Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Ventajas de la Co-Simulación y símil eléctrico en simuladores virtuales

Autor: Sara Coca Guerrero

Tutor: María de los Ángeles Martín Prats

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Ventajas de la Co-Simulación y símil eléctrico en simuladores virtuales

Autor:
Sara Coca Guerrero

Tutor:
María de los Ángeles Martín Prats
Profesor Titular

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Trabajo Fin de Grado: Ventajas de la Co-Simulación y símil eléctrico en simuladores virtuales

Autor: Sara Coca Guerrero
Tutor: María de los Ángeles Martín Prats

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

En primer lugar, me gustaría agradecer a mi familia, amigos y a mi pareja por haberme apoyado incondicionalmente estos años de carrera. También me gustaría agradecer a todos los profesores que han hecho que disfrute más todas las clases, gracias a ellos he aprendido muchísimo y he podido descubrir lo que realmente me gusta dentro de este mundo. Además, quiero agradecer a Víctor Murillo, por todo el apoyo, la dedicación y la ayuda recibida. Por último, agradecer a mi tutora, María de los Ángeles Martín Prats, por confiar en mi desde el principio y darme las oportunidades que me ha dado, además de su dedicación y todo lo que he podido aprender de ella.

Sara Coca Guerrero
Estudiante del Grado de Ingeniería Aeroespacial

Sevilla, 2023

Resumen

La Co-Simulación y el símil eléctrico, son dos técnicas con múltiples ventajas en varios sectores, y sobre todo en el sector aeroespacial. El objetivo de este trabajo es analizar diferentes simulaciones y Co-simulaciones en las que se demuestren las ventajas que tienen para la industria. Lo que se pretende es ver qué plataformas reducen el tiempo de computación, el tamaño de los archivos y la implementación de los sistemas, para así tener una mejor Co-simulación. Asimismo, se comprobará cómo mediante la Co-Simulación es posible simular un sistema divido en partes, y generado desde plataformas distintas, sin ningún tipo de problema. Otro de los objetivos, es reducir el tiempo de simulación y la dificultad de implementación de sistemas hidráulicos a través del uso de una analogía electro-hidráulica. Para ello se van a utilizar tres programas de modelado y simulación diferentes, con los que, a partir del estándar FMI (Functional Mock-Up Interface), se generaran archivos para realizar Co-Simulaciones en Python, a través de varios test unitarios. Analizando de los resultados obtenidos de las simulaciones y los test unitarios, además de las comparaciones entra las diferentes plataformas, simulaciones y archivos exportados, se contrastarán las ventajas y la utilidad de la Co-Simulación y el símil eléctrico en la industria aeroespacial.

Abstract

Co-Simulation and electric simile are two techniques with multiple advantages, especially in the aerospace sector. This work aims to test these two techniques, verifying the advantages that each of them offers to the industry. One of the objectives, is to see which platforms reduce computing time, file sizes and system implementation, in order to have a better Co-simulation. In addition, it will be verified how, through Co-Simulation, a system divided into parts, and generated from different platforms, can be simulated without any type of problem. Another objective, is to reduce the simulation's time and the difficulty of implementing hydraulic systems through the use of an electro-hydraulic analogy. To do this, different modeling and simulation programs will be used, with which files will be exported, using the FMI standard (Functional Mock-Up Interface), to perform the Co-Simulations in Python using unit tests. Based on the results obtained from the simulations and the unit tests, in addition to the comparisons between the different platforms, simulations and exported files, the advantages and usefulness of the CoSimulation and electrical simile in the aerospace industry will be confirmed.

Índice Abreviado

Índice

1 Introducción

1.1 Motivación

En este trabajo se pretende mostrar las ventajas de la Co-simulación y del símil eléctrico. En la industria aeronáutica actualmente se siguen dos tendencias; conseguir un avión más eléctrico y utilizar la digitalización para tener un mayor nivel de seguridad en las aeronaves.

Al adoptar la configuración de MEA (*More electric Aircraft*) lo que se pretende es reemplazar los sistemas hidráulicos por eléctricos. Así se puede reducir el coste, el peso, la complejidad. Además, se pretende arrancar eléctricamente el motor eliminando así los engranajes aunque reduce la potencia del arranque, especialmente en condiciones de frío. Otro de los factores que implica la configuración MEA, es integrar un Advanced Magnetic Bearing en el arrancador del motor principal para tener unidades de potencia auxiliar. Estos son sólo algunos de los cambios en los sistemas de una aeronave para conseguir que el avión sea eléctricos. En resumen, el MEA ha revolucionado la industria aeroespacial mejorando el peso en vacío de la aeronave, el coste, el mantenimiento, el soporte y la fiabilidad que el sistema puede lograr. [8]

En el sector aeroespacial la digitalización de la producción hace referencia al uso y coordinación de la información, automatización, sensores y software para así tener una nueva perspectiva del trabajo colaborativo entre diferentes empresas. Los software y programas proporcionan un medio a través del cuál los sistemas pueden funcionar entre sí, pudiendo así realizar tareas y procesos con relaciones intra e interempresariales en la cadena de producción. La industria aeroespacial es líder en la implementación de nuevas tecnologías, procesos y metodologías. Como consecuencia de este liderazgo existe una gran competencia entre las diferentes empresas del sector, lo que hace que los gobiernos y dichas compañías adopten propuestas innovadoras con el fin de tener una mayor seguridad, reducir tiempos y costes y tener así una mayor consciencia de la contaminación. A pesar de esta competitividad, los obstáculos de la producción aeroespacial, la cual es compleja y muy distribuida geográficamente, hace que haya más colaboración y comunicación entre diferentes empresas y gobiernos. La NASA (*National Aeronautics and Space Administration*) y la Comisión Europea, mediante instancias y programas como Marco, siguen con la búsqueda de nuevos procesos para el sector aeronáutico. [11]

Por tanto adoptando la configuración de un avión MEA y con una mayor digitalización se consigue mejorar varios aspectos en el sector aeroespacial, como el coste, la reducción del peso, una mayor comunicación o mayor seguridad de las aeronaves y sistemas.

La Co-Simulación contribuye a la digitalización, ya que debido a la complejidad de los sistemas embarcados las tareas de diseño y producción se suelen dividir entre varios equipos de diferentes empresas; que pueden usar software distintos. Este método hace posible que los diseños de dichos equipos puedan implementarse conjuntamente a pesar de no estar diseñados con el mismo programa. Por tanto, la Co-Simulación supone un gran avance en la comunicación entre las empresas del sector aeroespacial, ayudando así a crear nuevos sistemas más complejos que aumenten el nivel de seguridad de la aeronave. Además hay que añadir el desarrollo de la Inteligencia Artificial, que puede ayudar al piloto a tomar decisiones en situaciones complicadas de manejar o inusuales. Asimismo, conlleva una mejora para la formación de los pilotos, al poder conseguir un aprendizaje en un aula con un simulador igual que la cabina que van a utilizar cuando trabajen, lo cual es un aspecto que mejora indirectamente la seguridad de las operaciones.

El símil eléctrico, al igual que la Co-Simulación, contribuye de forma positiva en el sector aeroespacial. En especial, porque reduce los tiempos de análisis de sistemas y por tanto de su producción. Consiguiendo así que los circuitos neumáticos e hidráulicos se puedan estudiar de una forma más sencilla, menos costosa y con un rendimiento mayor como se verá a lo largo del documento.

Para verificar realmente todas las ventajas que ofrecen la Co-Simulación y el símil eléctrico en el sector aeroespacial se va a realizar el análisis de diferentes sistemas, diseñados mediante programas distintos y exportados mediante el estándar FMI (*Functional Mock-up Interface*). A partir de la comparación y el análisis de los diferentes FMUs (*Functional Mock-up Unit*) exportados, se obtendrán resultados para verificar las ventajas de la Co-Simulación y el símil eléctrico. Para verificar el correcto funcionamiento de dichos modelos, se someterán a test unitarios individuales cada sistema. Así, se comprobará si todos pueden dar la misma solución ante un mismo código, teniendo en cuenta que estos sistemas se han creado en programas distintos.

En resumen, los objetivos de este trabajo son: analizar las diferentes herramientas de modelado y simulación, para deducir cuál es la que genera los archivos FMU con menor tamaño, y la que reduce más la carga computacional, para que la Co-Simulación se optimice. Además se quiere demostrar cómo, a partir de la Co-Simulación se pueden unir diferentes partes de un mismo sistema, creadas desde plataformas distintas. Asimismo, se analizará cómo el símil es más sencillo de implementar, que los sistema a los que emula, y que además, es capaz de reducir el tiempo de simulación y el tamaño del archivo generado.

En la industria aeroespacial, estas dos técnicas son fundamentales. Esto es debido a que, uno de los principales objetivos del sector es aumentar el nivel de seguridad de las aeronaves. Para ello se replican diferentes circuitos y sistemas, ya sea con una réplica exacta o utilizando lógica inversa. Al introducir ese nivel de redundancia en los equipos, los circuitos son más complejos y por tanto, es mucho más sencillo dividirlo en diferentes partes para poder desarrollarlos a nivel de software, esta la utilidad de la Co-Simulación. Además el símil eléctrico es muy útil para emular algunos de los sensores de sistemas hidráulicos y neumáticos de la aeronave.

Para la realización de este trabajo, se han utilizado tres herramientas de modelado y simulación, las cuales son Simulink, Open Modelica y Hopsan. Siendo Open Modelica y Hopsan dos herramientas que no se han utilizado a lo largo del grado de ingeniería aeroespacial. Para los test unitarios se ha optado por programarlos en Python. Asimismo, se ha hecho uso de la herramienta Boole Deusto, para simplificar funciones "*booleanas*".

Además se han estudiado las diferentes metodologías de trabajo, para llevar a cabo la que se adaptara mejor al proyecto. Las diferentes metodologías de trabajo estudiadas, así como la herramienta

Notion, utilizada para llevar a cabo un método de trabajo ágil, se presentan en el apéndice A.

A continuación se va a definir detalladamente los conceptos más importantes para que se comprenda mejor la motivación y realización del trabajo.

1.1.1 Co-Simulación

En la actualidad las técnicas de simulación se han ido extendiendo a lo largo del mundo, tanto en la industria como en el ámbito científico. Generalmente cuando se quiere modelar y simular un sistema complejo se divide en partes. Esas divisiones normalmente las hacen diferentes equipos y expertos, y puede que estén modeladas mediante diferentes técnicas, herramientas, algoritmos o programas. Los modelos no son fáciles de integrar, debido a que se han realizado con métodos especializados y totalmente diferentes. Además, si hay partes que se proporcionan externamente, puede que tengan propiedad intelectual. Por tanto, hace que el producto sea más caro, porque hay que saber como poder usarlo. Otro de los aspectos a tener en cuenta, es que hay que integrar hardware, con software y operadores humanos que sepan como hacer funcionar todas las partes del sistema conjuntamente. Todos estos factores son los que hacen que la simulación de ese sistema más grande sea difícil.

La Co-Simulación surge para hacer frente a ese problema, Es una técnica que permite realizar simulaciones del sistema de estudio, mediante la composición de las partes en las que se ha dividido, independientemente de cómo se han creado esas partes. Estas divisiones del sistema global se van a definir como *Cajas Negras*, es decir, pueden mostrar su comportamiento a partir de unas entradas que les de el operador, y produciendo unas salidas adecuadas a la operación a la que está destinando ese subsistema.

La Co-simulación es muy útil para entrenar a las personas que tienen que hacer operaciones complicadas en su trabajo, acciones que no se podrían hacer directamente sin haber probado y entendido cómo funcionan los diferentes sistemas que necesitan manejar. Por ejemplo, se usa para entrenar a los pilotos de aviones y conseguir así, que conozcan todos los sistemas del avión antes de manejar uno. También puede ser útil a la hora de desarrollar nuevas partes del sistema e integrarlas con las que ya había sin modificar las demás. Otra de las razones por las que se usa la Co-Simulación, es porque a partir de información detectada por sensores en una operación normal del sistema, se puede predecir, mediante simulaciones del mismo, los posibles fallos y cuando se pueden dar, lo que es un aspecto muy importante para las operaciones de mantenimiento. [21].

En este trabajo se va realizar la técnica de Co-Simulación a varios modelos que se presentaran a lo largo de este documento. Estos ejemplos se han creado en tres programas distintos, y se han exportado mediante el *estándar FMI*. Además se les han realizado varios test unitarios, para cada tipo de sistema, comprobando así las ventajas de esta técnica.

1.1.2 Analogía electro hidráulica: Símil Eléctrico

Una analogía hace referencia a la relación que existe entre dos cosas distintas. En este documento se va a ver la relación entre los circuitos eléctricos e hidráulicos, y se va demostrar que en algunos casos es más sencillo, y se obtiene un mayor rendimiento, usando circuitos eléctricos en lugar de circuitos hidráulicos.

Muchos autores ya han usado esta analogía para poder mostrar los efectos hidráulicos en circuitos eléctricos. Uno de los motivos por los que se da esta relación, es que los algunos dispositivos

eléctricos y electrónicos tienen la capacidad de almacenar energía, que se puede asimilar a tanques de agua en el ámbito hidráulico, como son los condensadores. [26] y [18]

En este trabajo se pretenden comparar dos simulaciones, a las que se le puede aplicar la analogía electro hidráulica. El llenado y vaciado de un tanque de agua a través de una válvula controlable y la carga y descarga del condensador controlando el voltaje del mismo.

Conceptos relacionados con el símil eléctrico

Se van a introducir una serie de conceptos que se van a nombrar en la realización del trabajo. Todos los conceptos tienen relación con la simulación, en concreto con el símil eléctrico.

La definición estricta de **emulación** es el "deseo intenso de imitar e incluso superar las acciones ajenas" [27]. Este concepto es interesante, porque lo que se pretende hacer en este caso, con la analogía electro hidráulica, es imitar mediante un circuito eléctrico la acción de un sensor de llenado de un tanque. Y no solo se quiere imitar, sino que con este circuito se pretende mejorar el rendimiento de la simulación. Un emulador trata de modelar de forma precisa, de manera que funcione como si fuera el sistema original al que trata de imitar.

Otro concepto que puede ser de interés en el trabajo, es **modelo subrogado**, que en simulación se utiliza cuando el resultado que se quiere analizar no se puede medir directamente, porque para obtenerlo se necesita mucho tiempo de simulación [25]. Es decir, la función de un modelo subrogado en simulación, consiste en reproducir el comportamiento del modelo original pero reduciendo el tiempo de procesamiento, el tiempo de simulación. [15].

Con el símil eléctrico que se va a estudiar en este trabajo ,se ha querido realizar una **simulación en tiempo real**, lo cual implica que interactúa con su entorno. Si las variables de entrada cambian durante la simulación, las de salida también lo van a hacer. Así, se reflejan los cambios producidos por el usuario en el programa. Lo que se hace es imponer una serie de condiciones en la simulación que se deben cumplir, y estas condiciones se van a ajustar al comportamiento del sistema que se desee, para parecerse al sistema físico.[14]

1.2 Estado del Arte

Para poder entender los programas usados para la realización de este trabajo, además de explicar el porqué se han utilizado los mismos, se van a explicar cada uno de ellos en esta sección.

1.2.1 Estándar FMI y programas de simulación

El *Functional Mock-up Interface* (FMI) es un estándar abierto que da la posibilidad de modelar y simular para el intercambio y Co-Simulación de modelos dinámicos. El objetivo que tiene, es proporcionar las herramientas adecuadas para que los distintos fabricantes y equipos encargados de un sistema, puedan intercambiar los modelos sin importar que las herramientas usadas, para la creación de los mismos, hayan sido distintas. Este estándar tiene dos usos principalmente:

- **Intercambio de modelos.** Consiste en generar un código de un modelo de una herramienta cualquiera, que resuelva numéricamente los sistemas que componen el modelo, para que se pueda utilizar en otra herramienta. El código del modelo que se genera, es un archivo comprimido con una extensión "*.zip*", pero en este caso la extensión se denomina "*.fmu*" (Functional Mock-up Unit).

- **Co-Simulacion.** Facilita una interfaz capaz de conectar uno o más programas de modelado y simulación. La información se da de manera síncrona y los modelos se simulan individualmente entre intervalos de comunicación. Hay que tener en cuenta que el modelo y el código se tienen que exportar conjuntamente para poder realizar la Co-Simulación adecuadamente.

Como resumen, el estándar FMI genera un archivo FMU mediante la exportación de un modelo, creado por una herramienta de simulación cualquiera. Tras esto el objetivo es importarla en un entorno cerrado para poder usarla.

Para abrir los archivos FMU (con extensión "*.fmu*"), se tiene que renombrar la extensión por "*.zip*", para poder así, tratarlo como si fuera un archivo comprimido. Dentro del mismo se pueden encontrar varias carpetas y documentos:

- Un archivo con extensión "*.xml*" denominado "*modelDescription.xml*". Al abrir este archivo se puede ver información relevante del FMU. En él se indica el la versión del estándar FMI usada para la exportación, el nombre del modelo, la herramienta con la que se creó, la fecha y la hora a la que se exportó, y más o menos información dependiendo de la herramienta con la que se generó el modelo. Uno de los aspectos más importantes de este archivo es la descripción de las variables del sistema exportado, indicando su nombre, su valor inicial y el tipo de variable...
- Para el intercambio de modelos se genera un código en lenguaje C que contiene las funciones de las ecuaciones necesarias para la simulación del modelo. Estas funciones se pueden encontrar en código fuente o en código binario para distintas plataformas. Por tanto, en el "*.zip*" se va a encontrar una carpeta llamada "*binaries*".
- Dependiendo de la herramienta desde la que se exportó el archivo FMU se han podido añadir otros elementos como una foto del modelo, documentación, datos, fuentes necesarias para generarlo...

Algunas de las extensiones de los diferentes archivos incluidos en dichas carpetas se describen brevemente a continuación, ya que, más adelante se mencionaran a fin de comparar los diferentes FMU generados.

- ".xml" : son archivos destinados a almacenar datos de tal forma que las personas los puedan leer y entender. Además, a nivel de ordenador también permite que se puedan leer y procesar, para así poder utilizarlos.[3]
- ".dll" : son archivos contenedores, guardan todos los recursos que usa el programa, es decir, contienen código ejecutable y las partes del software al que pertenecen. Si este archivo se corrompe el programa no se puede ejecutar. [6]
- ".h" : los denominados "archivos de cabecera" por los programadores. Generalmente son los archivos que almacenan las definiciones de las variables, constantes y funciones del sistema.[17]
- ".c" : código en C o C++.
- ".log" : son registros que guardan todos los eventos que pueden afectar a la ejecución de un programa. Proporciona una seguridad del comportamiento del sistema.[31]
- ".json": son muy similares a los ".xml", aunque existen algunas diferencias entre ellos, como por ejemplo, el lenguaje, el de los documentos con extensión ".json" es más sencillo y en los ".xml" son más personalizados. Aunque la principal diferencia está en que los archivos ".json" solo pueden almacenar datos numéricos y textos, mientras que los ".xml" almacenan también imágenes y gráficas.[12]

- ".lib": archivo tipo biblioteca que almacena información sobre objetos y datos de la exportación usada en otros programas.[16]

Todas las extensiones explicadas se van a poder encontrar en los diferentes FMU, aunque puede que no estén todas, va a depender del programa a partir del cual se ha generado el mismo.

Retomando el estándar FMI, hay que destacar que existen varias versiones del mismo, que se han ido creando a lo largo de los años. Se pueden encontrar tres versiones, FMI 1.0, FMI 2.0 y FMI 3.0 (la más actual). Para la realización de este trabajo se ha usado la versión FMI 2.0, ya que las herramientas con las que se han creado los modelos solo permitían la exportación mediante estándar FMI con las versiones 1.0 y 2.0.

No todas las herramientas disponibles para crear modelos de simulación son compatibles con el estándar FMI. Los software desde los que se permite exportar modelos mediante el estándar FMI se pueden ver en [4]

En este proyecto se han usado varias herramientas, ya sean de coste como *open source*. Antes de seleccionar los programas con los que se va a trabajar, se muestra un análisis de varias herramientas compatibles con el estándar FMI.

En las tablas que se presentan a continuación se muestra un análisis desarrollado de cinco herramientas de coste compatibles con el estándar FMI. Se han tenido en cuenta varios factores para hacer este análisis, como pueden ser, el coste, el tipo de modelos que se pueden crear a través de ella, si hay o no documentación que ayude a aprender cómo usar dicha herramienta, las versiones de FMI que usa...

