible) {
31         this.visible = visible;
32     }
33
34     public Transformation getTransformation() {
35         return transformation;
36     }
37
38     public void setTransformation(Transformation transformation) {
39         this.transformation = transformation;
40     }
41
42     public boolean isIconVisible() {
43         return iconVisible;
44     }
45
46     public void setIconVisible(boolean iconVisible) {
47         this.iconVisible = iconVisible;
48     }
49
50     public Transformation getIcoTransformation() {
51         return icoTransformation;
52     }
53
54     public void setIcoTransformation(Transformation icoTransformation) {
55         this.icoTransformation = icoTransformation;
56     }
57
```

```
58 }
```

Código B.22: Implementación de la clase que representa los desplazamientos de los iconos en el área de diseño.

Código de la clase que representa las transformaciones de los iconos en el área de diseño: Transformation.java

```
1  package com.fluideditor.model.icon;
2
3  import java.text.DecimalFormat;
4  import java.text.DecimalFormatSymbols;
5  import java.util.Locale;
6  import javafx.geometry.Point2D;
7
8  /**
9   * Representa las transformaciones que realiza Modelica al los iconos.
10  *
11  * @author Jackson F. Reyes Bermeo
12  */
13 public class Transformation {
14
15     private Point2D origin;
16     private Extent extent;
17     private double rotation = 0;
18
19     public Transformation() {
20         origin = new Point2D(0, 0);
21     }
22
23     public String getCodeString() {
24         // Crear un objeto DecimalFormat para redondear el número a dos
               decimales
25         DecimalFormat decimalFormat = new DecimalFormat("#.##", new
               DecimalFormatSymbols(Locale.US));
26         String xOrigin = decimalFormat.format(origin.getX());
27         String yOrigin = decimalFormat.format(origin.getY());
28         String code = "transformation(origin={" + xOrigin + "," +
               yOrigin + "}";
29         code += "," + extent.getCodeString() + ")";
30         return code;
31     }
32
33     public Point2D getOrigin() {
34         return origin;
35     }
36
37     public void setOrigin(Point2D origin) {
38         this.origin = origin;
39     }
40
41     public Extent getExtent() {
42         return extent;
43     }
44
45     public void setExtent(Extent extent) {
46         this.extent = extent;
47     }
```

```
48
49      public double getRotation() {
50          return rotation;
51      }
52
53      public void setRotation(double ritation) {
54          this.rotation = ritation;
55      }
56
57  }
```

Código B.23: Implementación de la clase que representa las transformaciones de los iconos en el área de diseño.

B-3.2. Implementación del árbol de componentes

Código de la clase encargada de leer ficheros Modelica y construir el árbol de componentes: ModelicaAnalizer.java

```
1   package com.fluideditor.model.tree;
2
3   import java.io.BufferedReader;
4   import java.io.BufferedWriter;
5   import java.io.File;
6   import java.io.FileReader;
7   import java.io.FileWriter;
8   import java.io.IOException;
9   import java.util.ArrayList;
10  import java.util.List;
11  import java.util.regex.Matcher;
12  import java.util.regex.Pattern;
13  import javafx.scene.control.TreeItem;
14
15  /**
16   *
17   * @author Jackson F. Reyes Bermeo
18   */
19  public class ModelicaAnalizer {
20
21      private final String pathFile;
22      public List<String> fileContent;
23      public List<String> fileFlatContent;
24      private final boolean isModelicaFile;
25      private Boolean isPackage;
26      private String within;
27      private String rootName;
28      private String typeComponent;
29      private String levelNode = "";
30
31      public ModelicaAnalizer(String pathFile) {
32          this.isModelicaFile = hasExtension(new File(pathFile), ".mo");
33          this.pathFile = pathFile;
34          fileContent = new ArrayList<>();
35          fileFlatContent = new ArrayList<>();
36      }
37
```

```
38      public String getTypeComponent() {
39          return typeComponent;
40      }
41
42      public void setLevelNode(String levelNode) {
43          this.levelNode = levelNode;
44      }
45
46      public boolean isIsModelicaFile() {
47          return isModelicaFile;
48      }
49
50      public Boolean isPackage() {
51          return isPackage;
52      }
53
54      // Cargar el fichero
55      private Boolean load() {
56          if (!this.isModelicaFile) {// Si no es fichero modelica (.mo) no
                lo carga
57              return false;
58          }
59          String line;
60          try (BufferedReader bufferedReader = new BufferedReader(new
                FileReader(pathFile))) {
61              int currentLine = 0;
62              while ((line = bufferedReader.readLine()) != null) {
63                  currentLine++;
64                  fileContent.add(line);
65              }
66              bufferedReader.close();
67          } catch (IOException ex) {
68              return false;
69          }
70          return true;
71      }
72
73      /**
74       * Comprobar si el fichero tiene una extensión especifica.
75       *
76       * @param file Fichero a comprobar.
77       * @param extension Extensión que se desea verificar.
78       * @return True si tiene la extensión buscada, false en el resto de
             casos.
79       */
80      private boolean hasExtension(File file, String extension) {
81          String fileName = file.getName();
82          return fileName.endsWith(extension);
83      }
84
85      /**
86       * Elimina los comentarios, lineas en blanco y deja al fichero en el
             que
87       * cada linea tiene una intrucción que termina en ;
88       */
89      private void toFlat() {
90          StringBuilder contenido = new StringBuilder();
91          //String reservedWordPattern = "^(initial equation|encapsulated|
                partial|package|model|block|class|connector|record|equation|
                for|while|if|else)";
```

```
92          String reservedWordPattern = "^(initial equation|encapsulated|
                partial|package|model|block|class|connector|record|equation)
                ";
93          Pattern pattern = Pattern.compile(reservedWordPattern);
94          for (String lineContent : fileContent) {
95              lineContent = lineContent.strip();
96              if (lineContent.isBlank() || lineContent.isEmpty() ||
                    lineContent.startsWith("//")) {
97                  continue; // No se leen comentarios o lineas en blanco
98              }
99              //contiene palabra clave y no tenga un igual (esto sucede en
                     los replaceable y redeclare), a ado indicador
100             if (pattern.matcher(lineContent).find() && !lineContent.
                    contains("=") && !contenido.toString().contains("
                    annotation")) {
101                 fileFlatContent.add("::" + lineContent);
102                 continue;
103             }
104             //Lineas que no tengan al final un ;, Se concatenan hasta
                    encontrar ;
105             contenido.append(" ").append(lineContent);
106             if (lineContent.endsWith(";")) {
107                 // si es una anotación verifico que no sea ; de alguna
                        documentacion
108                 if (contenido.toString().replaceAll("\\s", "").contains(
                        "annotation")) {
109                     int parrentesisCount = 0;
110                     // Utilizar expresiones regulares para eliminar el
                            HTML de la documentación
111                     String cadenaSinHTML = contenido.toString().
                            replaceAll("<html>.*?</html>", "");
112                     int startAnnotation = cadenaSinHTML.replace("
                            annotation (", "annotation(").indexOf("
                            annotation(") + 10;
113                     String contentAnnotation = cadenaSinHTML.substring(
                            startAnnotation);
114                     String parentesisForm = contentAnnotation.replaceAll
                            ("[^()]", "");
115                     for (char c : parentesisForm.toCharArray()) {
116                         if (c == '(') {
117                             parrentesisCount++;
118                         } else if (c == ')') {
119                             parrentesisCount--;
120                         }
121                     }
122                     if (parrentesisCount <= 0 && lineContent.endsWith(")
                            ;")) {
123                         fileFlatContent.add(" " + contenido.toString());
124                         contenido.setLength(0); //reset
125                     }
126                 } else {
127                     fileFlatContent.add(contenido.toString());
128                     contenido.setLength(0); //reset
129                 }
130             }
131         }
132
133     }
134
135     /**
136      * Guardar una lista en un fichero.
```

```
137         *
138         * @param pathFile Ruta del fichero.
139         * @param lista Lista que se guardará en el fichero.
140         * @return True si se completo correctamente.
141         */
142        public boolean saveContent(String pathFile, List<String> lista) {
143            try (BufferedWriter writer = new BufferedWriter(new FileWriter(
                   pathFile))) {
144                for (int i = 1; i < lista.size(); i++) {
145                    writer.write(lista.get(i));
146                    writer.newLine();
147                }
148                System.out.println("Fichero" + pathFile + " guardado
                       correctamente");
149            } catch (IOException e) {
150                return false;
151            }
152            return true;
153        }
154
155        /**
156         * Extrae el código en una lista de un modelo u otro componente por
                su
157         * nombre.
158         *
159         * @param name Nombre del compone a extraer.
160         * @return Lista del contenido del componente.
161         */
162        private List<String> extractModelicaComponentByName(String name) {
163            String patron = "(\\w+)+\\s" + name;
164            Pattern patternInit = Pattern.compile("::" + patron);
165            Pattern patternEnd = Pattern.compile(patron + ";");
166            Matcher matcherInit;
167            Matcher matcherEnd;
168            List<String> tempList = new ArrayList<>();
169            boolean start = false;
170            boolean stop = false;
171            for (int i = 0; i < fileFlatContent.size(); i++) {
172                String line = fileFlatContent.get(i);
173                matcherInit = patternInit.matcher(line);
174                matcherEnd = patternEnd.matcher(line);
175                if (matcherInit.find()) {
176                    start = true;
177                }
178                if (matcherEnd.find()) {
179                    stop = true;
180                }
181                if (start) {
182                    tempList.add(line);
183                }
184                if (stop) {
185                    break;
186                }
187            }
188            return tempList;
189        }
190
191        /**
192         * Construye un árbol de los componentes de un Model de forma
                recursiva.
193         *
```

```
194         * @param type Tipo de componente: Model, Connector, Class, etc.
195         * @param nameComponent Nombre del componente.
196         * @return Arbol completo con el código de cada elemento en cada
                rama.
197         */
198        public TreeItem<NodeItemCode> createTreeItem(String type, String
               nameComponent) {
199            NodeItemCode currentCode = new NodeItemCode(nameComponent);
200            currentCode.setType(type);
201            TreeItem<NodeItemCode> currentTreeItem = new TreeItem(
                   currentCode);
202            //String genericPattern = "::((?:encapsulate\\s)|(?:partial\\s))
                   ?(package|model|connector|record|function)+\\s";
203            String genericPattern = "::((?:encapsulate\\s)|(?:partial\\s))?(
                   package|class|model|block|connector|record|function)+\\s";
204            String firsName = type + " " + nameComponent;
205            String endName = "end " + nameComponent + ";";
206            String newComponent = genericPattern + "+(\\w+)";
207            Pattern endPattern = Pattern.compile(endName);
208            Pattern firstPattern = Pattern.compile(firsName);
209            Pattern newPattern = Pattern.compile(newComponent);
210
211            Pattern inlineConnectorPattern = Pattern.compile("(\\w+)\\s(\\w
                   +)\\s?=\\s?(input|output)\\s(\\w+)\\s(\\\".*[\\\"])\\s");
212            boolean startStore = false;
213            int maxIter = fileFlatContent.size();
214            int contIter = 0;
215            while (!fileFlatContent.isEmpty()) {
216                if (contIter > maxIter) {
217                    return null; // error se ha desbordado
218                }
219                contIter++;
220                String line = fileFlatContent.get(0);
221
222                // Tratamiento para los conector declarados INLINE
223                Matcher inlineConnectorMatcher = inlineConnectorPattern.
                       matcher(line);
224                if (inlineConnectorMatcher.find()) {
225                    String typeConnector = inlineConnectorMatcher.group(1);
226                    String nameConnector = inlineConnectorMatcher.group(2);
227                    String commentConnector = inlineConnectorMatcher.group
                           (5);
228                    String allMatch = inlineConnectorMatcher.group();
229                    String annotationStr = line.substring(line.indexOf("
                           annotation"));
230
231                    currentCode.addCodeLine("<<codeChildren" + nameConnector
                           + ">>");
232
233                    NodeItemCode connectorCode = new NodeItemCode(
                           nameConnector);
234                    connectorCode.setType("::" + typeConnector);
235                    connectorCode.addCodeLine("::" + typeConnector.strip() +
                           " " + nameConnector + " " + commentConnector);
236                    connectorCode.addCodeLine(annotationStr);
237                    connectorCode.addCodeLine("end " + nameConnector + ";");
238                    currentTreeItem.getChildren().add(new TreeItem<>(
                           connectorCode));
239                    fileFlatContent.remove(0);
240                    continue;
241                }
```