En la Tabla 1.1 se desarrolla el análisis de Simulink de Matlab. Una de las ventajas, que se muestra en la Tabla 1.1, es que permite hacer una gran variedad de modelos de distintos dominios pudiendo conectar unos bloques con otros. Además de la gran cantidad de documentación. [23]

Tabla 1.1 Información de Simulink.

Matlab Simulink by Mathworks	
Descripción	Es un entorno de diagrama de bloques para la simulación de varios dominios. Proporciona diseño a nivel de sistema ,simulación, generación automática de código y prueba y verificación continua de los sistemas.
Coste	**Alumno de la US**: gratuito **Estudiante**: 69€ al año. **Estandar**: 860€ al año. **Empresas**: 3500€ al año (Fácil de encontrar en la web oficial)
Tipo de modelos que puede generar	Modelos de muchos dominios, como son de automática, eléctrico, electrónico. Además permite que se conecten unos con otros mediante bloques especiales.
Existencia de documentación y tutoriales	Gran variedad de tutoriales, webs explicativas y documentación. La web de Mathworks incluye una ayuda con ejemplos de cada bloque que tiene Simulink, además de los comandos de Matlab y las opciones de los dos.
Plataformas en las que se puede usar	Windows, macOs, Linux.
Versión de FMI	1.0 y 2.0
Importación y exportación de FMI	Importa y exporta modelos ME (*model exchange*) y CS (Co-Simulación).
Tipo de entorno	Es un programa muy intuitivo, ya que se despliega un panel de todos los componentes que hay ordenados por secciones al pulsar el botón de librería. Los bloques tienen bastantes detalles y la interfaz es buena.

En la Tabla 1.2 se analiza la herramienta Activate de Altair. Este software presenta muy buenas prestaciones, aunque el plan gratuito no tenga tantas opciones como el plan de coste.[9]

Tabla 1.2 Información de Activate.

Activate by Altair	
Descripción	Entorno software de modelado simulación y análisis de sistemas multi-disciplinarios.
Coste	**Empresas**: No es público, solo se conoce si se quiere contratar. **Personal**: gratuito. Pero el plan personal no consta de la mayoría las opciones.(Complicado de encontrar en la web).
Tipo de modelos que puede generar	Elementos de varios dominios, como pueden ser eléctrico, mecánico, hidráulico, térmico. Estos bloques que se pueden conectar entre sí .
Existencia de documentación y tutoriales	Existen varios vídeos explicando la gran variedad de opciones que tiene la herramienta, además de varias páginas web y documentación. En comparación con las demás herramientas, la documentación y páginas web son una menor cantidad. En la página oficial se encuentran varios webinars.
Plataformas en las que se puede usar	Windows y Linux.
Versión de FMI	1.0, 2.0 y 3.0
Importación y exportación de FMI	Importa y exporta modelos ME y CS.
Tipo de entorno	Es una herramienta con la que es fácil aprender a crear distintos modelos. Cuenta con un panel en el que se buscan los componentes por su nombre, lo que facilita la implementación de los sistemas. La interfaz es bastante buena, se distinguen claramente los elementos (que son parecidos a los de Simulink).

En la Tabla 1.3 se presenta la información obtenida de Dymola. Cabe destacar que además de la información que se presenta en dicha tabla, hay una prueba gratuita de la herramienta de manera online, que se ha utilizado para valorar mejor Dymola.[29]

Tabla 1.3 Información de Dymola.

Dymola by Dassault Systèmes	
Descripción	Es una herramienta completa de modelado y simulación de sistemas complejos e integrados.
Coste	No es público, sólo se conce si se quiere contratar.
Tipo de modelos que puede generar	Componentes de dominios como automoción, aeroespacial, robótica, procesado, y otros elementos de otros dominios.
Existencia de documentación y tutoriales	Hay menos cantidad de páginas web que explican como manejar Dymola. Además es la herramienta que tiene menos vídeos tutoriales. Aunque el la página web oficial existen dos webinars.
Plataformas en las que se puede usar	Windows y Linux
Versión de FMI	1.0, 2.0 y 3.0
Importación y exportación de FMI	Importa y exporta modelos ME y CS
Tipo de entorno	Es un programa intuitivo, ya que como los dos anteriores tiene un panel con todas las librerías y los componentes ordenados por categoría, donde se pueden encontrar buscándolos por su nombre.

En la Tabla 1.4 se muestra la información obtenida por de la herramienta de modelado y simulación Maplesoft. Tiene muy buenas prestaciones, además de la cantidad de documentación y vídeos al que se tiene acceso. El inconveniente es el precio, aunque en comparación con los otros programas presentados anteriormente sea menor. [22]

Tabla 1.4 Información de MapleSim.

MapleSim by Maplesoft	
Descripción	Es una plataforma de modelado y simulación con un alto rendimiento que pretende reducir el tiempo de modelado y análisis del sistema.
Coste	**Estudiante**: 100€ al año. **Empresas**: entre 1500€ y 6070€ al año (Fácil de encontrar en la web oficial).
Tipo de modelos que puede generar	Se pueden crear modelos de diferentes dominios y combinar los distintos componentes entre sí.
Existencia de documentación y tutoriales	Gran cantidad de vídeos en la propia página oficial, además de los vídeos tutoriales de Maplesoft. A parte de esos vídeos no se encuentra mucho más, aunque en varias páginas web explican cómo se usa el entorno de simulación. Asimismo, la página web oficial se encuentra, en "*resources*", una gran cantidad de documentación.
Plataformas en las que se puede usar	Windows, macOs, Linux
Versión de FMI	1.0 y 2.0
Importación y exportación de FMI	Importa y exporta modelos ME y CS
Tipo de entorno	Al igual que los otros tres anteriores, tiene el panel para poder buscar los componentes, que están divididos por categorías. Hay que destacar que además de ese panel a la derecha de la pantalla se abre otro panel que permite ir cambiando las propiedades de los bloques que se van usando sin necesidad de clicar dos veces. Tiene funciones complejas que ya no son tan intuitivas.

Por último se presenta el programa de simulación Amesim de Siemens en la Tabla 1.5. Es un entorno de simulación que permite relacionar distintos archivos generados por el estándar FMI, así como diseños un poco más complejos de diferentes sectores. [28]

Tabla 1.5 Información de Amesim.

Amesim by Siemens	
Descripción	Entorno de simulación de sistemas mecatrónicos integrados y escalables, que permite a los ingenieros de diseño evaluar y optimizar virtualmente el rendimiento del sistema. Además de poder crear diferentes modelos de simulación admite modelos de orden reducido (ROM) y la simulación de interacción entre diferentes archivos fmu.
Coste	No es público. Se conoce si se quiere contratar el servicio.
Tipo de modelos que puede generar	Permite crear modelos eléctricos, hidráulicos y neumáticos, propulsivos, de orden reducido, térmicos...
Existencia de documentación y tutoriales	Tiene varios tutoriales tanto en la web oficial como en Youtube, además de un manual de uso completo.
Plataformas en las que se puede usar	Windows y Linux.
Versión de FMI	1.0, 2.0 y 3.0
Importación y exportación de FMI	Importa y exporta modelos ME y CS
Tipo de entorno	Es intuitivo aunque, tiene varias funciones más complicadas y específicas en el diseño de diferentes modelos. En relación con la creación de un modelo compuesto por varios archivos generados por el estándar FMI, es muy sencillo de entender cómo conectar un bloque FMU con otro, aunque la denominación de las salidas y entradas de los diferentes bloques puede ser un poco confusa al principio.

Para evaluar el tipo de entorno de cada herramienta, se han utilizado las diferentes pruebas gratuitas que presentaban los programas.

Tras haber analizado la información de todos los software de modelado y simulación, se escogió usar Simulink de MathWork, al ser gratuito para los alumnos de la Universidad de Sevilla y se utiliza en varias asignaturas durante el Grado de Ingeniería Aeroespacial.

Seguidamente se va a presentar, de la misma manera que se mostró la información de las herramientas de coste compatibles con el Estándar FMI, los programas *open source* que se analizaron para escoger varios de ellos.

En la Tabla 1.6 se detalla la información del software AutoFOCUS3. Se probó el programa para ver cómo funcionaba, al comprobar que era complicado de utilizar, además de no ser gráficamente intuitivo para los modelos que se presenten estudiar en este trabajo, se descartó su uso.[19]

Tabla 1.6 Información de AutoFOCUS3.

AutoFOCUS3 by fortiss	
Descripción	Herramienta basada en modelos y en una plataforma de investigación de sistemas integrados críticos. Se basa en *Eclipse*, un código abierto con licencia *Apache 2.0*, que se vuelve a lanzar actualizado cada dos años, para todas las plataformas en la que se puede descargar.
Tipo de modelos que puede generar	Se basa en bloques de componentes con entradas y salidas que define el usuario. Dentro de los bloques de componentes, hay otros bloques de otros componentes con estas entradas y salidas. Es decir se van generando bloques anidados unos dentro de otros hasta que se llega a los últimos bloques, que son definidos mediante ecuaciones escritas en un lenguaje de programación.
Existencia de documentación y tutoriales	Al ejecutar el programa se abre una ventana de "*welcome*", al clicar se ofrece documentación y vídeos tutoriales que los carga en la propia aplicación. Hay varias páginas web que proporcionan información, pero no demasiadas. En la página oficial se puede encontrar bastante documentación.
Plataformas en las que se puede usar	Windows, macOs (con una rutina de Java), Linux.
Versión de FMI	2.0
Importación y exportación de FMI	Importa CS y ME pero solo exporta CS.
Tipo de entorno	Es ordenado porque se va creando de lo más general hasta los detalles, teniendo unos bloques dentro de otros, como las carpetas en un ordenador. Sin embargo no es intuitivo, es complicado de aprender.

En la Tabla 1.7 se muestra la información referida a la herramienta Hopsan. Cabe destacar alguna información adicional, como que es el único software que se basa en la técnica de modelado de líneas de transmisión (TLM). Consiguiendo así, que los modelos se dividan automáticamente y se pueden introducir retrasos de tiempo motivados físicamente entre los componentes. Cada parte del modelo se resuelve independientemente durante cada paso de tiempo, mejorando así el rendimiento de la simulación. [30]

Tabla 1.7 Información de Hopsan.

Hopsan by LIU	
Descripción	Es una herramienta de simulación de sistemas multidominio y modelado de código abierto desarrollada en la Universidad de Linköping, en Suecia. Fue la primera herramienta con un soporte integrado para simulaciones multi-núlcleo.
Tipo de modelos que puede generar	Se pueden hacer varios tipos de modelos. Proporciona componentes de diferentes dominios como son neumática, eléctrica, hidráulica, de señal, etc. El problema componentes según el puerto que tengan se podrán o no conectar entre sí.
Existencia de documentación y tutoriales	Hay un manual de usuario completo, en el que dan información sobre todas las opciones de la herramienta, así como de los bloques que hay y las conexiones que se pueden hacer. Además, se proporciona una serie de tutoriales en forma de documentos y ejemplos. No hay muchas páginas web donde se ofrezca más información ni muchos vídeos.
Plataformas en las que se puede usar	Window y Linux.
Versión de FMI	1.0 y 2.0
Importación y exportación de FMI	Importa y exporta modelos CS.
Tipo de entorno	Presenta una interfaz buena, las simulaciones se pueden ver en gráficas, pero también existen componentes como los tanques de agua que al simular se puede ver como se vacían o se llenan, dependiendo del modelo que se haya creado.Además tiene el panel para buscar los componentes fácilmente.

En la Tabla 1.8 se presenta la información obtenida de OpenModelica. Se puede remarcar que, además de ser intuitivo, existen varios libros como [20] que indican cómo funciona el software de manera muy clara y con ejemplos, que pueden ayudar al usuario interesado en la aplicación. [24]

Tabla 1.8 Información de OpenModelica.

OpenModelica	
Descripción	Entorno de modelado y simulación basado en el código abierto Modelica diseñado para uso industrial y académico.
Tipo de modelos que puede generar	Puede crear modelos de varios dominios diferentes, como pueden ser eléctrico magnético, hidráulico, térmico, matemático, etc. Además, los bloque de los diferentes dominios se puede conectar entre sí en el modelo que se cree.
Existencia de documentación y tutoriales	Existe una gran cantidad de vídeos tutoriales y de páginas web. Asimismo, se puede encontrar de un manual de usuario muy completa en la página oficial del programa. También hay libros accesibles con el objetivo de enseñar cómo se usa esta herramienta correctamente.
Plataformas en las que se puede usar	Linux, Windows y macOs
Versión de FMI	1.0 y 2.0
Importación y exportación de FMI	Solamente importa ME y exporta ME Y CS.
Tipo de entorno	Es muy intuitivo, ya que en el panel de la izquierda aparecen todos los bloques que se pueden usar para crear un modelo, separados por dominios, lo que ayuda a encontrar el bloque se va buscando. La interfaz es bastante buena, tiene elementos con detalles y las soluciones las da con el detalle suficiente como para verificar la solución deseada.

En la Tabla 1.9 se expone la información del programa DACCOSIM. Tal y como se puede ver el objetivo de esta herramienta no es el que se iba buscando para este trabajo, ya que solo une los distintos FMU creados. Esa función en este trabajo se realiza programando varios test unitarios en Python.[13]

Tabla 1.9 Información de DACCOSIM.

DACCOSIM	
Descripción	Es un entorno de desarrollo y que ejecuta situaciones de Co-Cimulación impulsados por JavaFMI (fmu wrapper y fmu builder). La cuales son un conjunto de herramientas para que se ejecute la interoperabilidad de los archivos FMU.
Tipo de modelos que puede generar	Tiene tres tipos de componentes, bloques para los archivos FMU, bloques para designar Input y bloques de Output. Esta herramienta permite que los FMU puedan trabajar conjuntamente y da soluciones numéricas y gráficas.
Existencia de documentación y tutoriales	Solamente hay dos vídeos tutoriales, aunque hay un manual el la página web oficial, donde explica todas las opciones del programa, así como las instalaciones en las diferentes plataformas. Además, hay varias páginas web que proporcionan más información del la herramienta.
Plataformas en las que se puede usar	Linux y Windows.
Versión de FMI	2.0
Importación y exportación de FMI	Solo puede importar y exportar CS no ME.
Tipo de entorno	La interfaz es muy simple, solo tiene siete bloques distintos con sus nombres, por tanto es intuitivo pero muy sencillo. Aún así los resultados de la simulación que provee son detallados.

En el caso de las herramientas *open source* solo se han analizado esos cuatro software, porque hay cinco software de simulación compatibles con el estándar FMI. El quinto, se denomina FMU SDK de Synopsys y se trata de un software de simulación específicamente para vehículos, por lo que, no tiene gran interés para este trabajo. Las demás herramientas, que no son programas de modelado y simulación, son librerías que se pueden usar en estos software. Como puede ser el caso de *FMI Kit for Simulink de Dassault Systèmes* o *PyFMI de Modelon*. La última librería mencionada es la que se va a utilizar para ejecutar los test unitarios en Python.

En resumen, a lo largo de este trabajo se han utilizado *Simulink, OpenModelica* y *Hopsan*. La versión de Matlab, y por tanto de Simulink, con la que se ha trabajado para la realización del proyecto es Matlab R2022b, la versión de Open Modelica usada es la v1_20_64bit y la de Hopsan es la 2.21.1.

1.2.2 Test unitarios

Un test unitario o *unit testing* es un método por el cuál se puede comprobar que algunas unidades individuales más pequeñas funcionan correctamente. Estos test unitarios forman parte de la metodología ágil del trabajo de desarrollo, de la que se puede saber más en el apéndice A. Con estas pruebas se pueden detectar fallos en los diferentes sistemas antes de que el desarrollo del mismo esté más avanzado, evitando así que el error en el sistema sea mayor.

Se realizan mediante una escritura de fragmentación del código fuente en una aplicación o programa para poder probar las unidades del sistema que se desean. Las unidades que se someten al test tiene que funcionar correctamente y de manera individual. [2]

En este trabajo los test unitarios se van a programar en Python. Mediante estos test se pretende verificar el correcto funcionamiento de sistemas simples y algunos modelos más complejos, para comprobar los beneficios ofrecidos por la Co-Simulación.

En Python los test unitarios dan soporte a conceptos importantes de la programación orientada a objetos, como son:

- **Configuración de prueba**: un *test fixture*, es uno de los preparativos para pruebas de limpieza asociadas, por ejemplo, creación de bases de datos.
- **Caso de prueba**: un *test case*, verifica la respuesta del programa antes unas entradas.
- **Conjunto de pruebas**: agrupa distintas pruebas para realizarlas conjuntamente.
- **Ejecutor de pruebas**: un *test runner*, dirige la ejecución de las pruebas y da el resultado.

Para este trabajo se ha utilizado el *test case*. Las instancias del *test case* proporciona tres grupos, un grupo para ejecutar el test, otro para evaluar los fallos y reportarlos y el último para recopilar más información del test unitario.

Los métodos usados en los test unitarios para el desarrollo del trabajo son los que se presentan a continuación:

- *setUpClass()* : preparación del banco de los test, se realiza antes de ejecutar cualquier test unitario. El único argumento que tiene es la clase.
- *tearDownClass()*: se llama justo después de llamar al método de prueba, una vez que se ha registrado el resultado. Este método solo se implementa si el *setUpClass()* no ha dado ningún fallo.
- A continuación se ejecuta el el test unitario, modificando el valor de las entradas y obteniendo el resultado de la simulación.
- Se realiza un *assert* para verificar que no se están produciendo fallos específicos durante la ejecución del programa. [1]

Por tanto la estructura que van a tener los test unitarios usados para el trabajo es:

```
import libraries

class SimpleTestCase(unittest.TestCase)
    def setUp(self):
        set up options
        print input list
        print output list
        set initial variables
    def tearDown(self):
        unittest.TestCase.tearDown(self)
    def Function_name(self)
    t = 0.0;
    dt = 0.01;
    while(True)
        simulation of different cases at specific instants of time, with
            different values of input variables
        print the result of every case
        t +=dt
        it t>10: break # while loop breakout time
    assert
```

```
    #If there is some plots result
    plot

if __name__ == "__main__"
    unittest.main() #run all test
```

Pyfmi

Para poder trabajar con los archivos FMU en Python se necesita el paquete Pyfmi, el cual ofrece una interfaz con la que Python interactúa con los archivos generados por la exportación FMI. Este paquete es el que permite cargar los diferentes modelos, darle un valor a las variables de entrada, evaluar las diferentes ecuaciones del modelo... [5]

Pyfmi se tiene que instalar en Python, mediante Anaconda o pip, para poder usar sus funciones. Una vez instalado, para utilizar esas funciones, se debe importar al principio de cada test unitario.

Antes de explicar las funciones utilizadas en este trabajo, hay que tener en cuenta que dependiendo del Pyfmi instalado y la versión del estándar FMI que se está usando, la librería va a disponer de una serie de funciones u otras. Para verificar las funciones que contiene el paquete Pyfmi se escribe el siguiente código en la pantalla de comando, con el que se imprimen por pantalla todas las funciones del Pyfmi instalado.

```
import pyfmi
from pyfmi.fmi import FMUModelME2
print(dir(FMUModelME2))
```

Una vez que se conocen todas las funciones que se pueden usar, se investigaron cuales eran útiles para la ejecución de los diferentes test unitarios. Las funciones más importantes elegidas se describen a continuación.[7]

- *load_fmu()*: función que sirve para cargar el archivo FMU.
- *setup_experiment()*: determina la tolerancia del modelo, la tolerancia de la simulación, el tiempo de comienzo de la simulación, el tiempo de parada de la simulación y otro tiempo de parada definido.
- *get_model_variable()*: función que proporciona la lista de todas las variables definidas en el archivo "*modelDescription.xml*". En el caso de los test unitarios que se van a realizar lo que se quiere con esta función es poder crear dos listas, una de variables de entrada del modelo y otra de las variables de salida. Para poder separar estas variables en las listas mencionadas y obtener solamente las entradas, la función se utiliza de la siguiente manera: "*get_model_variable(causality=2)*" y para las variables de salida "*get_model_variable(causality=3)*". Se probaron otras dos funciones específicas para las entradas y salidas "*get_input_list*" y "*get_output_list*" respectivamente. El problema de estas funciones, es que si la variable es binaria no la añade a la lista, solo reconoce las variables reales. Por tanto, como en este trabajo se trabaja con variables binarias se ha optado por usar "*get_model_variables()*".
- *set()*: función para definir el valor de las variables de entrada del modelo.
- *get()*: obtiene el valor de las variables de salida durante las simulaciones.