```
242             Matcher matcherFirst = firstPattern.matcher(line);
243             Matcher matcherEnd = endPattern.matcher(line);
244             Matcher matcherNew = newPattern.matcher(line);
245             if (matcherFirst.find()) { // Empieza a guardar code en el
                    node
246                 startStore = true;
247                 currentCode.addCodeLine(line);
248                 fileFlatContent.remove(0);
249                 levelNode += "-";
250             } else if (matcherEnd.find()) {
251                 currentCode.addCodeLine(line);
252                 fileFlatContent.remove(0);
253                 levelNode = levelNode.substring(0, levelNode.length() -
                        1);
254                 return currentTreeItem;
255             } else if (matcherNew.find()) {
256                 String nameNewComponent = matcherNew.group(matcherNew.
                        groupCount());
257                 nameNewComponent = nameNewComponent.strip();
258                 String typeNewComponent = matcherNew.group().replace(
                        nameNewComponent, "");
259                 typeNewComponent = typeNewComponent.strip();
260                 currentCode.addCodeLine("<<codeChildren" +
                        nameNewComponent + ">>"); // Marca para indicar que
                        ahí va el código de su hijo
261                 currentTreeItem.getChildren().add(createTreeItem(
                        typeNewComponent, nameNewComponent));
262             } else {
263                 currentCode.addCodeLine(line);
264                 fileFlatContent.remove(0);
265             }
266         }
267         return currentTreeItem;
268     }
269
270     /**
271      * Comprueba que sean correctos los niveles de anidamiento
272      *
273      * @return True si el nivel de anidamiento es cero, si es distinto
             de cero
274      * significa que no se cerro completamente el anidamiento.
275      */
276     public boolean checkHierarchy() {
277         return levelNode.length() == 0;
278     }
279
280     /**
281      * Visualizar el árbol generado: Imprimiendo su código de forma
             recursiva.
282      *
283      * @param item Rama actual.
284      * @param level Nivel de anidamiento.
285      */
286     public void printTreeItemHierarchy(TreeItem<NodeItemCode> item, int
            level) {
287         // Imprimir el nombre del elemento indentado según el nivel
288         String indentation = "- ".repeat(level);
289         System.out.println(indentation + item.getValue());
290         // Recorrer los hijos del elemento de forma recursiva
291         for (TreeItem<NodeItemCode> child : item.getChildren()) {
292             printTreeItemHierarchy(child, level + 1);
```

```
293         }
294     }
295 
296     /**
297      * Cargar, Aplanamiento y Obtener propiedades básicas del fichero:
             Nombre,
298      * within, package, etc.
299      */
300     public void analize() {
301         if (fileContent.isEmpty()) {
302             this.load();
303         }
304         if (fileFlatContent.isEmpty() && !fileContent.isEmpty()) {
305             this.toFlat();
306         }
307         if (!fileFlatContent.isEmpty()) {
308             this.extractMainProperties();
309         }
310     }
311 
312     /**
313      * Buscar si el fichero cargado es un package.
314      *
315      * @return True si es package.
316      */
317     private boolean findPackage() {
318         for (int i = 1; i < fileFlatContent.size(); i++) {
319             String line = fileFlatContent.get(i);
320             if (line.contains("package")) {
321                 return true;
322             }
323         }
324         return false;
325     }
326 
327     /**
328      * Buscar si el fichero contiene un withing.
329      *
330      * @return within route si el fichero tiene within, empty caso
             contrario.
331      */
332     private String findWithin() {
333         for (int i = 1; i < fileFlatContent.size(); i++) {
334             String line = fileFlatContent.get(i);
335             if (line.contains("within")) {
336                 return line.substring(line.indexOf("within") + 7).strip
                        ();
337             }
338         }
339         return "";
340     }
341 
342     /**
343      * Extraer las principales propiedades del fichero: Nombre, tipo,
             within,
344      * isPackage.
345      */
346     private void extractMainProperties() {
347         String basicPattern = "((?:encapsulate\\s)|(?:partial\\s))?(
                package|model|class|record|connector)+\\s";
348         Pattern rootPattern = Pattern.compile(basicPattern + "+(\\w+)");
```

```
349         Matcher matcher = null;
350         boolean existPropieties = false;
351         for (int i = 0; i < fileFlatContent.size(); i++) {
352             String line = fileFlatContent.get(i);
353             matcher = rootPattern.matcher(line);
354             if (matcher.find()) {
355                 existPropieties = true;
356                 break;
357             }
358         }
359         if (existPropieties) {
360             this.rootName = matcher.group(matcher.groupCount());
361             this.typeComponent = matcher.group().replace(rootName, "").
                    strip();
362         }
363         this.within = findWithin();
364         this.isPackage = findPackage();
365     }
366 
367     public String getNameOfRootNode() {
368         return rootName;
369     }
370 
371     public void saveStringOnFile(String fileName, String contentStr) {
372         try (BufferedWriter writer = new BufferedWriter(new FileWriter(
                fileName))) {
373             writer.write(contentStr);
374             writer.flush();
375             System.out.println("El texto se ha guardado correctamente en
                     el archivo.");
376         } catch (IOException e) {
377             System.out.println("Error al guardar en el archivo.");
378         }
379     }
380 
381     public List<String> getFlatCode() {
382         this.extractMainProperties();
383         return fileFlatContent;
384     }
385 
386     /**
387      * Obtener el código en una sola linea (String).
388      *
389      * @return String con el código aplanado.
390      */
391     public String getStringCode() {
392         String code = "";
393         for (String line : fileFlatContent) {
394             code += line + "\n";
395         }
396         return code;
397     }
398 
399     public static void main(String args[]) {
400         // Test: cargar un connector declarado INLINE.
401         ModelicaAnalizer analizerConnectorInline = new ModelicaAnalizer(
                "C:\\Users\\jacks\\Desktop\\FluidEditor\\lib\\Modelica\\
                Blocks\\Interfaces.mo");
402         analizerConnectorInline.analize();
403         analizerConnectorInline.saveContent("E:\\InterfaceReaded.mo",
                analizerConnectorInline.fileContent);
```

```
404         analizerConnectorInline.saveContent("E:\\InterfaceFlat.mo",
                analizerConnectorInline.fileFlatContent);
405
406         // Test: comprobar la carga de un fichero Modelica.
407         String fileName = "Dissipation.mo";
408         String pathModel = "C:\\Users\\jacks\\AppData\\Roaming\\.
                openmodelica\\libraries\\Modelica 4.0.0+maint.om\\Fluid\\";
409         ModelicaAnalizer analizer = new ModelicaAnalizer(pathModel +
                fileName);
410         String pathSave = "C:\\Users\\jacks\\OneDrive - UNED\\0_TFM\\
                Modelica\\Ejercicios\\";
411         analizer.analize();
412         analizer.saveContent(pathSave + "flat" + fileName, analizer.
                fileFlatContent);
413         analizer.saveContent(pathSave + "readed" + fileName, analizer.
                fileContent);
414         String rootName = analizer.getNameOfRootNode();
415         String startMark = "::";
416         String typeComponent = analizer.getTypeComponent();
417         typeComponent = startMark + typeComponent;
418         analizer.printTreeItemHierarchy(analizer.createTreeItem(
                typeComponent, rootName), 0);
419         System.out.println(rootName + " ES CORRECTO?: " + analizer.
                checkHierarchy());
420
421         //############## Comprobar todo el directorio ***#//
422         //String pathCheck = "C:\\Users\\jacks\\AppData\\Roaming\\.
                openmodelica\\libraries\\Modelica 4.0.0+maint.om\\Fluid\\";
423         String pathCheck = "C:\\Users\\jacks\\Documents\\
                NetBeansProjects\\FluidEditor\\lib\\Modelica";
424         List<String> incorrectsFiles = new ArrayList<>();
425         for (File child : new File(pathCheck).listFiles()) {
426             System.out.println("File: " + child.getAbsolutePath());
427             if (child.isFile()) {
428                 ModelicaAnalizer modAnalizer = new ModelicaAnalizer(
                        child.getAbsolutePath());
429                 modAnalizer.analize();
430                 modAnalizer.setLevelNode(""); // reinicio el level cada
                        intento
431                 String rootNameChild = modAnalizer.getNameOfRootNode();
432                 String startMarkV2 = "::";
433                 String typeComponentV2 = modAnalizer.getTypeComponent();
434                 modAnalizer.createTreeItem(startMarkV2 + typeComponentV2
                        , rootNameChild);
435                 boolean isCorrectTree = modAnalizer.checkHierarchy();
436                 System.out.println(typeComponentV2 + " " + rootNameChild
                         + " ES CORRECTO?: " + isCorrectTree);
437                 if (!isCorrectTree) {
438                     incorrectsFiles.add(rootNameChild);
439                 }
440             }
441         }
442
443         // imprimer los ficheros erroneos
444         System.out.println("**** Incorrect files *****");
445         for (String fileError : incorrectsFiles) {
446             System.out.println(fileError);
447         }
448
449     }
450
```

```
}
```

Código B.24: Implementación de la clase encargada de leer ficheros Modelica y construir el árbol de componentes.

Código de la clase que almacena información en cada nodo del árbol de componentes: NodeItemCode.java

```
package com.fluideditor.model.tree;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import javafx.scene.Node;

/**
 *
 * @author Jackson F. Reyes Bermeo
 */
public class NodeItemCode implements Serializable {

    private String name;
    private String type;
    private String path;
    private String route = "";
    private List<String> code;
    private Node iconGraphic = null;

    public NodeItemCode() {
        code = new ArrayList<>();
    }

    public NodeItemCode(String name) {
        this.name = name;
        code = new ArrayList<>();
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public Node getIconGraphic() {
        return iconGraphic;
    }

    public void setIconGraphic(Node iconGraphic) {
        this.iconGraphic = iconGraphic;
    }

    public boolean existGraphic() {
        return iconGraphic != null;
    }

    public String getPath() {
```

```
51          return path;
52      }
53
54      public void setPath(String path) {
55          this.path = path;
56      }
57
58      public String getName() {
59          return name;
60      }
61
62      public void setName(String name) {
63          this.name = name;
64      }
65
66      public List<String> getCode() {
67          return code;
68      }
69
70      public void addCodeLine(String line) {
71          this.code.add(line);
72      }
73
74      public String getRoute() {
75          return route;
76      }
77
78      public void setRoute(String route) {
79          this.route = route;
80      }
81
82  }
```

Código B.25: Implementación de la clase que almacena información en cada nodo del árbol de componentes.

B-3.3. Implementación de las clases internas Modelica

Código de la clase encargada de gestionar el modelo interno: ModelManager.java

```
1  package com.fluideditor.model.modelica;
2
3  import com.fluideditor.model.icon.CodeAnalizer;
4  import com.fluideditor.model.icon.LineAnnotation;
5  import com.fluideditor.model.icon.Placement;
6  import com.fluideditor.model.tree.NodeItemCode;
7  import java.util.HashMap;
8  import java.util.List;
9  import java.util.Map;
10 import javafx.scene.control.TreeItem;
11
12 /**
13  * Gestor de la interación con el Modelo Modelica.
14  *
15  * @author Jackson F. Reyes Bermeo
```

```
16  */
17 public class ModelManager {
18
19     private final String rootPath;
20     private final TreeItem<NodeItemCode> rootTree;
21     private final CodeAnalizer codeAnalizer;
22     private Model model;
23     private final NodeItemCode currentNodeItemCode;
24     private final Map<String, List<String>> allDeclarations;
25     private final boolean DEBUG = false;
26
27     public ModelManager(TreeItem<NodeItemCode> rootTree, NodeItemCode
           nodeItemCode, String rootPath) {
28         this.rootPath = rootPath;
29         allDeclarations = new HashMap<>();
30         this.rootTree = rootTree;
31         currentNodeItemCode = nodeItemCode;
32         codeAnalizer = new CodeAnalizer(nodeItemCode.getCode(), rootPath
               );
33         codeAnalizer.setComponentName(nodeItemCode.getName());
34         codeAnalizer.setComponentType(nodeItemCode.getType());
35         codeAnalizer.analize();
36     }
37
38     /**
39      * Obtiene la información de una linea de conexión definida en
            Modelica.
40      *
41      * @param lineConnectrionString Contiene la linea que define la
            conexión en
42      * Modelica.
43      * @return Map con el nombre del elemento, sus conectores.
44      */
45     public Map<String, String> getConnectionMapByLineString(String
           lineConnectrionString) {
46         return codeAnalizer.getConnectionMapByLineString(
               lineConnectrionString);
47     }
48
49     /**
50      * Obtiene la información de la linea de conexión definida mediante
51      * anotaciones.
52      *
53      * @param lineConnectionString Contiene la anotación de la linea.
54      * @return LineAnnotation que contiene información de la linea de
            conexión.
55      */
56     public LineAnnotation getLineAnnotationOfConnectionByLineString(
           String lineConnectionString) {
57         return codeAnalizer.getLineAnnotationOfConnectionByLineString(
               lineConnectionString);
58     }
59
60     public Map<String, List<String>> getAllDeclarationsMap() {
61         if (allDeclarations.isEmpty()) {
62             this.makeAllDeclarations(currentNodeItemCode);
63         }
64         return allDeclarations;
65     }
66
67     /**
```