Estas funciones se introducen en las diferentes partes de la estructura del test unitario y se van modificando según el sistema que se va evaluando.

2 Co-Simulación

Antes de presentar la Co-Simulación, con el objetivo de comenzar a conocer cómo se trabaja con el estándar FMI desde Open Modelica, Hopsan y Simulink se han realizado varias simulaciones de sistemas sencillos, que luego se implementarán mediante Co-Simulación generando así sistemas más complejos.

2.1 Simulaciones Aisladas

2.1.1 Puerta lógica: AND

Para poder comparar los diferentes archivos FMU, generados por las distintas plataformas mencionadas, se decidió crear un mismo sistema, en este caso una puerta lógica AND. Este sistema tan sencillo tiene como fin, además de dicha comparación, crear un test unitario que verifique que se obtiene la misma solución para los tres FMU, generados por las tres plataformas de modelado y simulación. Así se solventarán los problemas que pueden surgir al exportar mediante el estándar FMI, para que cuando se hagan las Co-Simulaciones, no se presenten esos mismos fallos.

En primer lugar se creó el mismo modelo en las diferentes plataformas, presentados en la Figura 2.1.

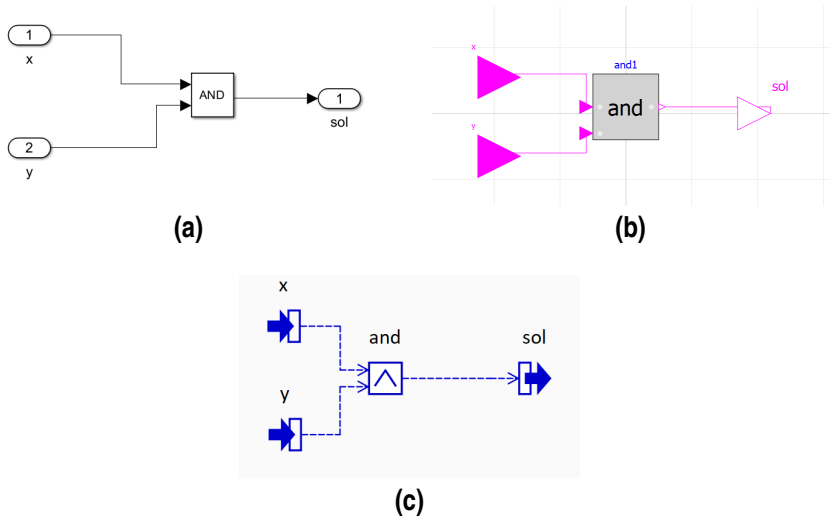


Figura 2.1 Modelos de la Puerta AND en (a) Simulink, (b) OpenModelica y (c) Hopsan.

Observando la Figura 2.1 se puede ver como es el mismo sistema en tres plataformas distintas. La puerta lógica AND tiene una salida llamada "sol" y dos entradas, que han sido denominadas "x" y "y".

A la hora de realizar los test unitarios de estos sistemas se verifican las diferentes combinaciones posibles de las entradas ("x", "y"), siguiendo la tabla de verdad de una puerta lógica AND, presentada en la Tabla 2.1

Tabla 2.1 Tabla de verdad de una Puerta lógica AND.

x	y	sol
0	0	0
0	1	0
1	0	0
1	1	1

Verificación mediante un Test unitario

Como se ha comentado con anterioridad, el objetivo del test unitario es tener el mismo test para los tres archivos con extensión ".fmu". Verificando así que los tres dan el mismo resultado, al ser el mismo sistema.

Uno de los inconvenientes que se presentaron cuando se quiso al crear este test unitario, fue que al exportar el modelo de la puerta lógica AND desde la plataforma Hopsan mediante el estándar FMI, el nombre de las entradas y salidas cambia. Es decir, las variables de entrada y salida del archivo FMU no se llaman igual al modelo que se creó en Hopsan. Al abrir el documento "*modelDescription.xml*" de dicho FMU, se averiguó que las variables de entrada se denominan con el nombre que se le ha dado en el modelo seguido de *"_out_"* y a las salidas se nombran con el nombre que se le ha dado en el modelo seguido de *"_in_"*. Por tanto, esto genera un problema a la hora de programar el test unitario, puesto que se tienen que llamar a las variables de entrada para darles un valor, y a la de salida para obtener el resultado, de la misma forma para los tres programas.

La solución para lograr un solo test para la verificación de las tres plataformas fue probar diferentes funciones que lleva incorporadas en "Pyfmi", paquete instalado en Python para poder trabajar con el estándar FMI. Con la función "*get_model_variable*()" se pueden obtener las listas de variables de entrada y salida del archivo FMU. Una vez se tienen las listas, si las variables se llaman como el modelo, se sigue manteniendo el nombre, y si se llaman como las variables de Hopsan, se cambia el nombre a la hora de referirse a ellas en el test unitario. No se cambia el nombre de las variables del archivo, solo en Python.

Tras solucionar el problema, se verifica que las tres puertas AND de las tres plataformas cumplen la tabla de verdad. En consecuencia se verifica que están bien diseñadas. Los resultados se presentan en la Figura 2.2.

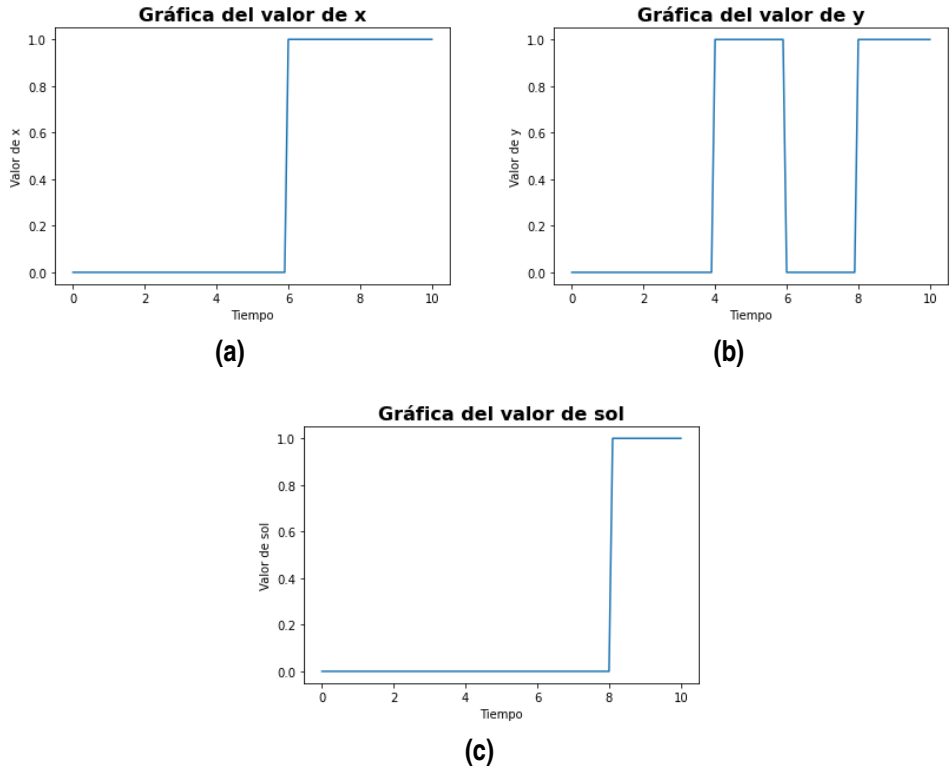


Figura 2.2 Gráficas de (a) valores de la variable de entrada 'x', (b) valores de la variable 'y'y (c) valores de la variable de salida 'sol'.

Además para poder comparar las distintas simulaciones, se ha hecho un análisis estadístico del tiempo que tarda en ejecutarse el test unitario completo y el tiempo de ejecución de la simulación, ejecutada por Python. Los resultados de este análisis se muestran en la Tabla 2.2. Estos resultados se corresponden con los tiempos medios de 50 simulaciones realizadas bajos las mismas condiciones.

Tabla 2.2 Resultado estadístico de los tiempos de simulación y el test unitario.

Plataformas	Simulink	Hopsan	Open Modelica
Tiempo medio de ejecución de la simulación del modelo (s)	0.0033	0.0053	0.0037
Tiempo medio de ejecución del test unitario (s)	0.2656	0.5459	0.7966

Si se observa la Tabla 2.2 se puede ver como el tiempo de la simulación de todos los casos es muy pequeño en comparación con el del test completo. Comprobando así, como el test unitario tarda bastante más tiempo cargando el modelo, iniciando todas las opciones del test unitario, así como definiendo las variables.

Si se comparan los tiempos entre las diferentes plataformas, se comprueba que el más rápido es el test unitario de Simulink. Debido a eso, se podría considerar más eficiente. Siendo un 67 % más rápido que Open Modelica en ejecutar el test untario, y un 51 % más rápido que Hopsan. El segundo programa más rápido es el de Hopsan,siendo un 31 % más rápido que Open Modelica. Por último, el que más tarda es Open Modelica, esto se puede ser debido a que en el archivo de extensión ".fmu" de Open Modelica se guardan todas las variables del modelo, no solo las variables de entrada y salida, sino todos los puertos de cada bloque usado.

Ante estos resultados se podría concluir que Simulink es el más rápido, como era de esperar, al ser una plataforma mejor optimizada. Si no se puede disponer de esta plataforma al ser de pago, sino que se comparan los tiempos de ejecución del test unitario de las dos plataformas *open source*, la más rápida sería Hopsan. Aunque los tiempos de la propia simulación son muy similares y muy pequeños, porque el sistema es muy simple, si que podría ser relevante cuando los modelos son más complejos y grandes.

Además de la comparación de los tiempos, se puede comparar las diferentes características de las tres plataformas, así como el contenido de los diferentes archivos generados al exportar el modelo mediante el estándar FMI.

Comparación de los distintos archivos FMU y las plataformas

Se van a comparar aspectos a considerar de los diferentes archivos FMU de las tres plataformas, para tenerlo en cuenta a la hora de hacer la Co-Simulación. Con esta comparativa, además se pueden sacar conclusiones de las tres plataformas usadas. Así, se podrá escoger entre una plataforma u otra para generar un modelo, según la aplicación concreta.

Para comparar los diferentes archivos de manera más exhaustiva, se van a presentar las características de los distintos archivos FMU y lo que ofrece cada programa a la hora de la exportación.

- **Simulink**: la exportación mediante el estándar FMI se puede hacer de dos formas. La primera es, con el comando *"s=exporToFMU(Model,Options)"*, que tiene como argumento de entrada el nombre del modelo de Simulink que se quiere exportar y diferentes opciones que ofrece el programa, (como especificar la dirección en la que se guarda el archivo, los nombres de los parámetros que se vas a exportar, los nombres de las variables internas, etc). La segunda alternativa para exportar mediante el estándar FMI, es haciendo clic en el botón de "*Standalone FMU*" que se encuentra en "*Save*". Al seleccionar en ese botón se abre una ventana con las diferentes opciones de exportación, como por ejemplo añadir o no una fotografía del modelo que se va a exportar, si se deja al usuario acceder al código fuente desde el FMU, la dirección en la que se guarda....

  Una de las opciones a destacar, y que puede resultar de interés para escoger o no Simulink, es que permite generar una fotografía del sistema creado en la plataforma y da la opción de incluirla en el FMU. En muchas ocasiones, el equipo que ha generado el archivo de extensión ".fmu" no quiere que se sepa cómo es el diseño del sistema de estudio, por tanto, se escogería la opción de no incluir la fotografía. Aunque, se puede presentar la situación contraria, pudiendo ser beneficioso incluir la fotografía del modelo. Por ejemplo, si es necesario consultar a otro equipo dentro de la mismo asociación que lo creó, o si el cliente que ha pedido ese modelo desea saber el diseño final.

  Al abrir del FMU generado por Simulink, se pueden encontrar el archivo "*modelDescription.xml*" y tres carpetas; una de binarios otra de fuentes y otra de documentación. En el "*modelDescription.xml*" se encuentra toda la información acerca del programa con el que se generó el FMU, la hora en la que se exportó el archivo, la versión de FMI usada, etc. La información más importante que contiene este documento es la definición de todas las variables del sistema, lo que incluye el tipo, el valor inicial, el nombre...

  En la carpeta de "*binaries*" solo de encuentra un archivo de extensión ".dll". En la carpeta de "*sources*" están todos los archivos de los código de C y C++ del modelo generado en Simulink. Por último en la carpeta "*documentation*" se encuentra una archivo llamado "*index.html*"

que contiene información acerca del dónde se encuentra el código fuente del FMU. Además el archivo "*index.html*", da una serie de instrucciones para volver a compilar la biblioteca dinámica FMU desde la fuente, según si se está usando Windowss, Linux o macOs.

Si se escoge la opción de no dejar acceso al código fuente, en el FMU sólo se encontraría la carpeta "*binaries*" con el archivo ".dll" y el "*modelDescription.xml*". Por tanto, el tamaño del FMU se reduce, y en consecuencia tarda menos en cargarse en los test unitarios.

Algunos aspectos para comparar con los otros archivos FMU generados de otras plataformas se muestran en la Tabla 2.3

Tabla 2.3 Características del FMU generado por Simulink.

Información que contiene	Tamaño	Sensibilidad del contenido
Carpeta binarios modelDescription.xml Carpeta documentation Carpeta de sources	Sin la fotografía 83 KB Con fotografía 88 kB Sin acceso al código fuente 60 KB	con la fotografía se revela como se ha creado el sistema a evaluar pero si no contiene la fotografía solo se conocen las entradas y salidas del modelo

- **Open Modelica**: en el caso de este programa la exportación FMI se ajusta directamente desde la configuración del mismo. En ella se puede seleccionar si se quiere obtener solamente "*Model Exchange*", Co-Simulación o ambas, además de la versión FMI que se usa, la 1 o la 2. También da la posibilidad de denominar al FMU con un nombre concreto, así como la posibilidad de elegir el directorio donde guardarlo. Si estas dos últimas opciones no se modifican, se guarda con el mismo nombre del sistema y en la misma dirección donde se encuentran los archivos donde se descargaron las carpetas del programa. Asimismo, da la opción de el tipo de "*solver*" que se quiere usar y varias elecciones más.

  Al igual que ocurría en Simulink, dentro del archivo FMU generado se encuentra el archivo "*modelDescription.xml*", en el que se encuentra una información muy parecida a la descrita anteriormente con Simulink. Se da información de la fecha en la que se creó en archivo, el programa a partir del cual se exportó, la versión FMI usada, etc. La diferencia es que no solo se definen las variables de entrada y salida del modelo, sino que también se proporciona los datos de todas las variables de los puertos de cada bloque usado en el sistema. Al saber la información de todos los puertos de los bloques se puede intuir cómo se ha creado el sistema que se va a evaluar, si se conoce la nomenclatura de cada puerto de cada bloque usado. Por tanto, si el usuario que genera el FMU no quiere que se sepa como se ha creado el sistema no es conveniente que se describan todas estas variables.

  Se pueden encuentrar tres carpetas más, una carpeta de "*binaries*", otra "*resources*" y por último la denominada "*sources*". La primera carpeta incluye dos archivos con extensiones distintas ".dll" y ".lib", previamente explicadas. El contenido de la carpeta "*resources*" son otros dos documentos con diferentes extensiones un ".json" y otro ".log". En el caso de Open Modelica, a diferencia de Simulink, se incluye el documento ".json", que como se comentó con anterioridad en la introducción, es muy similar al "*modelDescription.xml*". Por último, en la carpeta restante al igual que en Simulink se encuentran todos los códigos generados de C y C++.

Una de las opciones que tiene Open Modelica en la exportación del FMU, es que se puede o no incluir dentro del FMU la carpeta de "*sources*". Si no se incluye, el FMU funciona correctamente y su peso es menor que cuando contiene la carpeta.

En la Tabla 2.4 se observan los aspectos relevantes para comparar con los otros FMUs.

Tabla 2.4 Características del FMU generado por Open Modelica.

Información que contiene	Tamaño	Sensibilidad del contenido
Carpeta binarios modelDescription.xml Carpeta resources Carpeta de sources	Con la carpeta *sources* 726 KB Sin la carpeta *sources* 341 KB	el modelDescription.xml es más extenso y si se sabe interpretar se podría conseguir averiguar cómo se ha creado el sistema que se está estudiando

Si se compara el peso de los diferentes documentos ".xml" el generado en el FMU de Open Modelica (6 kB) es prácticamente el triple de pesado que el de Simulink (2.3 kB). Esto demuestra también que en el archivo de Open Modelica se describen muchas más variables que en el de Simulink, en el que solo se incluían las variables de entrada, salida y el tiempo. El ".dll" de Simulink también se ha comprobado que es aproximadamente siete veces menos pesado que el de Open Modelica.

- **Hopsan**: en este programa el método de exportación es el más simple de todos. Existe una pestaña llamada "*export*", y dentro de ella está la opción de FMI, cuando se clica en esa opción aparecen cuatro más. Entre las cuatro opciones de exportación, mediante el estándar FMI, que permite Hopsan, se puede elegir la versión de FMI con la que exportar (FMI 1 o FMI 2), y si se prefiere la opción de 32 bit o 64 bit. Una vez se selecciona una de esas cuatro opciones, el programa genera directamente el archivo FMU con el mismo nombre que el sistema que se está exportando. La carpeta en la que se va a guardar el FMU la selecciona el usuario.

  Para poderlo comparar con los demás se ha escogido la misma versión de FMI usada con Simulink y Open Modelica, es decir, la versión 2 de FMI. Además, se ha exportado con 64 bit, porque es lo que se ha hecho con los otros dos.

  El contenido del FMU generado por Hopsan, como en los otros dos FMU descritos, es un archivo "*modelDescription.xml*" y dos carpetas "*binaries*" y "*resources*". El "*modelDescription.xml*" contiene información muy parecida al de Simulink, ya que solo se describen las variables de entrada y salida. También se proporciona la información del programa con el que se generó, la fecha y hora etc. Hay que destacar que, en este caso el archivo pesa 1 kB, por tanto es el que menos peso contiene, se puede deducir que es el que menos información da. La carpeta "*resources*" se encuentra vacía y la carpeta "*binaries*" contiene un archivo ".dll" que es alrededor de 74 veces mayor que el de Simulink, lo que hace que sea el que más tamaño tenga de los tres archivos generados.

  Gracias al archivo "*modelDescription.xml*" se supo como Hopsan llama a las variables de entrada y salida. Sin ese archivo sería muy difícil averiguar cómo se guardan, y por tanto, no hubiera sido posible usar este FMU en los test unitarios. Este documento es muy importante para verificar el modelo que se ha generado y comprobar que se tienen las variables deseadas.

Asimismo, si existe algún fallo que tiene relación con el nombre o el tipo de variable, se puede recurrir a él para solucionarlo más rápidamente.

En la Tabla 2.5 se muestran los diferentes aspectos destacados, al igual que en los otros dos programas, para poder terminar de comparar los FMU.

Tabla 2.5 Características del FMU generado por Hopsan.

Información que contiene	Tamaño	Sensibilidad del contenido
Carpeta binarios modelDescription.xml Carpeta resources	1780 KB	el modelDescription.xml es útil para conocer el nombre de las variables, pero no se puede intuir a partir del mismo cómo es el sistema que se está estudiando

Conclusiones

Una vez recopilada toda la información de los programas y los distintos FMU, se puede decir que Hopsan es la plataforma más sencilla para la exportación FMI. Siendo la que menos opciones proporciona al usuario para ajustar el archivo FMU que se genera. Si lo que se pretende es tener muchas opciones para crear un FMU específico, la plataforma Simulink sería la que mejor puede ajustar el interior del FMU por las opciones que da. Open Modelica sería un programa que da opciones para poder generar un FMU más personalizado, aunque no tantas como Simulink. Dependiendo de la aplicación se podría elegir una u otra.

En el caso de escoger Hopsan habría que tener en cuenta el cambio del nombre de variables. Lo que se puede solucionar en Python, como se ha explicado anteriormente, si se desea hacer el mismo test para sistemas iguales provenientes de diferentes software. En resumen, si se pretende que el bloque de Hopsan trabaje con otro, entonces las salidas de Hopsan que sean entradas del otro FMU o viceversa, se deberán nombrar igual, ya sea cambiando el nombre en el diseño del otro programa o en Python.

Al comparar los FMU, el que menos pesa es el de Hopsan, siendo un 96 % mayor que el FMU de Simulink que menos tamaño tiene, y un 80 % mayor que el FMU de Open Modelica, generado sin la carpeta de "*sources*". Sin embargo, no es el más eficiente en términos de tiempo de ejecución del test unitario. Además, es el menos completo de los tres, ya que solo tiene el "*modelDescription.xml*" y el ".dll". El más eficiente de todos, con respecto del tiempo de ejecución y simulación, es Simulink. Siendo un 67 % más rápido que el test unitario con Open Modelica y un 51 % más rápido que el test con el archivo generado por Hopsan. Asimismo, el archivo FMU es más completo que el de Hopsan, pero menos que el de Open Modelica. Comparando los tamaños de los archivos, en ninguno de los tres casos, expuestos en la Tabla 2.3, tiene un tamaño mayor que los otros FMUs. El de Open Modelica es el que más tarda en la ejecución del programa completo y el que mejor describe al programa diseñado en el "*modelDescription.xml*".

La elección de la plataforma dependerá de varios factores a tener en cuenta, como la sensibilidad del contenido que se quiera tener, si se desea una descripción del modelo más extensa o no, el tamaño del FMU etc. En este caso concreto, al tratarse de un sistema muy simple, una puerta lógica AND, lo que se quiere es que el tiempo de ejecución sea lo más corto posible. También sería una ventaja que la exportación fuera sencilla y que el contenido del "*modelDescription.xml*" fuera fácil de analizar. Por tanto, el FMU que cumple mejor con estos aspectos sería el generado por Simulink.

2.1.2 Símil eléctrico

En esta sección del trabajo se ha querido estudiar una analogía electro hidráulica. Concretamente lo que se va a analizar es el llenado y vaciado de un tanque de agua. Esta acción se pretender emular con un circuito eléctrico compuesto por condensadores, ya que la carga y descarga de un condensador puede parecer el llenado y vaciado de un tanque.

Para empezar, se diseñó el circuito eléctrico que se va a simular, con el objetivo de controlar la carga y descarga de un condensador. Este circuito consta de una fuente de voltaje, dos resistencias, dos interruptores, un condensador y un voltímetro. Tal y como se ve en la Figura 2.3.

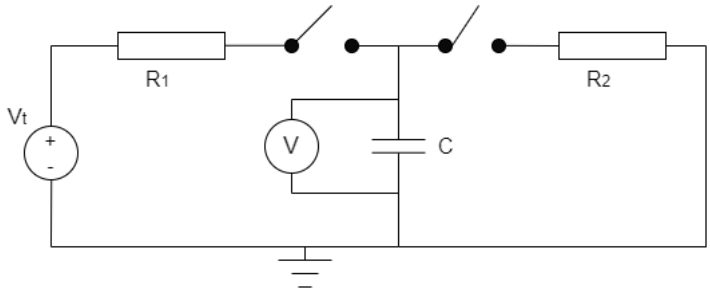


Figura 2.3 Diseño inicial del circuito eléctrico.

Los valores de cada uno de los componentes del circuito se calculan teniendo en cuenta que el objetivo es conseguir que el interruptor se cargue y se descargue totalmente en dos segundos, y que esa carga total sean 0.27 V. Siguiendo el esquema de la Figura 2.3, la tensión del condensador y la intensidad del mismo se va a denominar V_c *y* I_c respectivamente, los de la resistencia V_R *y* I_R. La carga almacenada en el condensador se representa por la letra Q.

Las ecuaciones que se van a utilizar para analizar el circuito son las que se presentan en 2.1.

$$\begin{cases} V_c = \frac{Q}{C} \\ V_t = V_R + V_c \\ I_c = I_R \end{cases} \tag{2.1}$$

En los instantes iniciales, Q es nula y por tanto, el voltaje del condensador va a ser nulo también, debido a eso, la tensión total recaerá sobre la resistencia en ese primer instante. A medida que aumenta el tiempo, el condensador comienza a cargase, la tensión del mismo va a comenzar a aumentar y la intensidad va disminuyendo. Tal y como se observa en las ecuaciones que se muestran en 2.2.

$$\begin{cases} V_R = V_T - V_c \\ I_c = I_R = \frac{V_R}{R} = \frac{V_t - V_c}{R} \end{cases} \tag{2.2}$$

Una vez que el condensador se carga, la tensión del condensador alcanza el valor de la fuente. Por tanto, el valor de la fuente en este caso será 0.27 V. Observando las fórmulas, se comprueba que la intensidad va tendiendo a cero, por ende, cuando el condensador está totalmente cargado se comporta como un circuito abierto.

Para considerar el tiempo que tarda el condensador en cargarse, hay que tener en cuenta la constante de tiempo. La cual se denomina τ. Para calcular esta constante de tiempo se tiene la fórmula 2.3

$$\tau = R(\Omega) \cdot C(F) \tag{2.3}$$

Se considera que cuando han transcurrido cinco constantes de tiempo el condensador está totalmente cargado. En este caso el objetivo es que se cargue en dos segundos, por lo que, si imponemos el valor de la resistencia a 400 Ω entonces la capacidad del condensador se calcula con la ecuación 2.4

$$t_{carga} = 5 \cdot \tau = 5 \cdot R \cdot C \longrightarrow C = \frac{t_{carga}}{5 \cdot R} = \frac{2}{5 \cdot 400} = 0.001F \tag{2.4}$$

El circuito con los valores calculados se muestra en la Figura 2.4.

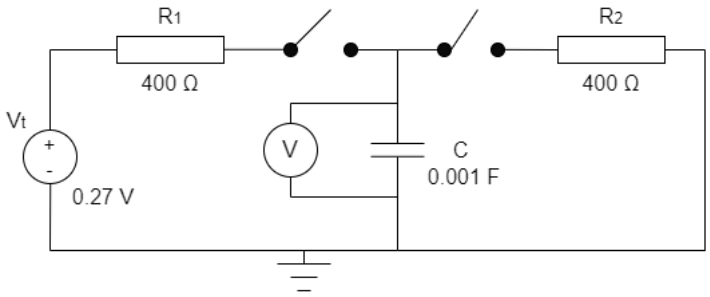


Figura 2.4 Diseño del circuito inicial.

Una vez que se ha diseñado el circuito, se implementa en Simulink, Open Modelica y Hopsan. Al ser un circuito eléctrico, y no electrónico, se han presentado varios problemas en dos de los software usados para crear el circuito.

El primer problema que se presentó fue, que los componentes de Hopsan pueden tener tres puertos distintos, puerto Q, puerto C o puerto S. Los componentes con puerto Q representan componentes resistivos, como pueden ser las resistencia y los interruptores. Los componentes con puerto C simbolizan elementos de líneas de transmisión, como pueden ser algunas fuentes eléctricas o los condensadores. Por último, los elementos con puertos tipo S son para componentes de señal, que son usados generalmente para operaciones matemáticas.

El principal problema es que los componentes con puerto Q no se pueden unir directamente a otro componente con el mismo puerto, lo mismo pasa con los componentes de puerto C . Por lo que, solo se pueden conectar puertos Q con puertos C. Como la resistencia y los interruptores tienen puertos Q, no se pueden unir directamente. Por tanto se tuvo que rediseñar el circuito, tal y como se ve en la Figura 2.5.

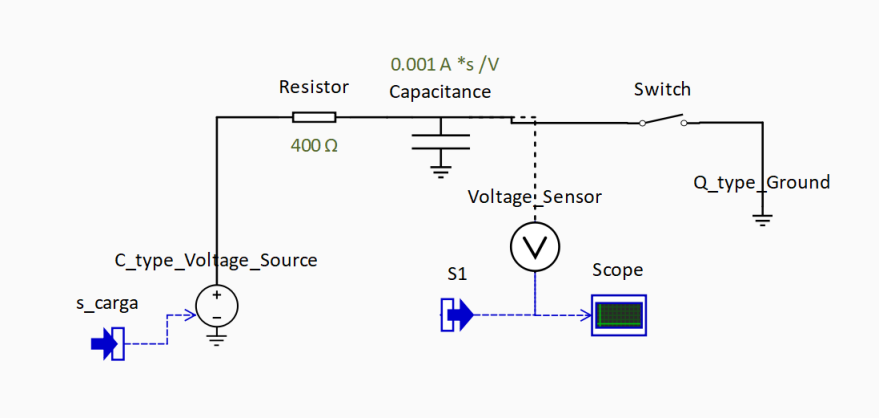


Figura 2.5 Circuito implementado en Hopsan.

Si se observa la Figura 2.5 se puede ver que la solución es diferente al circuito que se diseñó en un primer momento. Se ha utilizado una fuente de voltaje variable, la cual va a servir de interruptor. En el test unitario lo que se va a ir modificando es el valor del voltaje de la fuente, cuando se quiera cargar s_{carga} tendrá como valor 0.27 V, y para descargarse será nulo. El interruptor ("*Switch*") no tiene ninguna función, solo sirve para poder conectar el condensador, que tiene puerto C, con la tierra que también tiene puerto C. Por tanto, necesitan un elemento intermedio con puertos tipo Q, y se optó por un interruptor que va a permanecer cerrado durante toda la simulación.

Con Simulink se presentó otro problema. El circuito se pudo implementar tal y como se diseñó sin ningún problema, es decir, la creación del circuito eléctrico en este software es muy sencilla. Además se obtuvo la respuesta deseada. El problema se presenta en la exportación del FMU. Para exportar cualquier sistema desde Simulink mediante el estándar FMI, es necesario modificar las opciones de Simulink. En Simulink hay dos tipos de "*solvers*": el "*Fixed Time*" y el "*Variable Time*". Para exportar mediante FMI, es necesario haber seleccionado "*Fixed Time*". Al seleccionar ese tipo de "*solver*", el software no es capaz de realizar correctamente la simulación del sistema, y por tanto la solución no es la correcta. Incluso simulando en Simulink, con la opción "*Fixed Time*", el resultado obtenido no es el que corresponde a ese circuito, es decir, la gráfica no es la que debería ser. Aunque si el mismo sistema se simula con la opción "*Variable Time*", sí que funciona adecuadamente.

En consecuencia, al tener que seleccionar la opción "*Fixed Time*" para poder generar el archivo FMU, no se ha podido utilizar, ya que la solución no era la correcta, al ser el "*solver*" incapaz de simular el sistema adecuadamente.

En la Figura 2.6 se puede ver el circuito implementado en Simulink, aunque no se pudo exportar.

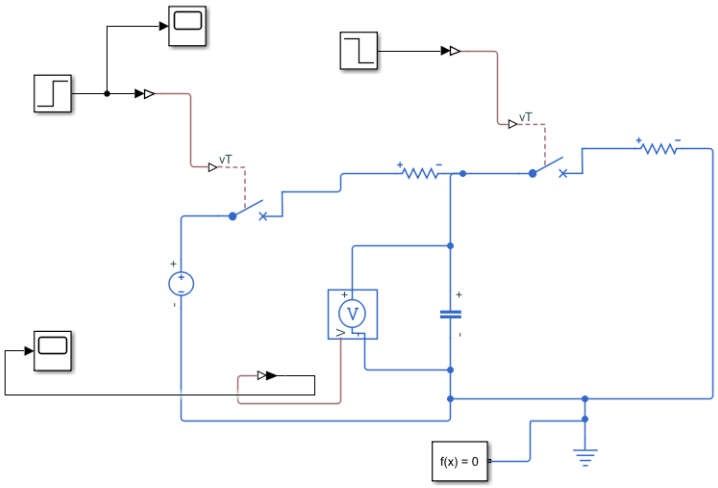


Figura 2.6 Circuito implementado en Simulink.

En cambio, durante la implementación y exportación del circuito de Open Modelica no surgió ningún tipo de problema. El circuito se presenta en la Figura 2.7.

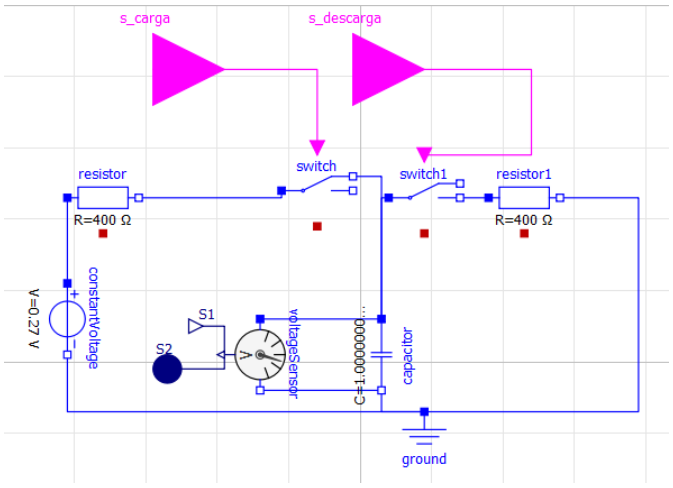


Figura 2.7 Circuito implementado en Open Modelica.

Una vez que se han generado los FMU de los circuitos eléctricos, que emulan el llenado y vaciado de un tanque de agua, se ha diseñado una simulación de llenado y vaciado de tanque de agua en Open Modelica. Al ser esta herramienta la que no presentado ningún problemas. El sistema que se ha diseñado es el que se muestra en la Figura 2.8.

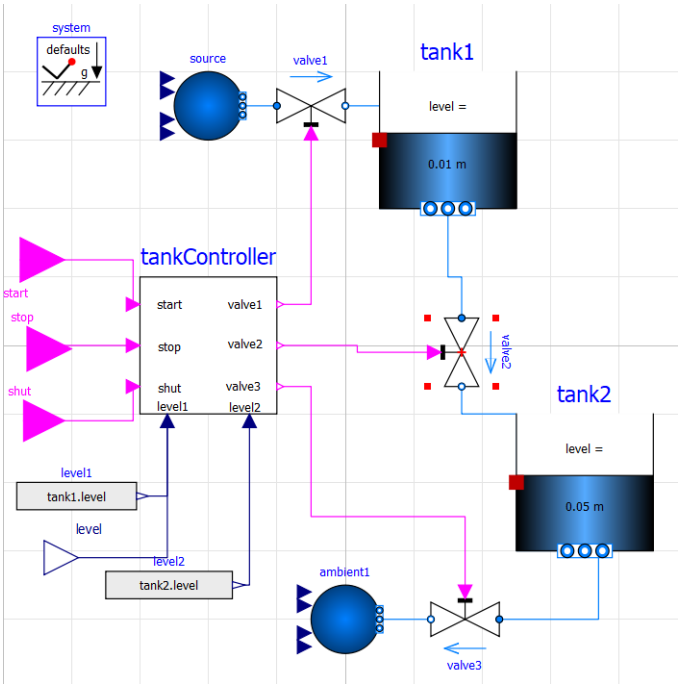


Figura 2.8 Sistema hidráulico de llenado y vaciado de un tanque.

Comparando las Figuras 2.8 y 2.7 se puede ver una de las ventajas del símil eléctrico. El circuito eléctrico es más sencillo de diseñar e implementar que el sistema hidráulico. En el test unitario del sistema hidráulico sólo se mide el nivel del "*tank1*", pero se ha tenido que introducir otro tanque porque el "*tankController*" se tiene que conectar a tres válvulas y medir dos niveles. Sin este controlador no se puede exportar de manera correcta el sistema de tanques. Además, los parámetros para conseguir que el tanque se llene y de descargue en dos segundos son más complicados de calcular. Otra complicación es que algunos parámetros de los componentes del sistema no tienen

las unidades indicadas, por lo que hace más difícil su ajuste. Por tanto, a la hora de una simulación es más sencillo crear un circuito eléctrico que un sistema hidráulico.

Se han programado tres test unitarios en Python para los dos circuitos eléctricos, ya que son diferentes, y el sistema hidráulico. El objetivo de estos test es, obtener un gráfica de la carga y descarga del condensador y del llenado y vaciado del tanque de agua. En dicha gráfica se tiene que mostrar como se llenan en dos segundo, se mantienen con el mismo nivel otros dos segundos y se descargan en dos segundos. Los resultados obtenidos se muestra en la Figura 2.9.

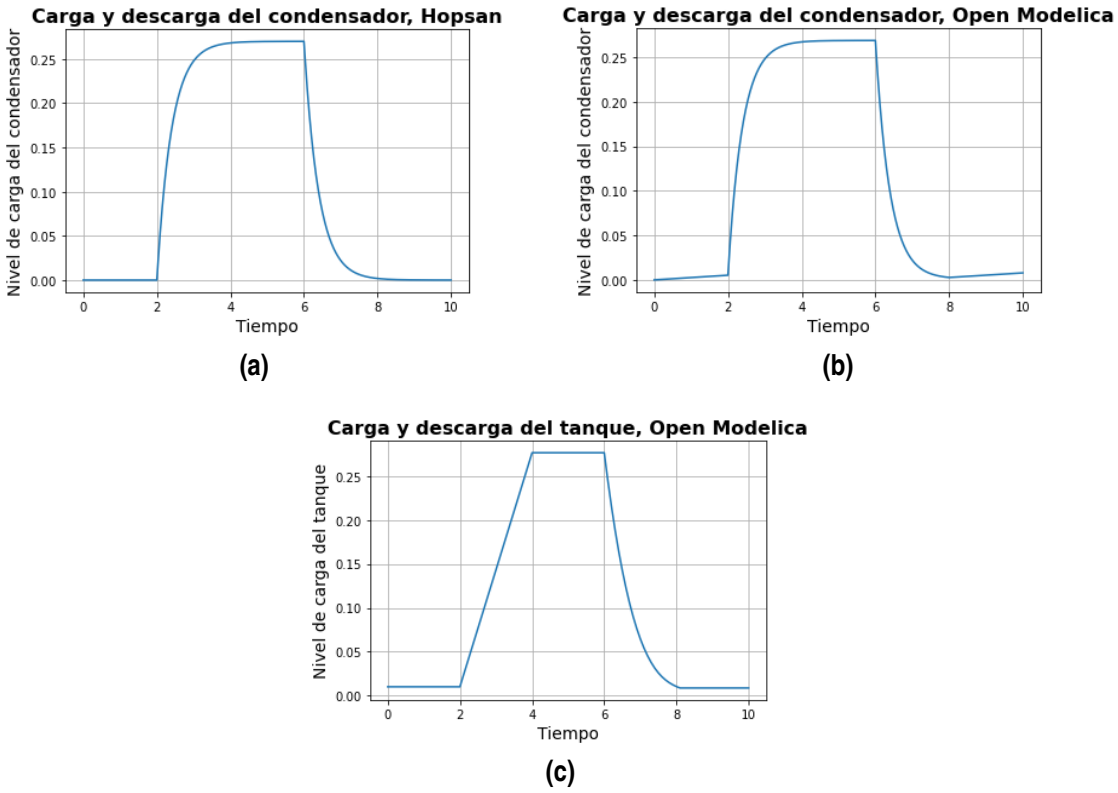


Figura 2.9 Gráficas de (a) carga y descarga del condensador, Hopsan, (b) carga y descarga del condensador, Open Modelica y (c) llenado y vaciado del tanque de agua .

Se puede ver como el resultado de los circuitos eléctricos, es decir, las Figuras 2.9 (a) y (b) son exactamente iguales. En cambio, en el resultado del sistema hidráulico cambia la curva de llenado del tanque, como se ve en la Figura 2.9 (c). Esta diferencia se puede deber al tipo de componente usado para la simulación, ya que el llenado de un tanque realmente suele seguir la gráfica del llenado de los condensadores, no una línea recta. También es posible que haya un fallo de fuga de memoria de Open Modelica, ya que se ha comprobado que las versiones más actuales de este programa, dependiendo del sistema que se simule, pueden tener este tipo de fallos.

Con estos resultados se puede ver otra ventaja del símil eléctrico, mediante el circuito eléctrico se consigue emular la realidad del llenado y vaciado de un tanque fielmente. Mientras que, con la simulación del sistema hidráulico los resultados distan más de la realidad.