```
68      * Contruye un Map con todos los elementos definidos en bloque de
69      * declaraciones del fichero Modelica.
70      *
71      *
72      * @param nodeCodeItem Código actual que se va analizar.
73      * @return
74      */
75     private boolean makeAllDeclarations(NodeItemCode nodeCodeItem) {
76         List<String> actualCodeList = nodeCodeItem.getCode();
77         String currentName = nodeCodeItem.getName();
78         String currentType = nodeCodeItem.getType();
79         String route = nodeCodeItem.getRoute();
80         CodeAnalizer codeAnalizerTemp = new CodeAnalizer(actualCodeList,
               rootPath);
81         codeAnalizerTemp.setComponentName(currentName);
82         codeAnalizerTemp.setComponentType(currentType);
83         codeAnalizerTemp.analize();
84
85         //Copiar las declarations a la lista global
86         Map<String, List<String>> currentDeclarationsMap =
               codeAnalizerTemp.getDeclarations();
87         addCurrentHashMapToGlobalHashMap(currentDeclarationsMap);
88
89         // Buscar declaraciones en las herencias: extends
90         for (String extendLine : currentDeclarationsMap.get("extends"))
               {
91             String routeToFind = extendLine.replace("extends", "").
                   replace(";", "").strip();
92             routeToFind = routeToFind.replaceAll("(\\(.*\\))", "");
93             if (!routeToFind.contains("Modelica.")) {//No tiene ruta
                   completa, toma la misma del actual fichero
94                 String currentCodeName = nodeCodeItem.getName();
95                 String parentRoute = nodeCodeItem.getRoute();
96                 routeToFind = normalizePath(parentRoute, routeToFind,
                       currentCodeName);
97             }
98
99             NodeItemCode nodeExtend = getNodeByRute(rootTree,
                   routeToFind);
100            if (nodeExtend != null) {
101                makeAllDeclarations(nodeExtend);
102            } else {
103                if (DEBUG) {
104                    System.out.println("ModelManager---> No se encontro
                           la ruta del extend: node:" + nodeCodeItem.
                           getName() + "\troute:" + routeToFind);
105                }
106            }
107
108        }
109        return false;
110    }
111
112    /**
113     * Intenta generar una ruta absoluta a partir de una relativa.
114     *
115     *
116     * @param parentPath Ruta un nivel por encima de la relativa.
117     * @param relativePath Ruta relativa.
118     * @param currentComponentName Nombre del componente.
119     * @return Ruta normalizada.
```

```
120         */
121         private String normalizePath(String parentPath, String relativePath,
                String currentComponentName) {
122             String routeToFind = parentPath.replace(currentComponentName,
                    relativePath);
123             String[] routesName = routeToFind.split("\\.");
124             String absolutePath = "";
125             String previousName = "";
126             for (String name : routesName) {
127                 if (!name.equals(previousName)) {//eliminar repeticiones en
                        la ruta
128                     absolutePath += name + ".";
129                 }
130                 previousName = name;
131             }
132             absolutePath = absolutePath.substring(0, absolutePath.length() -
                    1);//eliminar el último punto
133             return absolutePath;
134         }
135 
136         /**
137          * Agrega las declaraciones actuales a las declaraciones globales.
138          *
139          * @param currentDeclarationsMap
140          */
141         private void addCurrentHashMapToGlobalHashMap(Map<String, List<
                String>> currentDeclarationsMap) {
142             if (allDeclarations.get("parameters") == null) {
143                 allDeclarations.putAll(currentDeclarationsMap);
144             } else {
145                 List<String> tempList = allDeclarations.get("parameters");
146                 tempList.addAll(currentDeclarationsMap.get("parameters"));
147                 allDeclarations.put("parameters", tempList);
148 
149                 tempList = allDeclarations.get("replaceable");
150                 tempList.addAll(currentDeclarationsMap.get("replaceable"));
151                 allDeclarations.put("replaceable", tempList);
152             }
153         }
154 
155         /**
156          * Obtener el código de un Node que se buscar en el árbol por su
                ruta.
157          *
158          * @param rootTree Arbol en el que se busca el Node (código del node
                ).
159          * @param route Ruta que se va ha buscar.
160          * @return El código correspodiente a la ruta o null.
161          */
162         private NodeItemCode getNodeByRute(TreeItem<NodeItemCode> rootTree,
                String route) {
163             if (rootTree.getValue().getRoute().equals(route)) {
164                 return rootTree.getValue();
165             }
166             for (TreeItem<NodeItemCode> child : rootTree.getChildren()) {
167                 NodeItemCode foundNode = getNodeByRute(child, route);
168                 if (foundNode != null) {
169                     return foundNode; // Se ha encontrado el nodo en un hijo
170                 }
171             }
172             return null;
```

```
173     }
174
175     /**
176      * Obtiene un Map con los elementos de las redefiniciones extraido
             de una
177      * linea String.
178      *
179      * @param lineRedefinitionsString String que contiene las
             redefiniciones.
180      * @return Map con las redefiniciones.
181      */
182     public Map<String, String> getRedefinitionsByLineString(String
             lineRedefinitionsString) {
183         return codeAnalizer.getRedefinitionsByLineString(
                 lineRedefinitionsString);
184     }
185
186     public List<String> getTypeAndNameComponentByLine(String line) {
187         return codeAnalizer.getTypeAndNameComponentByLine(line);
188     }
189
190     public String getWithin() {
191         return codeAnalizer.getWithin();
192     }
193
194     public Placement getPlacementByLineString(String line) {
195         return codeAnalizer.getPlacement(line);
196     }
197
198     /**
199      * Devuelve los parametros locales del modelo
200      *
201      * @return Objeto con los parametros del modelo local.
202      */
203     public ModelicaParameter getParameterModel() {
204         ModelicaParameter parameter = codeAnalizer.getModelicaParameters
                 ();
205         parameter.setPath(currentNodeItemCode.getRoute());
206         return parameter;
207     }
208
209     /**
210      * Obtiene todos los parametros del Modelo, incluido las herencias.
211      *
212      * @return
213      */
214     public ModelicaParameter getAllParameterModel() {
215         makeAllDeclarations(currentNodeItemCode);
216         codeAnalizer.setCompleteDeclarations(allDeclarations);
217         ModelicaParameter parameter = codeAnalizer.
                 getCompletedModelicaParameters();
218         parameter.setPath(currentNodeItemCode.getRoute());
219         return parameter;
220     }
221
222     /**
223      * Obtiene una referencia al modelo actual.
224      *
225      * @return
226      */
227     public ModelicaClass getModel() {
```

```
228         model = new Model(currentNodeItemCode.getName());
229         model.setRoute(currentNodeItemCode.getRoute());
230         model.setAbsolutePath(currentNodeItemCode.getPath());
231         model.setWithin(codeAnalizer.getWithin());
232         return model;
233     }
234 }
```

Código B.26: Implementación de la clase encargada de gestionar el modelo interno.

Código de la clase abstracta que representa a cualquier clase definida en Modelica: ModelicaClass.java