Para seguir analizando las ventajas del símil eléctrico frente a la simulación del sistema hidráulico, en la Tabla 2.6 se muestra el tamaño de cada archivo FMU generado y simulado en los test unitarios.

Tabla 2.6 Tamaño de los distintos FMUs.

	Circuito eléctrico		Sistema Hidráulico
	Open Modelica	Hopsan	Open Modelica
Tamaño (KB)	974	1780	1417

Al observar la Tabla 2.12 se puede comprobar como el FMU generado por Hopsan es el que más tamaño tiene, aunque si observamos la Tabla 2.5, del análisis de la puerta lógica AND, se puede ver que pesa lo mismo, aunque el contenido es mayor. A lo largo de este documento se va a poder verificar que los archivos FMU exportados desde Hopsan tienen prácticamente el mismo tamaño, independientemente del sistema que se esté simulando. Por este motivo no es una muy buena comparación con el FMU del sistema hidráulico.Sin embargo, si se quieren comparar los tamaños más exhaustivamente, el FMU del circuito eléctrico de Hopsan es 45 % más pesado que el FMU del circuito eléctrico de Open Modelica.

La comparación de la que se pueden sacar realmente conclusiones se da entre el archivo FMU del circuito eléctrico y el FMU del sistema hidráulico, ambos exportados desde Open Modelica, con las mismas opciones de exportación, para que la comparación sea lo más justa posible. Si se observa la Tabla 2.6 se confirma otra de las ventajas del símil eléctrico, y es que el archivo generado es de menor tamaño que el sistema hidráulico, lo que puede ser beneficioso si se hacen Co-Simulaciones, y se tienen muchos archivos FMU que trabajen conjuntamente. En concreto, el tamaño del FMU del sistema hidráulico es un 31 % mayor que el del FMU del circuito eléctrico.

Además del tamaño de los diferentes FMUs se puede analizar el tiempo que tarda cada archivo en ejecutar el test unitario y el tiempo que tarda en hacer la propia simulación. Al ser tres archivos diferentes, y por tanto tres test unitarios distintos, la comparación no es muy precisa. A pesar de ello, puede ayudar a ver si el símil eléctrico puede ser un poco más eficiente, en términos de tiempo, o no. Para este análisis se han tomado 50 muestras de los tiempos de ejecución del test unitario completo y del tiempo de la simulación dentro de ese test unitario. De esas 50 muestras se ha calculado la media, esos resultados se muestran en la Tabla 2.7.

Tabla 2.7 Tiempo medio de ejecución del test unitario y tiempo medio de simulación para el símil eléctrico y el sistema hidráulico .

	Circuito eléctrico		Sistema Hidráulico
	Hosan	Open Modelica	Open Modelica
Tiempo medio de ejecución de la simulación del modelo (s)	0.0324	0.0338	0.0099
Tiempo medio de ejecución del test unitario (s)	0.4490	0.7610	0.8890

Al analizar los resultados obtenidos de la Tabla 2.7 se puede ver como el más eficiente es Hopsan. Esto se debe a que Hopsan no guarda en el FMU cada variable de cada puerto de cada componente que forma el circuito, al contrario de Open Modelica que sí que las guarda. Por tanto, tarda menos en cargarse. Concretamente, el FMU de Hopsan es un 49 % más rápido que el FMU del sistema hidráulico y 40 % más rápido que el circuito eléctrico de Open Modelica. Además hay que tener en cuenta que solo tiene una variable de entrada y una de salida, por lo que es lógico que sea el más rápido. Más adelante se demostrará que la ejecución del test unitario de archivos FMU generados

por Hopsan generalmente es rápida que con archivos exportados desde Open Modelica.

Si se observa el tiempo de simulación de los tres FMUs, se ve que el más rápido es el sistema hidráulico, esto se puede deber a que aunque el sistema sea más complicado, al ser los elementos que lo conforman más específicos y desarrollados hagan que la simulación sea más rápida. Este resultado no supone un inconveniente para el símil eléctrico ya que lo que realmente importa es el tiempo de ejecución del test unitario y además que se están comparando centésimas de segundos con décimas de segundos, son tiempos muy pequeños, a la hora de la simulación apenas se va a apreciar. Particularmente, la simulación en el test unitario del sistema hidráulico es un 70 % más rápida que la simulación del circuito eléctrico de Open Modelica y un 69 % más rápido que el FMU del circuito de Hopsan.

Lo más importante de estos resultados es que se verifica que la ejecución del test unitario de los circuitos eléctricos es más rápida que la del sistema hidráulico. Se puede ver como el test unitario del circuito eléctrico de Open Modelica es un 14 % más rápido que el del sistema hidráulico. Se comprueba así, que el símil eléctrico es más eficiente que la simulación del sistema hidráulico.

Conclusiones

En esta sección se ha comprobado la utilidad del símil eléctrico, además de las ventajas que tiene frente a simulaciones de los propios sistemas.

La ventajas principales que se han podido ver claramente, es que el circuito del símil eléctrico es mucho más sencillo de implementar en varias plataformas de simulación que el sistema hidráulico. La gran mayoría de los software de simulación trabajan con componentes eléctricos y electrónicos, no todos tienen elementos para crear un sistema hidráulico. Además, si se trabaja con Open Modelica, que sí dispone de una sección dedicada a los fluidos, esos componentes que conforman el sistema son más complicados de ajustar para que, en este caso, el tanque se llene y se vacíe en un tiempo determinado y que llegue hasta un nivel requerido. También hay que destacar que hay muchos de los parámetros que definen el comportamiento de cada elemento que no indican las unidades. Por tanto, es más complicado de implementar este tipo de sistemas. En cambio, con el circuito eléctrico no ha habido ningún problema en Open Modelica y se ha conseguido ajustar los parámetros muy fácilmente, siguiendo las fórmulas comentadas en la sección, para obtener el resultado deseado.

Si se fijan de nuevo en los resultados, en la Figura 2.9, se comprueba que la gráfica de la carga y descarga de los condensadores es más parecida al llenado y vaciado de un tanque en la realidad, que el resultado obtenido de la simulación del sistema hidráulico. Esto se puede deber a los componentes usados, ya que los componentes eléctricos y electrónicos están muchos más ajustados a la realidad en los software de simulación que elementos más complicados, que se definen por muchas ecuaciones, como pueden ser todos los de la familia de la mecánica de fluidos. También puede ser por un fallo de fuga de memoria de Open Modelica, hay versiones actuales en las que dependiendo del sistema que se quiera simular, pueden producirse fallos de fuga de memoria y por tanto, no ajustarse adecuadamente a la realidad.

Otra de las ventajas que presenta es que el archivo FMU es de menor tamaño, un 31 % menor , por tanto, al cargar el archivo cuando se ejecutan los test unitarios, el tiempo que tarda es menor. Se ha comprobado como el tiempo de ejecución del test unitario del circuito eléctrico es menor que el del sistema hidráulico, en un 14 %, aunque la diferencia de tiempos es muy pequeña. En el caso en el que se tuviera que hacer una Co-Simulación, el archivo FMU del circuito al tardar menos beneficiaría al tiempo de ejecución del test, haciendo que fuera menor que si se tratara con el FMU

del sistema hidráulico.

En conclusión, si se necesita simular el sensor de cabina que indica el nivel de los tanques de combustible de un avión, hacerlo con un circuito eléctrico, con condensadores, que emule esa acción, tiene múltiples beneficios frente a la simulación del sistema de tanques en sí. Además la curva obtenida, o el movimiento de aguja del sensor reflejará mejor la realidad con las simulaciones de los circuito eléctricos, ya que si son sistemas hidráulicos habría que saber cómo de bien se ajustan a la realidad.

2.2 Modelos realizados con Co-Simulación

A lo largo de esta sección se va comprobar el funcionamiento de la Co-Simulación estudiándola en dos sistemas. El primer sistema se trata de un circuito de puertas lógicas que representa la probabilidad de fallo de los actuadores junto con los FCCs de la aeronave, parte del sistema *Fly By Wire (FBW)*. El segundo caso que se presenta, es un circuito de puertas lógicas que equivale a la detección de fallos en el FADEC (*Full Authority Digital Engine Control*) del motor de un avión.

2.2.1 Sistema del sistema *Fly By Wire* de una aeronave

Actualmente el sistema de control de vuelo que llevan las aeronaves se denomina sistemas "*Fly By Wire*" (FBW) o pilotaje de mando eléctrico. Esta tecnología consiste principalmente en sustituir los sistemas mecánicos por eléctricos, así el control de las superficies aerodinámicas se lleva a cabo mediante medios eléctricos y se ejecuta mediante medios hidráulicos.

Los ordenadores que interpretan las leyes de control y mandan órdenes a las diferentes superficies aerodinámicas del avión, para evitar que éste salga de la envolvente de vuelo, se denominan "*Flight Control Computer* " (FCC). Los FCCs son los componentes principales de los sistemas FBW.

En el siguiente esquema de la Figura 2.10 se presenta un ejemplo simplificado de la conexión de los FCC con los actuadores de las superficies aerodinámicas, es decir una parte del sistema FBW de una aeronave. Este esquema puede ser diferente según la aeronave, ya que no todas tienen el mismo número de FCCs a bordo y los actuadores aerodinámicos pueden variar de un avión a otro.

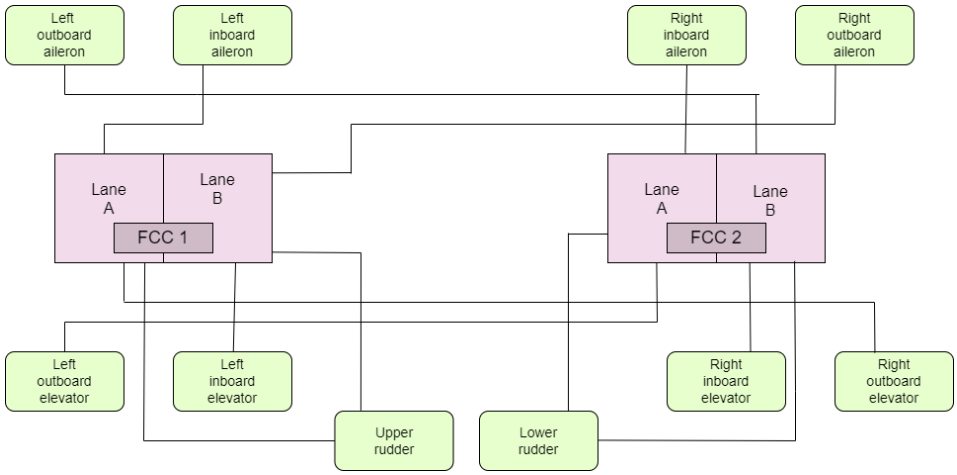


Figura 2.10 Esquema de un ejemplo simplificado de la conexión de los actuadores y los ordenadores de una aeronave. .

En la Figura 2.10, se puede ver que los bloques de color verde son los actuadores principales para el control de la aeronave, es decir, los alerones, los elevadores y el timón de dirección. Estos actuadores se conectan a los FCCs.

Este sistema es crítico para las operaciones de las diferentes aeronaves en las que se implementa, de ahí la importancia de evaluar el fallo total del sistema. Para ello, siguiendo el esquema presentado, se ha diseñado un circuito de puertas lógicas que detecta el fallo total de este sistema. Este circuito es el que se observa en la Figura 2.11.

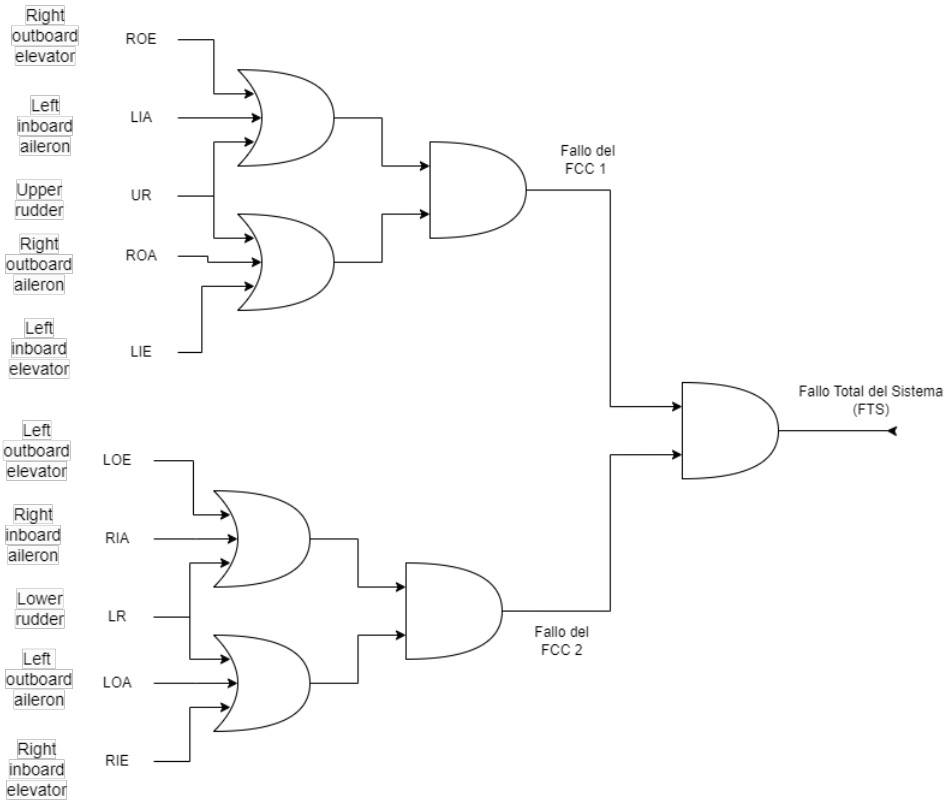


Figura 2.11 Circuito de puertas lógicas que detecta el fallo total del sistema..

Las entradas del circuito son los actuadores del esquema de la Figura 2.10, aunque como se puede observar se ha simplificado el nombre de cada uno de ellos usando la primera letra del nombre completo de los bloques del esquema. Se ha señalado también, el fallo de cada uno de los FCC, y por último, si éstos dos fallan sería cuando falla el circuito completo, que se corresponde con la salida.

En consecuencia, este circuito es un ejemplo en el que se podría aplicar la Co-Simulación de los tres programas distintos que se han mencionado anteriormente. Para ello se divide el modelo en tres subcircuitos que se implementarán en los Simulink, Open Modelica y Hopsan. Esta división se muestra en la Figrua 2.12.

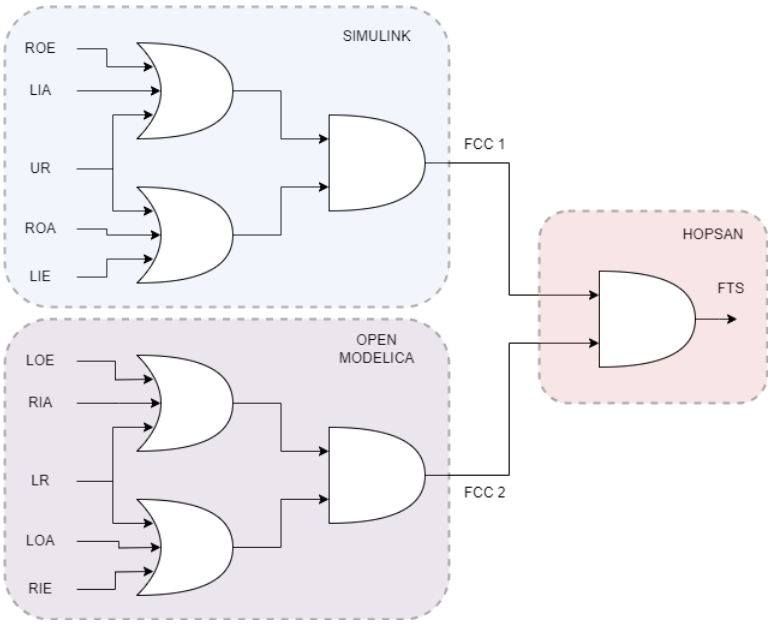


Figura 2.12 División del circuito de la detección del fallo total del sistema FBW..

En Hopsan se diseña la misma puerta AND analizada anteriormente, con la diferencia del nombre de las entradas y de la salida. En este caso las entradas serán FCC1 y FCC2 y la salida FTS. Hay que tener en cuenta que Hopsan cambia el nombre de las variables. Por lo que, en el archivo FMU estarán guardadas como "*FCC1_out_*", "*FCC2_out_*" y "*FTS_in_*".

En Simulink y en Open Modelica se va a diseñar el mismo sistema pero con nombres de entradas y salidas diferentes. Tal y como se ve en la Figura 2.12. Antes de hacer la Co-Simulación se verifica el correcto funcionamiento de estos subsistemas. Para hacer esa verificación, en los subsistemas de Open Modelica y Simulink se representan los casos que se muestran en la Tabla 2.8. Se va a presentar el sistema de Simulink pero hay que tener en cuenta que el de Open Modelica es análogo y se obtienen los mismos resultados.

Tabla 2.8 Resultados de la comprobación del circuito de puertas lógicas de Simulink..

ROE	LIA	UR	ROA	LIE	FCC 1
0	0	0	0	0	0
1	0	0	0	0	0
0	0	1	0	0	1
1	0	0	1	0	1

En la Figura 2.13 se presentan los resultados de este subsistema de puertas lógicas. Se comprueba que su funcionamiento es correcto ya que se corresponde con el resultado de la Tabla 2.8.

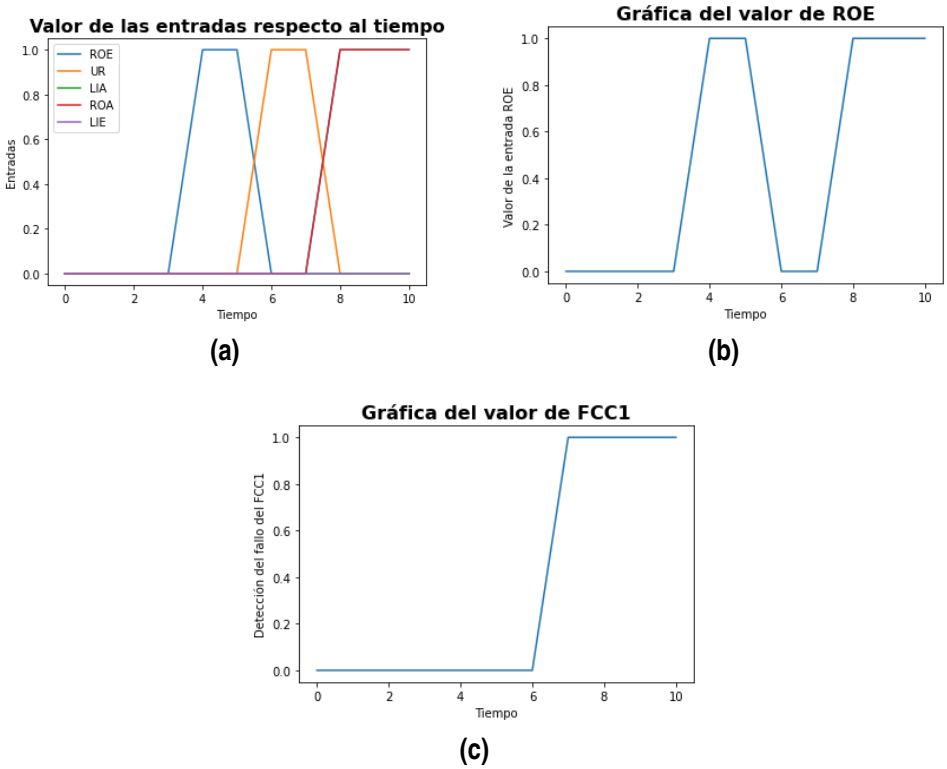


Figura 2.13 Gráficas de (a) los valores de todas las entradas, (b) valores de la variable 'ROE'y (c) valores de la variable de salida 'FCC 1'.

En la Figura 2.13 (a) se presentan todas las variables de entrada con respecto al tiempo. Si se observa la gráfica entre los segundos 8 y 10 no se puede apreciar bien, pero la variable ROA y la ROE se superponen y toman el valor 1, tal y como se ve en la Figura 2.13 (b).

Una vez que se ha comprobado el correcto funcionamiento de todos los subsistemas, se procede a unificar el sistema y realizar la Co-Simulación. Se parte del test unitario de sistemas sencillos y se modifica para poder simular el sistema completo. Así, los tres FMU funcionarán conjuntamente aunque se hayan creado en tres programas diferentes.

```
import libraries

class SimpleTestCase(unittest.TestCase)

    def setUp(self):
        set up options
        load fmu model from Simulink
        load fmu model from Open Modelica
        load fmu model from Hopsan
        initialize each fmu
        print input list
        print output list
        set initial variables

    def tearDown(self):
```

```
        unittest.TestCase.tearDown(self)

    def Function_name(self)
    t = 0.0;
    dt = 0.01;
    while(True)

        if t % 1 < 0.05:
        Simulink fmu's simulation. (different cases at specific instants
             of time with different values of input variables)
        print the result of every case
        t +=dt
        it t>10: break # while loop
        breakout time
        assert

        if t % 1 < 0.05:
        Open Modelica fmu's simulation. (different cases at specific
           instants of time with different values of input variables)
        print the result of every case
        t +=dt
        it t>10: break # while loop
        breakout time
        assert

        if t % 1 < 0.05:
        Hopsan fmu's simulation. (different cases at specific instants
           of time with different values of input variables) In this
           case input variables are Open Modelica and Simulink's output
           variables.
        print the result of every case
        t +=dt
        it t>10: break # while loop
        breakout time
        assert

    #If there is some plots result
    plot

if __name__ == "__main__"
    unittest.main() #run all test
```

Tras programar el test unitario correspondiente a la Co-Simulación que se quiere realizar, se preparan los casos en los que se va a evaluar el sistema. Los diferentes casos, junto con los tiempos en los que se van a dar se puede comprobar en las Tablas 2.9 y 2.10.

Tabla 2.9 Valores de las entradas y salidas de los subsistemas de Open Modelica y Simulink para la Co-Simulación.

	OPEN MODELICA						SIMULINK					
TIME	LOE	LR	RIA	RIE	LOA	FCC2	ROE	UR	LIA	ROA	LIE	FCC1
0-2	0	0	0	0	0	0	0	0	0	0	0	0
2-4	0	0	0	1	0	0	0	0	0	0	1	0
4-6	0	1	0	0	0	1	1	0	0	0	0	0
6-8	1	0	0	0	0	0	0	1	0	0	0	1
8-10	1	0	0	0	1	1	1	0	0	1	0	1

Tabla 2.10 Valores de las entradas y salidas de los subsistemas de Hopsan para la Co-Simulación.

HOPSAN	FCC1	0	0	0	1	1
	FCC2	0	0	1	0	1
	FTS	0	0	0	0	1

Los resultados de la Co-Simulación se van a mostrar en diferentes gráficas. En la Figura 2.14 se puede ver gráficamente el valor de las entradas indicado en la Tabla 2.9, separadas por bloques. La detección del fallo de cada uno de los FCCs se muestra en la Figura 2.15 (a). Por último, se puede apreciar que la Co-Simulación funciona correctamente ya que detecta el fallo total del sistema al producirse el fallo de los dos ordenadores de control a la vez. El resultado final se aprecia en la Figura 2.15 (b).

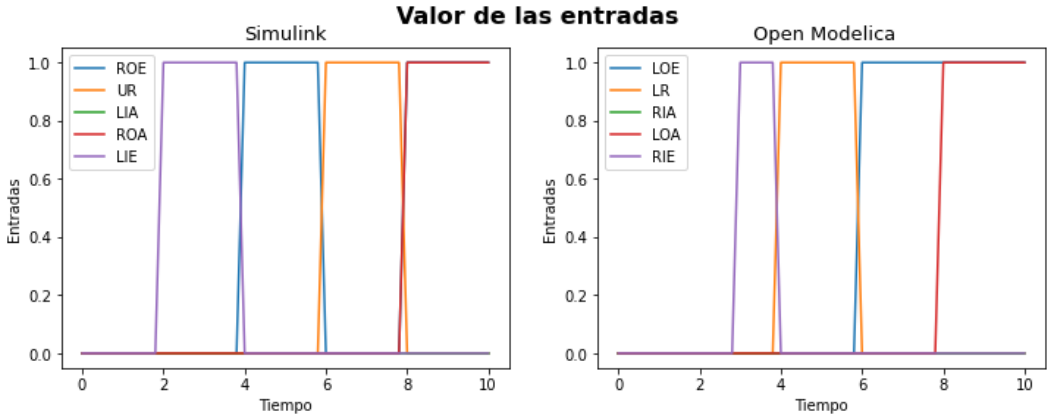
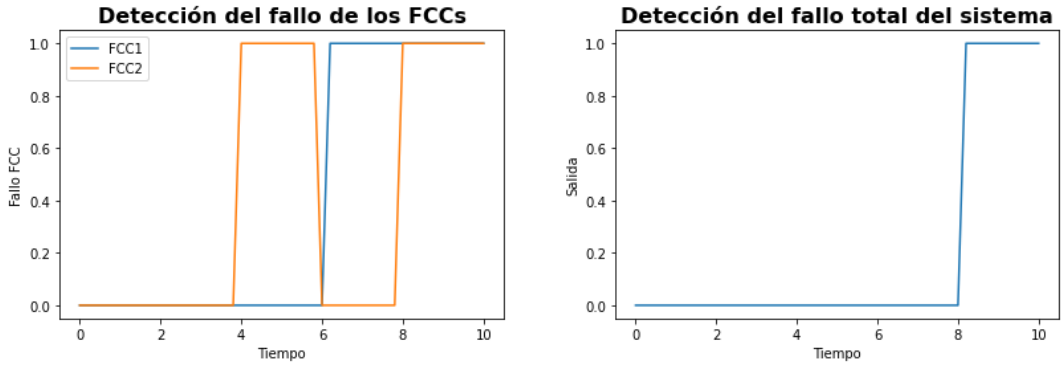


Figura 2.14 Valor de las entradas respecto del tiempo..



(a) (b)

Figura 2.15 Gráficas de (a) la detección del fallo en los FCCs, (b) la detección del fallo total del sistema .

Una vez que se ha conseguido realizar la Co-Simulación satisfactoriamente, se ha querido analizar el rendimiento de la misma, en términos del tiempo que tarda en ejecutarse. Para poder obtener más datos, se ha creado el mismo sistema de la Co-Simulación en Simulink, Open Modelica y Hopsan, sin dividirlo en subsistemas. Se han realizado 50 simulaciones de cada caso en las mismas condiciones. En la Tabla 2.11 se presenta el tiempo medio de simulación y el tiempo medio de la ejecución del test unitario completo, exceptuando el tiempo en el que se crean las gráficas mostradas anteriormente .

Tabla 2.11 Resultado estadístico de los tiempos medios de simulación y el test unitario del FBW.

Plataformas	Co-Simulación	Simulink	Hopsan	Open Modelica
Tiempo medio de ejecución de la simulación del modelo (s)	0.0233	0.0087	0.0189	0.0091
Tiempo medio de ejecución del test unitario (s)	0.6903	0.1148	0.4461	0.7424

Si se analiza la Tabla 2.11 se pueden concluir algunos parámetros que no se preveían antes de este análisis. En primer lugar cabe destacar que el sistema completo creado en Simulink es mucho más eficiente que los demás, en términos de tiempo, tal y como ocurría con la puerto lógica AND. En concreto, en términos del tiempo de ejecución del test unitario es un 83 % más rápido que la Co-Simulación, un 84 % más eficiente que el modelo de Open Modelica y un 74 % más rapido que el FMU de Hopsan.

Al observar el tiempo de ejecución de la simulación es lógico que el que más tarde sea el de la Co-Simulación, ya que está usando tres archivos FMU distintos y haciendo simulaciones en cada uno de ellos. Específicamente, en el tiempo de la simulación, la Co-Simulación es un 59 % más lento que el modelo de Open Modelica, un 60 % más lento que el sistema de Simulink y un 15 % menos eficiente que el FMU del Hopsan. Además, hay que tener en cuenta que las salidas de dos de esos FMUs son las entradas del tercero. Por tanto, es de esperar que sea el menos eficiente. Siguiendo con el tiempo medio de la ejecución de la simulación a la Co-Simulación, le sigue el FMU de Hopsan, lo que era de esperar ya que en el análisis que se hizo en la puerta lógica AND era el que más tiempo tardaba en simularse. Luego el más eficiente es el archivo de Simulink y luego el de Open Modelica.

En cambio, al observar el tiempo de ejecución del test unitario completo, se observa que el que más tiempo tarda es el de Open Modelica, tal y como pasaba en el análisis de la puerta lógica AND. Este hecho era más inesperado, ya que como la Co-Simulación tiene que cargar los tres fmu, leer las variables de cada uno y simularlos, se esperaba que fuera el que más tardara. En cambio, se puede ver cómo al usar otros programas como Simulink y Hopsan conjuntamente con Open Modelica se ha mejora el tiempo de ejecución del test unitario. En este caso concreto no es muy significativa la mejora de tiempo, se trata de un 7 %, pero si el sistema fuera mucho más complicado si que se podría apreciar más esa rapidez. Como en el tiempo medio de simulación, en el de ejecución del test Simulink sigue siendo el más eficiente, como se comentó con anterioridad.

Otra de las comparaciones que se puede hacer es el peso de los archivos FMU, para ello se presenta la Tabla 2.12 que contiene el valor del tamaño de cada FMU. Estos FMU se han descargado con las mismas opciones para poder hacer la comparación más exacta posible.

Tabla 2.12 Tamaño de los FMU de la Co-Simulación y la simulación con 1 sola plataforma.

	Co-Simulación			Simulación en 1 sola plataforma		
	Simulink	Open Modelica	Hopsan	Simulink	Open Modelica	Hopsan
Tamaño (KB)	84	346	1780	91	351	1782

Se puede ver que el tamaño de los FMU de Hopsan apenas varía entre el FMU que solo representa una puerta AND con respecto del FMU que contiene el circuito completo representado en el Figura 2.11. En Simulink se puede ver que aumenta el tamaño en un 7.6 %, principalmente porque el archivo en el que se describen las variables del circuito aumenta. Sin embargo, no supone una gran diferencia, ya que Simulink solo guarda en ese archivo las entradas y salidas, por ende, el tamaño va a aumentar en función de el incremento del número de salidas y entradas. Con el archivo de Open Modelica ocurre prácticamente lo mismo que con el archivo de Simulink, aumenta su tamaño en un 1.5 %.

Si se tiene en cuenta el peso Hopsan es el que más pesa, aunque este peso cambia muy poco entre un sistema muy sencillo, a otro más complicado. Open Modelica y Simulink si que pesan más cuanto más complicado sea el sistema que se ha exportado mediante el estándar FMI.

Aunque como es de esperar, los archivos de la que forman el circuito pesan mucho más conjuntamente que los que son el sistema completo en una única plataforma. Estos pesos, junto con las medidas de los tiempos que tardan en ejecutarse, puede ser un factor a tener en cuenta a la hora de elegir los diferentes programas que van a formar parte de la Co-Simulación.

Conclusiones

En esta sección del trabajo se ha podido comprobar como mediante la Co-Simulación se puede trabajar con diferentes programas para crear un solo sistema. Para ello es interesante conocer bien los programas con los que se van a generar las diferentes secciones del sistema completo, ya que pueden surgir problemas que serán más complicados de solucionar una vez que se hayan unido todas las partes mediante la Co-Simulación.

Es interesante que, a través la Co-Simulación se pueda crear un solo sistema que se ha dividido previamente en varios subsistemas y que se han generado por diferentes plataformas de simulación. Esta es la principal ventaja de la Cosimulación, el hecho de poder crear subsistemas independientes y que funcionen correctamente cuando se unen. Además, se ha podido comprobar que no supone un problema usar una plataforma de pago como es Simulink conjuntamente con dos "*open source*",

como son Open Modelica y Hopsan.

El análisis que se ha hecho sobre el rendimiento de la Co-Simulación del sistema con FMUs de tres plataformas distintas con respecto de las simulaciones del mismo sistema, generado con tan solo una plataforma, ha revelado que la diferencia de tiempo entre unos y otros. Exceptuando Simulink, no es tan grande en este sistema concreto. Si se compara con Simulink si que es mayor el tiempo medio de ejecución del test unitario, ya que se han usado otros dos FMU que son más lentos de simular.

2.2.2 Sistema FADEC

Otro de los sistemas críticos que se pueden encontrar en un avión es el FADEC "*Full Authority Digital Engine Control*". Este sistema se encarga de regular el motor y todos los sistemas relacionados con él, es decir, calcula la cantidad de combustible necesario para cada operación de vuelo, se encarga del arranque del motor, presenta los datos en cabina al piloto, etc.

Al ser otro sistema crítico, es muy importante que esté diseñado a prueba de fallos. Por tanto, es crucial la detección de fallos del mismo. En la electrónica de la aeronave cabe destacar la redundancia de los sistemas y de los canales de comunicación, para tener una mayor seguridad ante fallos que puedan surgir. Concretamente el sistema de detección de fallos para el FADEC se presenta en la Figura 2.16.

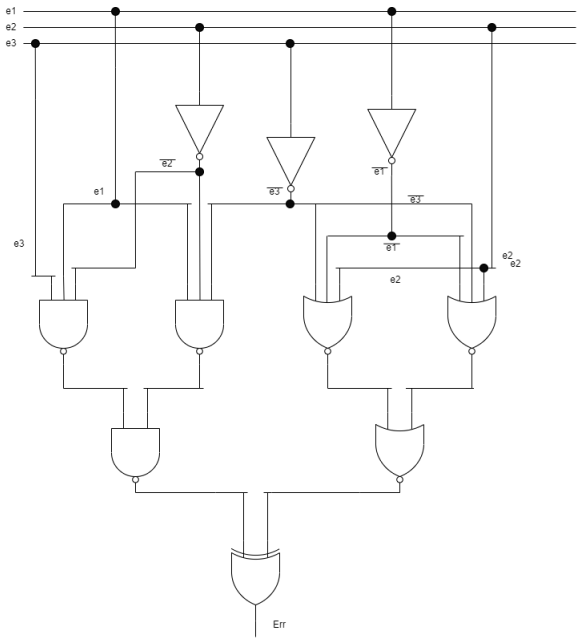


Figura 2.16 Circuito de detección de fallos del FADEC.

Si se observa el circuito se puede ver como se ha implementado lógica inversa y sistemas redundantes para incrementar la seguridad del propio sistema. Además, el circuito es más complejo que el que se ha analizado anteriormente.

En este caso se quiere hacer una Co-Simulación con cuatro FMU generados desde Simulink, ya que como se ha visto hasta el momento, es el más eficiente. Esta situación podría presentarse si una empresa está encargada de diseñar un sistema muy complejo y se dividen en grupos con subsistemas que van a crear de manera independiente, pero al ser la misma empresa podrían usar el

mismo programa y comprobar cómo se haría la Co-Simulación.

En la Figura 2.17 se puede ver la división que se ha realizado para generar los diferentes FMU con Simulink.

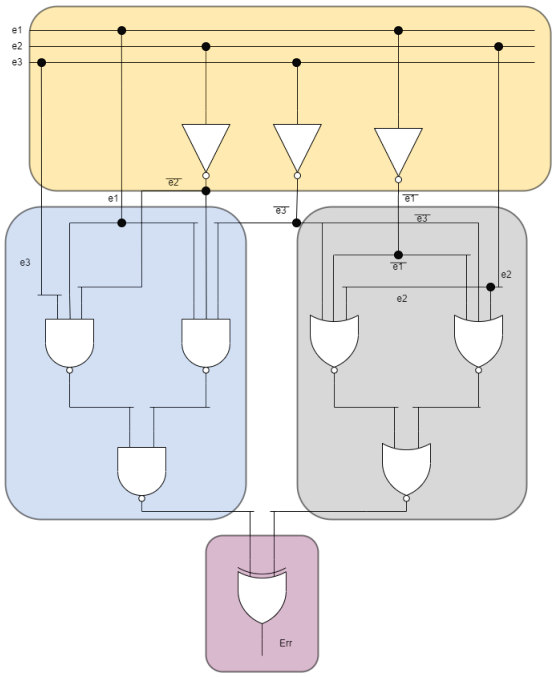


Figura 2.17 División para la Co-Simulación del sistema de detección de fallos del FADEC.

Una vez decidida la división se crean los diferentes archivos en Simulink y se exportan mediante el estándar FMI. Al comenzar a crear el test unitario y no tener una tabla de verdad, a partir de la cuál poder diseñar varias situaciones, se optó por calcular la función "*boolenana*" de salida. Si se observa el circuito, se puede intuir que la función va a ser complicada y extensa. En concreto la ecuación que se obtiene es la siguiente:

$$Err = \overline{\overline{e_1\bar{e}_2e_3}\cdot\overline{e_1\bar{e}_2\bar{e}_3}}\cdot\overline{\overline{\bar{e}_1+e_2+\bar{e}_3}+\overline{\bar{e}_1+e_2+\bar{e}_3}}+\overline{\overline{e_1\bar{e}_2e_3}\cdot\overline{e_1\bar{e}_2\bar{e}_3}}\cdot\overline{\overline{\bar{e}_1+e_2+\bar{e}_3}+\overline{\bar{e}_1+e_2+\bar{e}_3}} \tag{2.5}$$

Al ser la ecuación final tan extensa es complicado plantear los casos en los que se va a evaluar el sistema en el test unitario. Para simplificarla se ha decidido usar el programa *Boole Deusto*.

Boole Duesto es una herramienta software enfocada en la electrónica digital. Este programa permite trabajar con circuitos combinacionales y secuenciales. Boole Duesto permite introducir una tabla de verdad o una función "*boolenana*" y calcula el mapa de Karnaugh, una la función "*boolenana*" simplificada y muestra como sería el circuito para esa simplificación. En este caso se introdujo la función booleana 2.5 y se obtuvo la siguiente simplificación.

$$Err = \bar{e}_1 + e_2 + e_3 \tag{2.6}$$

A partir de esta función simplificada se pueden crear los diferentes casos en los que se va a evaluar el sistema en el test unitario. Al haber utilizado Boole Deusto también se ha obtenido el circuito de la función simplificada, el cual se muestra en la Figura 2.18.

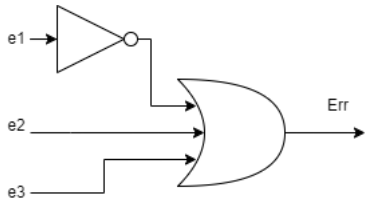


Figura 2.18 Circuito simplificado de la detección del error en el FADEC.

Si se compara la Figura 2.18 con la Figura 2.16, se puede ver que es mucho más sencillo pero carece de la redundancia tan importante en la industria aeroespacial. Para detectar los errores en sistemas críticos hay que asegurarse que, efectivamente, el error está ocurriendo, y que no es una alarma falsa o que el propio circuito de detección es el que falla. Por tanto, en el sector aeroespacial, aunque los circuitos sean más complejos, se incrementa el nivel de seguridad de la aeronave. Por ende, esa complejidad añadida para tener dos circuitos que detectan lo mismo y que usan lógica inversa es un beneficio para la industria.

A nivel teórico es interesante trabajar con el circuito reducido, para ver que da el mismo resultado y el tiempo de ejecución que tiene en comparación con el más complejo.

Tras obtener la función "*boolenana*" simplificada, en la Tabla 2.13 se muestran los casos que se van a evaluar durante la ejecución del test unitario.

Tabla 2.13 Tabla de verdad de los casos que se van a evaluar en el test unitario.

e_1	e_2	e_3	Err
0	0	0	1
o	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

En la Figura 2.19 se puede ver el valor de las tres entradas respecto del tiempo transcurrido en la ejecución del test unitario y en la Figura 2.20 se puede ver como el resultado de la Co-Simulación, el cual coincide con la Tabla 2.13, por tanto es correcto.

(a) (b) (c)

Figura 2.19 Gráficas del (a) valor de e1, (b) valor de e2 y (c) valor de e3. .

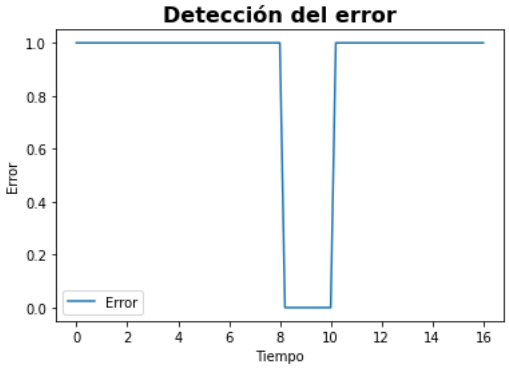


Figura 2.20 Detección del error del sistema en función del tiempo.