```
1  package com.fluideditor.model.modelica;
2
3  /**
4   * Clase abstracta que representa a todos los componentes de modelica:
        Model,
5   * Connector, Block, etc.
6   *
7   * @author Jackson F. Reyes Bermeo
8   */
9  public abstract class ModelicaClass {
10
11     protected String name;
12     protected String absolutePath;
13     protected String route;
14
15     public ModelicaClass(String name) {
16         this.name = name;
17     }
18
19     public String getName() {
20         return name;
21     }
22
23     public void setName(String name) {
24         this.name = name;
25     }
26
27     public String getAbsolutePath() {
28         return absolutePath;
29     }
30
31     public void setAbsolutePath(String absolutePath) {
32         this.absolutePath = absolutePath;
33     }
34
35     public String getRoute() {
36         return route;
37     }
38
39     public void setRoute(String route) {
40         this.route = route;
41     }
42
43     public abstract ModelicaClass getModelicaComponent();
44
```

```
}
```

Código B.27: Implementación de la clase abstracta que representa a cualquier clase definida en Modelica.

Código de la clase que representa al modelo, una clase concreta de ModelicaClass: Model.java

```java
package com.fluideditor.model.modelica;

import com.fluideditor.model.icon.IconAnnotation;
import java.util.ArrayList;
import java.util.List;
import javafx.scene.shape.Line;

/**
 * Representa el modelo que se esta editando.
 *
 * @author Jackson F. Reyes Bermeo
 */
public class Model extends ModelicaClass {

    private String within;
    private final List<ModelicaParameter> elementsModelComposition;
    private final List<ModelicaConnection> connections;
    private IconAnnotation icon;

    public void clear() {
        absolutePath = null;
        this.within = null;
        elementsModelComposition.clear();
        connections.clear();
        icon = null;
    }

    public Model(String name) {
        super(name);
        elementsModelComposition = new ArrayList<>();
        connections = new ArrayList<>();
    }

    @Override
    public String getAbsolutePath() {
        return absolutePath;
    }

    @Override
    public void setAbsolutePath(String absolutePath) {
        this.absolutePath = absolutePath;
    }

    public ModelicaConnection getModelicaConnectionById(String id) {
        for (ModelicaConnection mc : connections) {
            if (mc.getId() != null && mc.getId().equals(id)) {
                return mc;
            }
        }
        return null;
```

```
51      }
52
53      public boolean removeModelicaConnectionByLineObject(Line line) {
54          ModelicaConnection mcToRemove = null;
55          boolean removed = false;
56          for (ModelicaConnection mc : connections) {
57              if (mc.getLine() == line) {
58                  mcToRemove = mc;
59              }
60          }
61          if (mcToRemove != null) {
62              removed = connections.remove(mcToRemove);
63          }
64
65          return removed;
66      }
67
68      public List<Line> removeModelicaConnectionById(String id) {
69          List<ModelicaConnection> connectionsToRemove = new ArrayList<>()
                ;
70          for (ModelicaConnection mc : connections) {
71              String idFirstConnector = mc.getFirstConnector().getId();
72              String idSecondConnector = mc.getSecondConnector().getId();
73              String idTargetFirstConnector = id + mc.getFirstConnector().
                    getType();
74              String idTargetSecondConnector = id + mc.getSecondConnector
                    ().getType();
75              if (idFirstConnector.equals(idTargetFirstConnector) ||
                    idSecondConnector.equals(idTargetSecondConnector)) {
76                  connectionsToRemove.add(mc);
77              }
78          }
79          List<Line> linesToRemove = new ArrayList<>();
80          for (ModelicaConnection mc : connectionsToRemove) {
81              linesToRemove.add(mc.getLine());
82              connections.remove(mc);
83          }
84          return linesToRemove;
85      }
86
87      public void removeElementOfModelCompositionById(String id) {
88          ModelicaParameter mp = getElementOfModelCompositionById(id);
89          if (mp != null) {
90              elementsModelComposition.remove(mp);
91          }
92
93      }
94
95      public List<ModelicaConnection> getAllConnections() {
96          return connections;
97      }
98
99      public void addConnection(ModelicaConnection connection) {
100         connections.add(connection);
101     }
102
103     public ModelicaParameter getElementOfModelCompositionById(String id)
            {
104         for (ModelicaParameter parameter : elementsModelComposition) {
105             if (parameter.getId() == null ? id == null : parameter.getId
                    ().equals(id)) {
```

```
106                 return parameter;
107             }
108         }
109         return null;
110     }
111 
112     public int existElementOfModelCompositionByName(String name) {
113         String currentName = "";
114         int count = 0;
115         for (ModelicaParameter parameter : elementsModelComposition) {
116             if (parameter.getNameComponent().toLowerCase().contains(name
                    .toLowerCase())) {
117                 currentName = parameter.getNameComponent();
118                 count++;
119             }
120         }
121 
122         int num = count;
123         if (!currentName.isEmpty() && !currentName.isBlank() && count >
                0) {
124             currentName = currentName.replace(name, "");
125             if (currentName.isEmpty()) {
126                 return 1;
127             }
128             try {
129                 num = Integer.parseInt(currentName) + 1;
130             } catch (NumberFormatException e) {
131                 num = 10;
132             }
133         }
134         return num;
135     }
136 
137     public void addElementOfModelComposition(ModelicaParameter parameter
            ) {
138         elementsModelComposition.add(parameter);
139     }
140 
141     public void removeElementOfModelComposition(ModelicaParameter
            parameter) {
142         elementsModelComposition.remove(parameter);
143     }
144 
145     public String getWithin() {
146         return within;
147     }
148 
149     public void setWithin(String within) {
150         this.within = within;
151     }
152 
153     public List<ModelicaParameter> getAllCompositions() {
154         return elementsModelComposition;
155     }
156 
157     public IconAnnotation getIcon() {
158         return icon;
159     }
160 
161     public void setIcon(IconAnnotation icon) {
162         this.icon = icon;
```

```
163     }
164
165     @Override
166     public ModelicaClass getModelicaComponent() {
167         return this;
168     }
169 }
```

Código B.28: Implementación la clase que representa al Modelo, una clase concreta de ModelicaClass.

Código de la clase que representa un componente Modelica: ComponentModel.java

```
1  package com.fluideditor.model.modelica;
2
3  /**
4   * *
5   * Representa los elementos de composición de un Model.
6   *
7   * @author Jackson F. Reyes Bermeo
8   */
9  public class ComponentModel {
10
11     private String id;
12     private String prefix;
13     private String type;
14     private String name;
15     private String value;
16     private String comment;
17     private Dialog dialog;
18
19     public String getId() {
20         return id;
21     }
22
23     public void setId(String id) {
24         this.id = id;
25     }
26
27     public String getCodeComponent() {
28         String textCode = name + "=" + value;
29         return textCode;
30     }
31
32     public String getPrefix() {
33         return prefix;
34     }
35
36     public void setPrefix(String prefix) {
37         this.prefix = prefix;
38     }
39
40     public String getType() {
41         return type;
42     }
43
44     public void setType(String type) {
```