El test unitario que se ha usado para realizar la Co-Simulación se diferencia con los anteriores, ya que, todos los bloques, excepto el que representa las puertas lógicas NOT, dependen de la salida de los anteriores. Por tanto es un poco más complejo que los anteriores.

Para seguir analizando esta Co-Simulación, se va a comparar con una simulación del mismo sistema, creado en su totalidad en un mismo archivo de Simulink, sin dividirlo en subsistemas. También, teóricamente, puede ser interesante compararla con el circuito simplificado obtenido a partir de Boole Deusto. Para ello, como en los casos anteriores, se han realizado 50 simulaciones de cada archivo FMU y de la Co-Simulación y se han calculado los tiempos medios de ejecución del test unitario completo, y el tiempo medio de la simulación propiamente dicho. Los resultados obtenidos se presentan en la Tabla 2.14.

Tabla 2.14 Resultado estadístico de los tiempos medios de simulación y el test unitario del FADEC.

Plataformas	Co-Simulación	Simulink:sistema completo	Simulink: sistema simplificado
Tiempo medio de ejecución de la simulación del modelo (s)	0.0222	0.0079	0.0073
Tiempo medio de ejecución del test unitario (s)	0.4007	0.1113	0.1076

Si se observa la Tabla 2.14 se puede ver como la Co-Simulación es la que más tarda, tanto en el tiempo de ejecución del test unitario completo como el tiempo de simulación. En concreto, la Co-Simulación es un 72 % más lenta que el sistema completo de Simulink y un 73 % más lento que el FMU del sistema simplificado. Eso se debe a que hay que cargar cuatro archivos FMU distintos, leer cada una de sus variables de entrada y salida y en la simulación las salidas de un

FMU son las entradas de otro, están los cuatro relacionados. Por esa razón, es el que más tarda, aunque la diferencia no es demasiado grande, ya que son décimas y centésimas de segundos entre unos resultados y otros, es un tiempo muy pequeño.

Si se sigue analizando la Co-Simulación del sistema FADEC y se compara con la Co-Simulación del sistema FBW, es decir, si se examinan los resultados de la Co-Simulación de la Tabla 2.11 con los de la Tabla 2.14, se puede ver una mejora en el tiempo medio. Esta Co-Simulación es un 41 % más rápida que la Co-Simulación del sistema FBW. Esto es debido a que Simulink es el más eficiente con respecto a los otros dos programas en términos de tiempo. En consecuencia, aunque este circuito conste de más puertas lógicas y se haya dividido en más subsistemas, el tiempo de ejecución del test unitario es bastante menor que el del circuito de FBW. Por lo que, a la hora de hacer una Co-Simulación puede ser útil saber qué programa es más eficiente y si es beneficioso o no que la esa Co-Simulación se realice con archivos generados por un mismo programa, o por varios programas distintos.

Si se observan los resultados obtenidos de las simulaciones del sistema completo en un solo archivo FMU generado por Simulink y el sistema simplificado, se puede ver que tardan menos que la Co-Simulación. Esta diferencia se debe a que solo hay que cargar un archivo que tiene tres entradas y una sola salida, por tanto a la hora de cargar ese FMU el tiempo es menor. Además en la simulación no hay que enlazar las salidas de un archivo con las entradas de otro sino que como es un sistema único tarda muy poco.

Una curiosidad que se puede ver en la Tabla 2.14 es que los tiempos medios, tanto el de la ejecución del test unitario como el de la simulación del circuito completo en un único FMU, y el circuito simplificado ambos generados por Simulink, son muy parecidos, en concreto, la diferencia es de un 3.3 %. Por lo que se comprueba como, en las simulaciones software, la redundancia en los circuitos electrónicos de un avión no incrementa el tiempo de ejecución en exceso, con respecto de un mismo sistema pero mucho más simplificado. Por lo que, se aumenta la seguridad de las operaciones del avión aumentando la complejidad del circuito, pero el incrementar la complejidad de los circuitos no afecta demasiado al tiempo de detección del fallo en el sistema.

Otra de las peculiaridades que se ha podido observar durante este análisis es que todos los FMU generados, es decir, tanto el del sistema completo, como el simplificado y los que forman las divisiones del sistema, pesan lo mismo 61 KB. Excepto el archivo que representa la puerta XOR que pesa 60 KB. Hay que destacar que en el archivo FMU solo se ha introducido la carpeta "*binaries*" y el archivo "*modelDescription.xml*". El código no está, ya que como solo interesaba la simulación, para que sea más rápida, cuantos menos archivos y carpetas tenga el FMU, más rápido será carga el archivo para el test unitario. Con esto se comprueba una vez más que en Simulink el número de puertas lógicas apenas influye en el tamaño del FMU, si no se descarga el código ni la fotografía, ya que solo guarda las variables de entrada y salida.

Conclusiones

En esta sección se ha comprobado que la Co-Simulación será más eficiente o menos según los programas que se usen. En este caso, como Simulink era el más eficiente en términos de peso y tiempo de ejecución ha hecho que la Co-Simulación sea más rápida que las anteriores, en concreto un 41 % más efiencite.

Otra ventaja que se ha podido apreciar en el análisis de este sistema, es que la Co-Simulación también puede ser muy útil con archivos generados por el mismo programa. Ya que si hay un

proyecto en una empresa, en el que hay que diseñar un circuito de puertas lógicas muy complejo, es más sencillo si se divide en subgrupos y cada uno diseña un sistema independiente, sin fallos y luego se unen. Esto puede agilizar el trabajo de la empresa y hacer que el proyecto mejore porque hay un grupo reducido de personas encargándose de una pequeña parte del circuito, teniendo la idea general del mismo, así se puede optimizar cada subsistema. Además, como son independientes se pueden comprobar los fallos antes de unir todos los FMUs para realizar la Co-Simulación, evitando así que se produzca un fallo en el circuito completo, sería más complejo de saber dónde está y cómo solucionarlo.

Asimismo, se ha demostrado la importancia de la Co-Simulación, puesto que los circuitos en el sector aeroespacial suelen ser más complejos que en otros sectores por la importancia de la redundancia de sistemas, con circuitos iguales y circuitos diseñados con una lógica inversa o complementaria. Gracias a esta complejidad añadida por los sistemas replicados, la seguridad del avión aumenta, que es uno de los objetivos más importantes que tiene la industria aeroespacial.

Resumiendo, se puede llegar a la siguiente conclusión: si se quiere aumentar la seguridad de las aeronaves hay que aumentar la seguridad de la aviónica embarcada, para ello es necesario la redundancia de sistemas y canales de comunicación, por lo que los circuitos electrónicos son más complejos, ya que constan de la réplica de los mismo siguiendo la misma lógica y añadiendo lógica inversa. Como los circuito electrónicos son cada vez más complejos la Co-Simulación se vuelve una herramienta muy útil en este sector, ya que es más sencillo trabajar con subsistemas que con circuitos muy grandes y muy complicados.

3 Conclusiones finales

Uno de los objetivos más importantes en el sector aeroespacial es la seguridad de la aeronave. Para conseguir el grado de seguridad requerido, los sistemas, circuitos electrónicos y canales de comunicación tienen que ser redundantes. Esa redundancia se puede conseguir de dos formas, replicando los circuitos y sistemas varias veces, o creando circuitos análogos pero con una lógica inversa o complementaria. En muchos casos, en los sistemas críticos del avión se tienen que poner aprueba las dos técnicas de redundancia para asegurar que no ocurre un fallo y en caso de que ocurriera, detectarlo correctamente en el menor tiempo posible. Todo esto hace que los circuitos embarcados tomen un mayor nivel de complejidad y son más grandes. La Co-Simulación permite que esos considerables y difíciles circuitos se dividan en pequeñas partes, se generen como sistemas independientes y se puedan unir, una vez terminados, para que el sistema completo funcione correctamente.

La Co-Simulación, por tanto, es una técnica innovadora y con muchas ventajas en varios sectores de la industria. Asimismo, al poder trabajar con subsistemas independientemente, pertenecientes a un modelo complejo, las personas encargadas de esta parte del sistema pueden centrarse en que no se produzca ningún tipo de fallo, antes de unir todas esas partes en el sistema completo. Esto permite un mayor ritmo de producción y un nivel de seguridad mayor, al poder detectar fallos en las pequeñas divisiones antes que en el sistema complejo. Además, esta división permite el poder optimizar cada parte del circuito, al ser una más pequeño es más fácil detectar alguna optimización que si se mira de manera general.

Otra de las ventajas a destacar, es que la Co-Simulación permite que varios archivos generados desde plataformas totalmente distintas, puedan trabajar de manera conjunta sin ningún tipo de problema. Lo que es muy importante, puesto que no todos los sistemas y circuitos de una aeronave los diseña una misma empresa. Con este método no es necesario que todas las empresas que trabajan en una aeronave utilicen las mismas plataformas de modelado y simulación, la única restricción es que tienen que ser compatibles con un estándar para su exportación. Además a lo largo de este trabajo, se ha comprobado que la Co-Simulación funciona correctamente uniendo diferentes archivos de plataformas de coste y *open source* entre sí.

Además, la Co-Simulación permite introducir o cambiar varias partes de circuitos o sistemas de la aeronave sin tener que modificar todo el circuito. Así, se consigue que las partes que funcionen correctamente y no necesiten una mejora se puedan conservar sin problema, y que solo se sustituyan otras independientemente del programa con el que se han creado. Se ha comprobado que estas divisiones de los circuitos complejos, hacen que el sistema completo sea más flexible a la hora de hacer cambios o de diseñar sistemas muy difíciles.

Una de los aspectos que hay que tener en cuenta para realizar la Co-Simulación, son las herramientas a partir de las cuales se van a crear los diferentes modelos que formaran parte del sistema completo. Concretamente, en este documento, se han explicado tres plataformas de modelado y simulación que se han usado para realizar la Co-Simulación. Recogiendo todos los resultados y datos obtenidos a partir del desarrollo realizado previamente, se puede decir que de las tres herramientas usadas (Simulink, Open Modelica y Hopsan) la más eficiente en términos de peso y tiempos de ejecución de test unitarios es Simulink. Esta conclusión resulta es previsible, ya que es la plataforma mejor optimizada de las tres usadas, aunque también se ha comprobado cómo para algunos dominios no funciona bien, por los requisitos de exportación. Open Modelica es el más lento de los tres programas, ya que tarda más en cargar el archivo FMU en Python. Sin embargo, en sectores, diferentes al de la electrónica, como el eléctrico o hidráulico, se ha comprobado que no da fallos, y se pueden implementar los circuitos tal y como se diseñaron. Hopsan es el que genera los archivos más pesados, aunque no tarda tanto en ejecutarse como Open Modelica. Además, es el más sencillo, en términos de la exportación de FMU. El problema de Hopsan es el cambio de nombre de las variables de entrada y salida, y en otros dominios, diferentes al electrónico, hay que tener en cuenta los puertos de los componentes y saber que no se pueden unir directamente elementos con los mismos puertos.

En consecuencia, si los circuitos que se quieren diseñar son de puertas lógicas, el programa más eficiente es Simulink, ya que son muy fáciles de implementar y exportar. Además, son los que menos tamaño tienen y lo que menos tardan en ejecutarse. Si hubiera que elegir entre plataformas *open source* si se quiere que la simulación sea muy rápida es preferible usar Hopsan, en cambio, si se espera que el FMU ocupe menos memoria se escogería Open Modelica. Sin embargo, si se quiere implementar circuitos eléctricos o de otro dominio, el que menos inconvenientes tiene es Open Modelica, va a tardar más que los otros dos en simularse, aunque la diferencia no es mucha, pero a la hora de crear el modelo y exportarlo con Open Modelica va a ser más sencillo y rápido.

En relación con la Co-Simulaciones realizadas, se ha comprobado que los archivos generados por Simulink y Hopsan van a reducir el tiempo de ejecución del test unitario de Co-Simulación al unirlo a otro de Open Modelica, que tarda más en ejecutarse. También se ha verificado que cuanto menos pese el archivo FMU generado, menor va a ser el tiempo de ejecución del tipo test. Por ende, es importante elegir las opciones que hacen posible que el FMU tenga un menor tamaño en Open Modelica y en Simulink, puesto que en Hopsan va a ser siempre de un tamaño parecido. Otro de los aspectos que se ha verificado con las Co-Simulaciones, es que se pueden crear varios archivos FMU desde un mismo programa y unirlos sin problema, sin que afecte demasiado al tiempo de ejecución.

Se ha comprobado que las Co-Simulaciones son más lentas que la simulación del circuito completo en algunas ocasiones. Si el sistema completo es de Open Modelica y se compara con una Co-Simulación en la que no todos los FMUs son de Open Modelica, se comprueba que se mejora el tiempo de ejecución del test unitario. Aunque, dependiendo del circuito, la Co-Simulación sea más lenta, las ventajas que supone el trabajar con subsistemas independientes y detectar los fallos antes de tener el sistema completo, como el poder trabajar diferentes equipos con técnicas y herramientas distintas, supone una gran mejora en la producción del sector.

Uno de los resultados más importantes que se ha obtenido en este trabajo, es que los sistemas complicados y extensos, por la redundancia de los sistemas y circuitos, al compararlos con circuitos que tienen las mismas funciones pero están simplificados, influyen muy poco en el tiempo de ejecución de los mismos. Por tanto, la redundancia de los sistemas y circuitos complica los circuitos y mejora así la seguridad de la aeronave, pero además no perjudica apenas en los tiempos de simulación. Se van a seguir obteniendo tiempos muy parecidos a los que se obtienen con circuitos simplificados.

Hay que tener en cuenta que en este trabajo a lo que se hace referencia es a nivel de simulaciones y software, no hardware. Ya que a nivel de hardware, las puertas lógicas tienen un retraso, por tanto si aumenta el número de puerta lógicas aumenta el tiempo de ejecución. Si se quisiera hacer una simulación más fiel a la realidad se podría analizar las diferentes puertas lógicas usadas en este trabajo, con su tiempo de retraso de la señal.

Con respecto al símil eléctrico se han obtenido varias conclusiones. En primer lugar, la implementación de un circuito eléctrico es mucho más sencilla que la de un sistema hidráulico. Los componentes del sistema hidráulico son más complejos, y algunos de ellos no indican las unidades de los parámetros que los definen. Por tanto, ajustar los parámetros de los diferentes componentes del sistema hidráulico es mucho más complicado que los cálculos que hay que hacer para los valores de los elementos del circuito eléctrico. Asimismo, la exportación con el estándar FMI desde Open Modelica, presenta menos errores con el circuito eléctrico que con el sistema hidráulico.

Se ha podido comprobar como los resultados obtenidos por la carga y descarga del condensador se asimilan más a la realidad del llenado y vaciado de un tanque de agua, o de combustible, que la gráfica obtenida del sistema hidráulico. Esto se debe a la complejidad de los elementos que conforman el sistema hidráulico, así como a un posible fallo de fuga de memoria en Open Modelica, por el tipo de sistema que es. Al ser elementos más complejos, definidos con muchas más ecuaciones, es más complicado que los resultados obtenidos se parezcan tanto a la realidad como los del condensador.

En términos de rendimiento, también se ha comprobado que el tiempo de ejecución del test unitario es menor el del circuito eléctrico que el del sistema hidráulico, en concreto un 14 % más rápido. Ya que hay que tener en cuenta que el FMU del sistema hidráulico es de mayor tamaño. Por tanto, si se quiere realizar una Co-Simulación con varios circuitos, es mucho más fácil utilizar el símil eléctrico que el sistema hidráulico, y además el tiempo de ejecución del test unitario sería menor.

En conclusión, la Co-Simulación es una técnica muy útil actualmente en varios sectores de la industria, pero en específico en el sector aeroespacial, ya que la aviónica embarcada está en continuo cambio y mejora, para optimizar el nivel de seguridad de las aeronaves. Con esta técnica es posible mejorar todos los circuitos embarcados, así como la detección de fallos de los diferentes sistemas críticos. También, permite tener una mayor cooperación entre diferentes empresas, científicos e ingenieros, haciendo así que el conocimiento de la aviónica pueda avanzar más rápidamente, al poder unir diferentes partes de un mismo sistema independientemente de la herramienta con la que se haya generado. Otra de las ventajas que tiene es que, con este método la renovación se varias partes de sistemas es mucho más sencilla, ya que se puede conservar los subsistemas antiguos y solo cambiar una pequeña zona del circuito. El símil eléctrico también es muy importante a la hora de la simulaciones, ya que se pueden crear circuitos sencillos que emulen un circuito más complicando. Asimismo, no solo permite que los modelos simulados sean más sencillos de implementar, sino que además puede que se eviten las fugas de memoria y se obtengan resultados fieles a la realidad, reduciendo consigo el tiempo de ejecución de los test unitarios y el tamaño de los archivos FMU.

Por tanto, la Co-Simulación y el símil eléctrico son herramientas que pueden mejorar el sector aeroespacial, en muchos aspectos, y que actualmente siguen avanzando siendo útiles para todas las empresas e ingenieros dedicados a esto.

4 Líneas futuras

Al ser la Co-Simulación y el símil eléctrico dos técnicas que actualmente se usan en el sector aeroespacial, se van a seguir desarrollando, pudiendo así mejorarse. Por tanto, al ser dos métodos en continuo avance y crecimiento, se pueden seguir estudiando, así como analizándolas para hacer un uso más óptimo de las dos.

Algunos trabajos que se podrían desarrollar a partir de este, son los que se presentan a continuación.

- Una de las posibles líneas futuras sería utilizar el método de Co-Simulación con circuitos de electrónica secuencial. Para ello, habría que saber si con las plataformas usadas se pueden implementar circuitos con biestables y cómo sería la unión entre los diferentes componentes. Podría ser un buen análisis, sobre todo, teniendo en cuenta que hay muchos sistemas de la aeronave que necesitan de memoria, es decir, que necesitan circuitos secuenciales. Uno de estos sistemas, son los buses de datos, en los cuales se está trabajando mucho para optimizarlos y que la comunicación entre toda la aviónica de la aeronave sea más segura, rápida y eficiente.

- Otro análisis que podría ser interesante sería, realizar la Co-Simulación de un circuito eléctrico, ya sea propio de la aeronave o un símil eléctrico. Se podría comparar las diferentes velocidades, así como comparar el tiempo y el tamaño de los FMU con los circuitos electrónicos. Así se podría concluir, si es útil utilizar el método de la Co-Simulación con circuitos eléctricos, en el sector aeroespacial.

- A nivel de hardware, se podría probar cómo funciona la Co-Simulación. Asimismo, se puede analizar los retrasos reales de los diferentes componentes electrónicos, y determinar realmente si mediante la Co-Simulación se tarda más o menos, que haciendo el circuito completamente en una sola plataforma de modelado. También, se podría comprobar cómo afecta la redundancia de los circuitos al tiempo de ejecución en sistemas hardware.

- Con respecto al símil eléctrico, se pueden plantear otros sistemas hidráulicos y neumáticos, un poco más complicados, y emularlos mediante un circuito eléctrico. A nivel de implementación será más sencillo los circuitos eléctricos que la de los sistemas neumáticos o hidráulicos, pero puede ser más complicado diseñar esos circuitos eléctricos para imitar la acción real de los sistemas. Además se podría comprobar si la disminución de carga de computación se sigue dando con circuitos más complejos.

Apéndice A
Metodología de Trabajo

En esta parte del documento se va a describir la metodología de trabajo seguida, haciendo una comparación entre varias.

En la gestión de los proyectos, la metodología se refiere a técnicas, procedimientos y métodos que se llevan a cabo para llevar a cabo un proyecto. Estas técnicas pueden mejorar y afianzar la posición de una empresa en el mercado.[10]

A continuación, se describirán tres de las metodologías más utilizadas actualmente y se compararán, justificando la utilizada para realizar este trabajo. Se van a describir, la metodología *Waterfall* o modelo de cascada, que es una más tradicional, y otras dos, la metodología ágil y la Scrum, esta última es un método ágil pero con un enfoque específico.

A.1 Metodología *Waterfall*

La metodología *Waterfall* o modelo de cascada, fue implementada en 1970 por Winston W. Royce. A partir de ese momento, muchas de las empresas de la época la utilizaron para el desarrollo de sus proyectos.

Con este método se trabaja con un proyecto como una unidad, con grandes dimensiones y una estructura determinada. Siendo un proceso secuencial, en el que solo se va en una dirección.[10]

Las diferentes fases de esta metodología se presentan en la Figura A.1

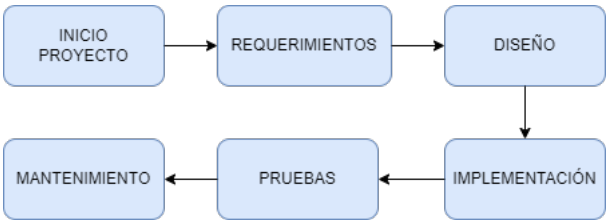


Figura A.1 Fases de la metodología *Waterfall*.

En la primera fase, los requerimientos, se define el proyecto y se aprueba su inicio. A continuación, en el diseño se planifica la gestión del mismo y en la fase de implementación se llevan a cabo las tareas planificadas en la fase de diseño. Durante la fase de prueba, se revisan las tareas completadas

en la fase de implementación y se comparan con las que se planificaron en la fase de diseño, en esta fase también se realizan correcciones. Por último, en la fase de mantenimiento, pueden darse tres circunstancias diferentes, una en la que se ha terminado correctamente el proyecto, otra en la que ha perdido viabilidad y otra en la que no es posible llevar a cabo los objetivos planificados.[10]

A.2 Metodología *AGILE*

Todas las metodologías *agile* o ágiles, se basan en el manifiesto ágil, creado por una serie de expertos. El manifiesto ágil se divide en doce principios y cuatro valores. Estos cuatro valores representan un cambio de mentalidad con respecto a la organización, estos valores se consiguen siguiendo los doce principios del manifiesto.

Esta metodología, al contrario que la *Waterfall*, permite tener una flexibilidad y una capacidad de modificar el proyecto a medida que se va avanzando con las diferentes tareas. Este método se basa en la entrega y revisión de entregables en periodos pequeños de tiempo. Esos periodos de tiempo se denominan *sprint*, y suelen durar unas dos semanas.

En resumen, este método se basa en la agilidad, la calidad y la eficiencia. La agilidad se refiere a que los requisitos y soluciones del desarrollo del proyecto van cambiando según el equipo que lo está llevando a cabo va trabajando. La eficiencia hace referencia a la capacidad de lograr grandes soluciones con el mínimo número de recursos usados. Por último la calidad, es la capacidad de conseguir el producto o trabajo con los requisitos que se habían planificado, o incluso mejor. [10]

A.3 Metodología *Scrum*

La metodología *Scrum* es un tipo de metodología ágil, basada en el control de los procesos, asegurando que el conocimiento proviene de la experiencia y de las resoluciones obtenidas, apoyándose en lo que ya se conoce.

Es uno de los métodos más utilizados actualmente, ya que los equipos tienen como objetivo obtener el mejor resultado posible, de un proyecto complejo, abordado de forma creativa y productiva. Se trata de una técnica iterativa e incremental, controlando el riesgo en cada una de las fases planificadas.

Esta metodología sigue tres principios transparencia, inspección y adaptación. La transparencia se refiere a la capacidad, que tienen los responsables del proyecto, de conocer todas las tareas y procedimientos que se están llevando a cabo por un equipo. Mediante las revisiones, o inspecciones, los miembros del equipo que están usando la metodología comprueban todos los aspectos del todas las tareas e identifican posibles variaciones. Por último, la adaptación hace referencia a que si se encuentran variaciones los procesos van a cambiar para ajustar el resultado del proyecto a los requisitos iniciales. [10]

A.4 Comparaciones

La metodología *Waterfall* y la ágil, tienen muchas diferencias. Mientras que el modelo de cascada toma el proyecto completo y define la ruta a seguir, enfocándose en las fases definidas del proceso, de manera secuencial. La metodología ágil comienza el trabajo desde la primera reunión, se definen los *sprint* para planificar las revisiones y se plantean reuniones diarias del equipo para ir ajustan las

fases del proyecto según se está llevando a cabo. Además, otra de las diferencias son las divisiones del equipo del trabajo, por un lado, en el modelo de cascada, las jerarquía están muy bien definidas, la comunicación entre las diferentes personas del equipo es más formal. Por otro lado, en las metodologías ágiles, se necesitan equipos multidisciplinarios y que se organicen ellos mismos, siendo además su comunicación más informal.

En consecuencia, se puede ver que la técnica *Waterfall* es fácil de llevar a cabo ya que las jerarquías están divididas, se tienen unas fechas concretas para entregar cada documento y las revisiones son sencillas al realizarse cuando ya se han terminado todas las fases del desarrollo. Al estar todas las fases y tareas definidas desde el primer día es más sencillo seguirla. Este tipo de metodología es adecuada para proyectos pequeños. Algunas de las desventajas que supone adoptar esta metodología de trabajo es que, no se pueden agregar nuevos requisitos al proyecto una vez que se ha definido, ya que no permite cambio en las fases definidas en los primeros días. Asimismo, si en la etapa de revisión se encuentran fallos que no se pueden corregir, habria que empezar de nuevo el proyecto.

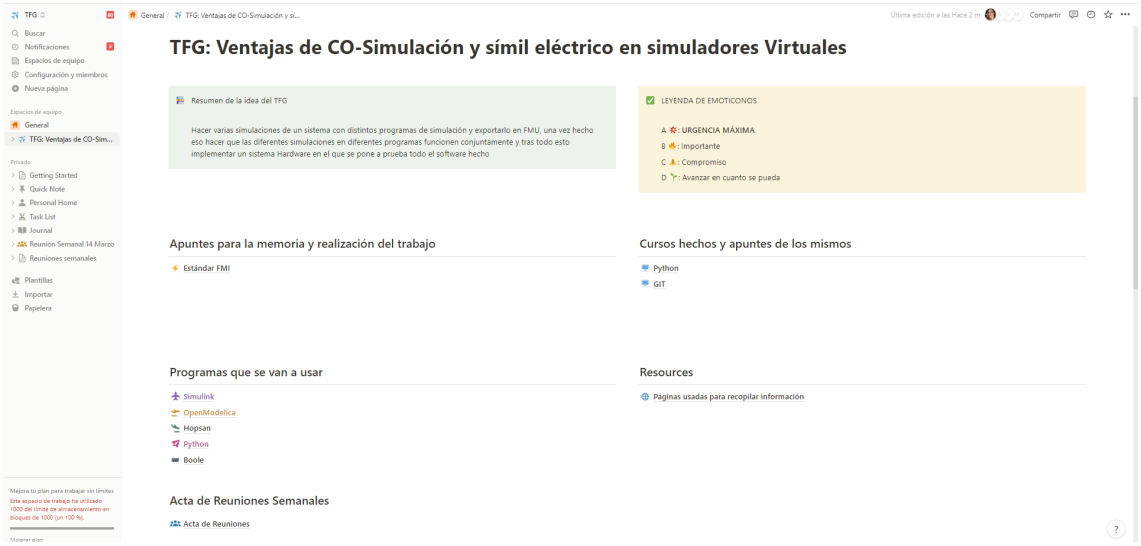
A partir de los datos expuestos de las metodologías ágiles, se pueden exponer algunas de las ventajas. Las reuniones diarias permiten una revisión y reajuste del desarrollo del proyecto, corrigiendo así los errores pequeños, antes de que aumenten. Otra de las ventajas es la colaboración, cooperación y comunicación con los clientes. Al tener los diferentes documentos al final de cada *sprint* y mostrárselos a los clientes, estos pueden hacer cambios en el producto antes de tenerlo terminado. Es un modelo más flexible y que según el esfuerzo del equipo que esté desarrollando el producto, se puede incluso hasta mejorar los requisitos iniciales a lo largo del trabajo.

Tras estas comparaciones, para este trabajo se escogió seguir una metodología ágil, ya que a medida que avanzaba el proyecto se han ido añadiendo requisitos diferentes a los que se tenían en un principio. No se escogió una metodología ágil *Scrum* porque son para problemas más complicados y suponen un control muy exhaustivo.

En este trabajo, se han hecho dos *sprint* distintos. Cada semana se ha realizado una reunión, una semana se revisaba lo que se había logrado hasta el momento y se corregían fallos, y la siguiente reunión se aumentaban los requisitos del trabajo final. Además, para la planificación se ha utilizado la herramienta *Notion*, en la que se ha ido guardan la documentación usada, los enlaces a las diferentes páginas web, los archivos FMU generados, los códigos de Python, etc. Asimismo, se creó un sprint específico para este trabajo, que se ha ido modificando a medida que iban sucediendo las semanas y se iba añadiendo requisitos.

En la Figura A.2 (a) se pueden ver la portada y en la Figura A.2 (b) se muestran las carpetas creadas en esta aplicación, con la documentación y códigos.

(a)



(b)

Figura A.2 (a) Portada de Notion (b) Carpetas de la documentación, códigos, actas de reuniones y enlaces usados. .

En la Figura A.3 se puede observar el sprint creado para este trabajo, que se iba modificando cada semana.

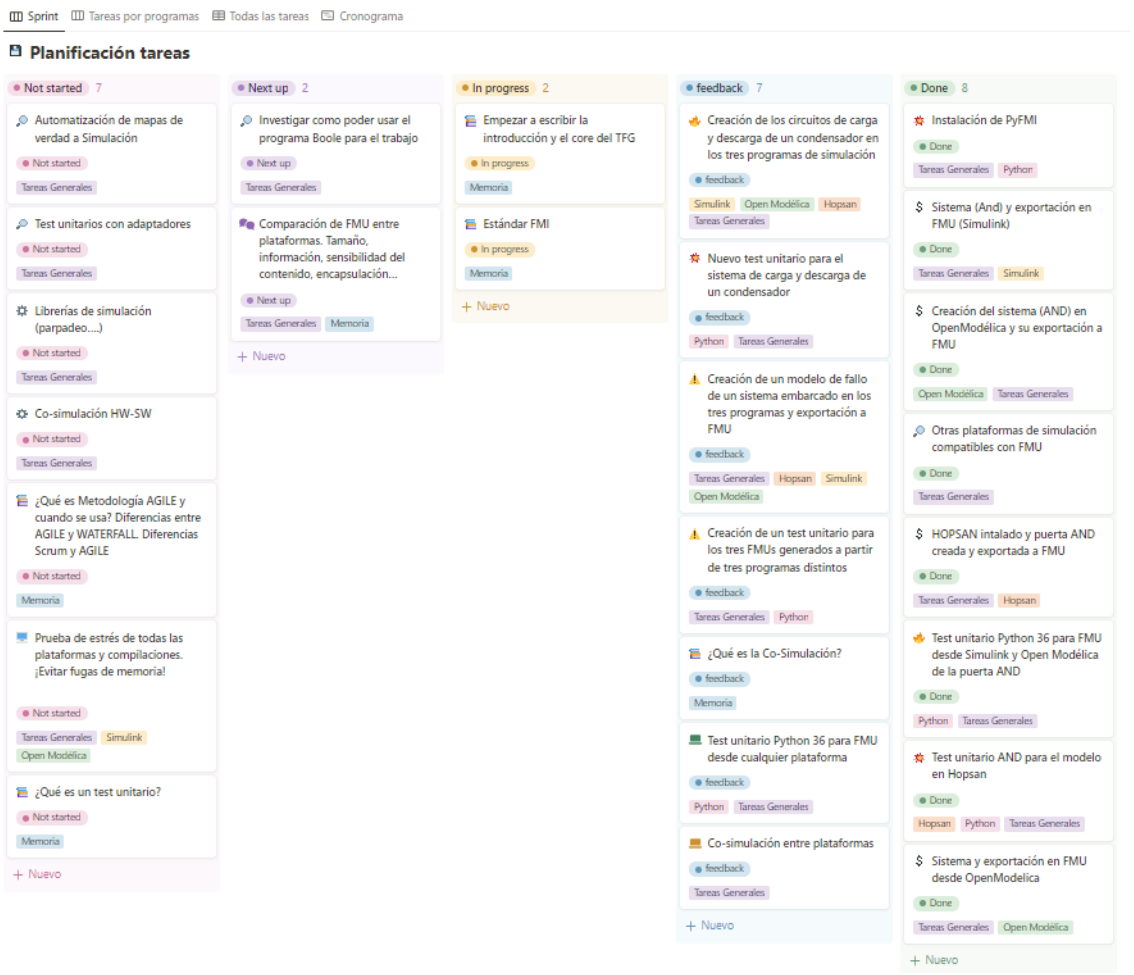


Figura A.3 Sprint de Notion .

Además del sprint, se creó un cronograma, una lista de tareas y las tareas dividas según los programas a los que correspondía esa labor, al tener tantos programas distintos fue de gran ayuda. En la Figura A.4 se puede ver la lista de tareas completa.

Sprint | Tareas por programas | Todas las tareas | Cronograma

Planificación tareas

Name	Start Date	FINISHED	Place	Status	Tags	Prioridad
Creación de un test unitario para los tre	5 de abril de 2023			feedback	Tareas Generales Python	Compromiso
Creación de un modelo de fallo de un s	5 de abril de 2023	5 de abril de 2023		feedback	Tareas Generales Hopsan	Compromiso
¿Qué es un test unitario?				Not started	Memoria	Compromiso
Empezar a escribir la introducción y el c				In progress	Memoria	Hacer cuando se pueda
Nuevo test unitario para el sistema de c	2 de abril de 2023			feedback	Python Tareas Generales	Compromiso
Creación de los circuitos de carga y des	25 de marzo de 2023	3 de abril de 2023		feedback	Simulink Open Modélica	Importante
Test unitario AND para el modelo en Ha	26 de marzo de 2023	2 de abril de 2023		Done	Hopsan Python Tareas G	Urgente
HOPSAN intalado y puerta AND creada	25 de marzo de 2023	25 de marzo de 2023		Done	Tareas Generales Hopsan	Compromiso
Prueba de estrés de todas las plataform				Not started	Tareas Generales Simulink	Hacer cuando se pueda
Comparación de FMU entre plataforma				Next up	Tareas Generales Memoria	Compromiso
¿Qué es Metodología AGILE y cuando s				Not started	Memoria	Compromiso
Creación del sistema (AND) en OpenMo	19 de marzo de 2023	19 de marzo de 2023		Done	Open Modélica Tareas Gen	Compromiso
Investigar como poder usar el program				Next up	Tareas Generales	Hacer cuando se pueda
Estándar FMI	17 de marzo de 2023			In progress	Memoria	Hacer cuando se pueda
¿Qué es la Co-Simulación?				feedback	Memoria	Compromiso
Co-simulación HW-SW				Not started	Tareas Generales	Hacer cuando se pueda
Librerías de simulación (parpadeo....)				Not started	Tareas Generales	Hacer cuando se pueda
Co-simulación entre plataformas				feedback	Tareas Generales	Hacer cuando se pueda
Test unitarios con adaptadores				Not started	Tareas Generales	Hacer cuando se pueda
Automatización de mapas de verdad a				Not started	Tareas Generales	Hacer cuando se pueda
Otras plataformas de simulación compa	19 de marzo de 2023			Done	Tareas Generales	Hacer cuando se pueda
Test unitario Python 36 para FMU desd				feedback	Python Tareas Generales	Hacer cuando se pueda
Sistema y exportación en FMU desde O				Done	Tareas Generales Open Mod	Hacer cuando se pueda
Test unitario Python 36 para FMU desd	25 de marzo de 2023	26 de marzo de 2023		Done	Python Tareas Generales	Compromiso
Sistema (And) y exportación en FMU (S	19 de marzo de 2023	19 de marzo de 2023		Done	Tareas Generales Simulink	Compromiso
Instalación de PyFMI	19 de marzo de 2023	19 de marzo de 2023		Done	Tareas Generales Python	Compromiso

+ Nuevo

Figura A.4 Lista de tareas de Notion .

Índice de Figuras

Índice de Tablas

Bibliografía

[1] *Unittest-Infraestructura de test unitarios https : //docs.python.org/es/3.9/library/unittest.html*, (10/02/2023).

[2] *¿Que son las pruebas unitarias de software? https : //keepcoding.io/blog/que − son − las − pruebas − unitarias − de − software/caracteristicas$_d$e$_l$as$_p$ruebas$_u$nitarias$_d$e$_s$oftware*, (23/05/2023).

[3] *Archivos .xml: cómo abrir y editarlos https : //www.ionos.es/digitalguide/servidores/know − how/extension − de − archivo − xml − 1/*, (23/2/2023).

[4] *Functional Mock-up Interface official web.https : //fmi − standard.org/tools/*, (Último acceso: 03/07/23).

[5] *Pyfmi https : //jmodelica.org/pyfmi/index.html*, (último acceso: 06/07/2023).

[6] *Archivos DLL: Qué son? Para qué sirven? https : //www.tecnologia − informatica.com/archivos − dll − que − son − sirven/*, (último acceso: 20/07/2023).

[7] *Pyfmi folder https : //jmodelica.org/pyfmi/pyfmi.html?highlight*, (último acceso: 20/07/2023).

[8] AA AbdElhafez and AJ Forsyth, *A review of more-electric aircraft*, International conference on aerospace sciences and aviation technology, vol. 13, The Military Technical College, 2009, pp. 1–13.

[9] Altair, *Altair Activate. https : //www.altair.com.es/activate/*, Último acceso: 03/07/23.

[10] J Aguirre Barrera and S Aguirre Barrera, *Metodologías para el desarrollo de Proyecto*, Administración de Empresas (2020).

[11] Ravenna Mónica Casalet, *V.Convergencia y digitalización de la producción en el sector aeroespacial*, El paradigma de la convergencia del conocimiento.

[12] Francisco Castañeda, *¿Tienes archivos JSON? Asi puedes abrirlos y editarlos*, (20/04/2023).

[13] DACCOSIM, *DACCOSIM. https : //bitbucket.org/simulage/daccosim/wiki/home*, Último acceso: 03/07/23.

[14] Departamento de Informàtica y Electrònica, *Tema 6 : Tècnicas para simulaciòn en tiempo real*.

[15] Instituto Tecnológico de la Energia, *Generación y usos de H2 a partir de residuos https://www.ite.es/generacion-y-usos-de-h2-a-partir-de-residuos/*, (12/07/2023).

[16] File.extention, *Extensión de archivo LIB https://www.file-extension.info/es/format/lib*, (15/12/2022).

[17] *File.extention*, *Extensión de archivo H https://www.file-extension.info/es/format/h*, (16/12/2022).

[18] G González Filgueira, *Modelización y estimación de un sistema dinámico hidráulico mediante uso de dispositivos eléctricos y electrónicos*.

[19] Fortiss, *Auto FOCUS 3: Model-based development of embedded systems. https://www.fortiss.org/en/results/software/autofocus-3*, Último acceso: 03/07/23.

[20] Peter Fritzson, *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*, John Wiley & Sons, 2014.

[21] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe, *Co-simulation: a survey*, ACM Computing Surveys (CSUR) **51** (2018), no. 3, 1–33.

[22] Maplesoft, *Maplesim: A Powerful Modelica Platform. https://www.maplesoft.com/products/maplesim/modelica.aspx*, Último acceso: 03/07/23.

[23] Mathworks, *Simulink. https://es.mathworks.com/products/simulink.html*, Último acceso: 03/07/23.

[24] Open Modelica, *Open Modelica. https://openmodelica.org/*, Último acceso: 03/07/23.

[25] David Muñoz Pellicer, *Integración y paralelización de algoritmos de optimización estructural*, (2019).

[26] David Israel Posadas Navarro, *Aplicación de la analogía hidráulica-eléctrica en el estudio experimental de un flujo pulsátil*, (2016).

[27] RAE, *Emulaciòn*.

[28] Siemens, *Simcenter Amesim software. https://plm.sw.siemens.com/es-es/simcenter/systems-simulation/amesim/*, (Último acceso: 03/07/23).

[29] Dassault Systeme, *Dymola Systems Engineering. https://www.3ds.com/es/productos-y-servicios/catia/productos/dymola/*, (Último acceso: 03/07/23).

[30] Linköping University, *Hopsan. https://liu.se/en/research/hopsan*, (Último acceso: 03/07/23).

[31] Wikipedia, *Log (informatica)*, (11/05/2023).