```
45          this.type = type;
46      }
47
48      public String getName() {
49          return name;
50      }
51
52      public void setName(String name) {
53          this.name = name;
54      }
55
56      public String getValue() {
57          return value;
58      }
59
60      public void setValue(String value) {
61          this.value = value;
62      }
63
64      public String getComment() {
65          return comment;
66      }
67
68      public void setComment(String comment) {
69          this.comment = comment;
70      }
71
72      public Dialog getDialog() {
73          return dialog;
74      }
75
76      public void setDialog(Dialog dialog) {
77          this.dialog = dialog;
78      }
79
80      @Override
81      public String toString() {
82          return "id: " + id + ","
83                  + "prefix: " + prefix + ","
84                  + "type: " + type + ","
85                  + "name: " + name + ","
86                  + "value: " + value + ","
87                  + "comment: " + comment + ","
88                  + "dialog: " + dialog;
89      }
90  }
```

Código B.29: Implementación la clase que representa un componente Modelica.

Código de la clase que representa los parámetros de cada componente de Modelica: ModelicaParameter.java

```
1  package com.fluideditor.model.modelica;
2
3  import com.fluideditor.model.icon.Placement;
4  import java.util.ArrayList;
5  import java.util.HashMap;
6  import java.util.List;
7  import java.util.Map;
8  import java.util.Map.Entry;
9
10 /**
11  * Representa cada uno de los elementos que se arrastran al contenedor
        de
12  * dise o.
13  *
14  * @author Jackson F. Reyes Bermeo
15  */
16 public class ModelicaParameter {
17
18     private String id;
19     private String nameComponent;
20     private String comment;
21     private String path;
22     private String defaultComponentName;
23     private String defaultComponentPrefix;
24     private List<ComponentModel> components; // declaraciones
25     private String missingInnerMessage;
26
27     private Placement placement;
28     private Map<String, String> redifinitions;// las modificaciones que
           redefinen el componente
29
30     public ModelicaParameter() {
31         this.components = new ArrayList<>();
32         placement = new Placement();
33         redifinitions = new HashMap<>();
34     }
35
36     public String getRedefinicionStr() {
37         String textRedifinitions = "";
38         if (redifinitions != null) {
39             for (Entry<String, String> set : redifinitions.entrySet()) {
40                 if (set.getValue() != null && !set.getValue().isBlank())
                       {
41                     textRedifinitions += set.getKey() + "=" + set.
                           getValue() + ",";
42                 }
43             }
44             if (textRedifinitions.length() > 1) {
45                 textRedifinitions = "(" + textRedifinitions + ")";
46                 textRedifinitions = textRedifinitions.replace(",)", ")")
                       ;
47             }
48         }
49         return textRedifinitions;
50     }
51
52     public void addRedefiniton(String key, String value) {
```

```
53          if (!key.isBlank()) {
54              redifinitions.put(key, value);
55          }
56      }
57
58      public void setRedefinicions(Map<String, String> redefinicions) {
59          this.redifinitions = redefinicions;
60      }
61
62      public String getCodeString() {
63          if (defaultComponentPrefix == null) {
64              defaultComponentPrefix = "";
65          }
66          String textCode = "   " + defaultComponentPrefix + " "
67                  + path + " "
68                  + nameComponent
69                  + getRedefinicionStr() + " "
70                  + "annotation(" + placement.getCodeString() + ");"; //+"
                        "+ parameter.getPath()+ parameter.getNameComponent
                        ()+"\n";
71
72          return textCode;
73      }
74
75      public ComponentModel getComponentById(String id) {
76          for (ComponentModel component : components) {
77              if (component.getId().equals(id)) {
78                  return component;
79              }
80          }
81          return null;
82      }
83
84      public ComponentModel getComponentByName(String name) {
85          for (ComponentModel component : components) {
86              if (component.getName().equals(name)) {
87                  return component;
88              }
89          }
90          return null;
91      }
92
93      public Placement getPlacement() {
94          return placement;
95      }
96
97      public void setPlacement(Placement placement) {
98          this.placement = placement;
99      }
100
101     public String getId() {
102         return id;
103     }
104
105     public void setId(String id) {
106         this.id = id;
107     }
108
109     public void addComponent(ComponentModel component) {
110         //components.add(component);
111
```

```
112         //test: the name is unique
113         if(getComponentByName(component.getName()) == null){
114            components.add(component);
115         }
116
117
118     }
119
120     public List<ComponentModel> getComponents() {
121         return components;
122     }
123
124     public String getMissingInnerMessage() {
125         return missingInnerMessage;
126     }
127
128     public void setMissingInnerMessage(String missingInnerMessage) {
129         this.missingInnerMessage = missingInnerMessage;
130     }
131
132     public void setComponents(List<ComponentModel> components) {
133         this.components = components;
134     }
135
136     public String getNameComponent() {
137         return nameComponent;
138     }
139
140     public void setNameComponent(String nameComponent) {
141         this.nameComponent = nameComponent;
142     }
143
144     public String getComment() {
145         return comment;
146     }
147
148     public void setComment(String comment) {
149         this.comment = comment;
150     }
151
152     public String getPath() {
153         return path;
154     }
155
156     public void setPath(String path) {
157         this.path = path;
158     }
159
160     public String getDefaultComponentName() {
161         return defaultComponentName;
162     }
163
164     public void setDefaultComponentName(String defaultComponentName) {
165         this.defaultComponentName = defaultComponentName;
166     }
167
168     public String getDefaultComponentPrefix() {
169         return defaultComponentPrefix;
170     }
171
```

```
172     public void setDefaultComponentPrefix(String defaultComponentPrefix)
            {
173         this.defaultComponentPrefix = defaultComponentPrefix;
174     }
175 }
```

Código B.30: Implementación la clase que representa los parámetros de cada componente de Modelica.

Código de la clase que representa cada uno de los paneles de los parámetros de los componentes Modelica: Dialog.java

```
1  package com.fluideditor.model.modelica;
2
3  /**
4   * Representa el Dialog de Modelica para crear la ventana de parametros.
5   *
6   * @author Jackson F. Reyes Bermeo
7   */
8  public class Dialog {
9
10     private String tab = "General";
11     private String group = "Parameters";
12     private boolean enable = true;
13     private boolean ShowStartAttribute = false;
14     private boolean colorSelector = false;
15     private String groupImage = "";
16     private boolean connectorSizing = false;
17
18     public String getTab() {
19         return tab;
20     }
21
22     public void setTab(String tab) {
23         this.tab = tab;
24     }
25
26     public String getGroup() {
27         return group;
28     }
29
30     public void setGroup(String group) {
31         this.group = group;
32     }
33
34     public boolean isEnable() {
35         return enable;
36     }
37
38     public void setEnable(boolean enable) {
39         this.enable = enable;
40     }
41
42     public boolean isShowStartAttribute() {
43         return ShowStartAttribute;
44     }
45
46     public void setShowStartAttribute(boolean ShowStartAttribute) {
```

```
47          this.ShowStartAttribute = ShowStartAttribute;
48      }
49
50      public boolean isColorSelector() {
51          return colorSelector;
52      }
53
54      public void setColorSelector(boolean colorSelector) {
55          this.colorSelector = colorSelector;
56      }
57
58      public String getGroupImage() {
59          return groupImage;
60      }
61
62      public void setGroupImage(String groupImage) {
63          this.groupImage = groupImage;
64      }
65
66      public boolean isConnectorSizing() {
67          return connectorSizing;
68      }
69
70      public void setConnectorSizing(boolean connectorSizing) {
71          this.connectorSizing = connectorSizing;
72      }
73  }
```

Código B.31: Implementación la clase que representa a cada uno de los paneles de visualización de los parámetros en cada componentes Modelica.

Código de la clase que guarda la información del conector: ModelicaConnector.java

```
1   package com.fluideditor.model.modelica;
2
3   /**
4    * Representa a un conector de Modelica.
5    * @author Jackson F. Reyes Bermeo
6    */
7   public class ModelicaConnector {
8
9       private String id;
10      private String prefix;
11      private String parent;
12      private String type;
13      private String name;
14      private String value;
15      private String comment;
16      private String redeclaration;
17      private boolean connected = false;
18      private boolean isArray;
19      private String indexName;
20      private int indexArray = 0;
21
22      public String getIndexName() {
23          return indexName;
24      }
```

```
25
26      public void setIndexName(String indexName) {
27          this.indexName = indexName;
28      }
29
30      public String getParent() {
31          return parent;
32      }
33
34      public void setParent(String parent) {
35          this.parent = parent;
36      }
37
38      public String getRedeclaration() {
39          return redeclaration;
40      }
41
42      public void setRedeclaration(String redeclaration) {
43          this.redeclaration = redeclaration;
44      }
45
46      public String getPrefix() {
47          return prefix;
48      }
49
50      public void setPrefix(String prefix) {
51          this.prefix = prefix;
52      }
53
54      public String getType() {
55          return type;
56      }
57
58      public void setType(String type) {
59          this.type = type;
60      }
61
62      public String getName() {
63          String nameTemp = this.name;
64          if (isArray) {
65              nameTemp = nameTemp.replace(indexName, "" + indexArray);
66          }
67          return nameTemp;
68      }
69
70      public void setName(String name) {
71          this.name = name;
72      }
73
74      public String getValue() {
75          return value;
76      }
77
78      public void setValue(String value) {
79          this.value = value;
80      }
81
82      public String getComment() {
83          return comment;
84      }
85
```

```
86      public void setComment(String comment) {
87          this.comment = comment;
88      }
89
90      public boolean isIsArray() {
91          return isArray;
92      }
93
94      public void setIsArray(boolean isArray) {
95          this.isArray = isArray;
96      }
97
98      public int getIndexArray() {
99          return indexArray;
100     }
101
102     public void setIndexArray(int indexArray) {
103         this.indexArray = indexArray;
104     }
105
106     public String getId() {
107         return id;
108     }
109
110     public void setId(String id) {
111         this.id = id;
112     }
113
114     public boolean isConnected() {
115         return connected;
116     }
117
118     public void setConnected(boolean connected) {
119         this.connected = connected;
120     }
121
122     @Override
123     public String toString() {
124         return "Type: " + type + ",name: " + name + "," + ",prefix: " +
                prefix;
125     }
126 }
```

Código B.32: Implementación la clase que guarda la información del conector.

Código de la clase que representa la conexión entre dos conectores: ModelicaConnection.java

```
1  package com.fluideditor.model.modelica;
2
3  import com.fluideditor.model.icon.LineAnnotation;
4  import javafx.scene.shape.Line;
5
6  /**
7   * Representa la conexión que se establece entre dos conectores.
8   *
9   * @author Jackson F. Reyes Bermeo
10  */
11 public class ModelicaConnection {
12
13     private String id;
14     private ModelicaConnector firstConnector;
15     private ModelicaConnector secondConnector;
16     private LineAnnotation lineConnection;
17     private Line line;
18
19     public ModelicaConnection(Line line) {
20         this.line = line;
21     }
22
23     public LineAnnotation getLineConnection() {
24         return lineConnection;
25     }
26
27     public Line getLine() {
28         return line;
29     }
30
31     public void setLineConnection(LineAnnotation lineConnection) {
32         this.lineConnection = lineConnection;
33         firstConnector = new ModelicaConnector();
34         secondConnector = new ModelicaConnector();
35     }
36
37     public String getCodeString() {
38         String textCode = "";
39         if (firstConnector != null && secondConnector != null) {
40             String firstConnectorName = firstConnector.getName().
                   replaceAll("\\(.*\\)", "").replaceAll("=\\s?.*|\\s?\\(.*
                   ", "");
41             String secondConnectorName = secondConnector.getName().
                   replaceAll("\\(.*\\)", "").replaceAll("=\\s?.*|\\s?\\(.*
                   ", "");
42             textCode += " connect("
43                     + firstConnector.getParent() + "." +
                           firstConnectorName + ","
44                     + secondConnector.getParent() + "." +
                           secondConnectorName
45                     + ") ";
46             textCode += "annotation(";
47             if (lineConnection != null) {
48                 textCode += lineConnection.getCodeString();
49             }
50             textCode += ");";
51         }
```

```
52          return textCode;
53      }
54
55      public String getId() {
56          return id;
57      }
58
59      public void setId(String id) {
60          this.id = id;
61      }
62
63      public ModelicaConnector getFirstConnector() {
64          return firstConnector;
65      }
66
67      public void setFirstConnector(ModelicaConnector firstConnector) {
68          this.firstConnector = firstConnector;
69      }
70
71      public ModelicaConnector getSecondConnector() {
72          return secondConnector;
73      }
74
75      public void setSecondConnector(ModelicaConnector secondConnector) {
76          this.secondConnector = secondConnector;
77      }
78  }
```

Código B.33: Implementación la clase que representa la conexión entre dos conectores.