Universidad de Valladolid

E.T.S. Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática – Mención TI

DxPCs 2.0 - Ampliación de la plataforma para diagnosis basada en modelos

Autor:
D. Mario Matesanz Niño

Tutores:
Dr. J. Belarmino Pulido Junquera
Dr. Carlos J. Alonso González

AGRADECIMIENTOS

A mi tutor Belarmino por su dedicación y apoyo al proyecto.

RESUMEN

Este TFG tiene como objetivo el análisis y la mejora de una herramienta de diagnosis basada en modelos con Posibles Conflictos, escrita en el lenguaje de programación Java. La finalidad de la herramienta es detectar dónde y cuándo pueden producirse errores en un sistema mediante una diagnosis completa siguiendo el método de la Diagnosis Basada en Modelos con Posibles Conflictos. Para el cálculo de los PCs la herramienta se apoya en otra de cálculo de CPCs.

La versión existente de la herramienta que se va a mejorar sólo tiene la capacidad de modelar sistemas continuos. La mejora consistirá en añadir la capacidad a la herramienta de modelar sistemas híbridos, pudiendo realizar así una diagnosis completa sobre ambos tipos de sistemas. Además se introducirán nuevos algoritmos para mejorar la eficiencia de la herramienta.

SUMMARY

This TFG aims the analysis and the improvement of a model-based diagnostic tool with Possible Conflicts, written in Java programming language. The purpose is to detect when and when faults appear in the system through a complete diagnostic process following the approach names Consistency-based diagnosis using Possible Conflicts. The tool relies upon the CPC or Possible Conflict Java calculation tool.

The previous version of the tool that is going to be improved is only capable to work with continuous systems. This TFG will increase these capabilities, working with hybrid systems too. In addition new algorithms will be introduced to improve the efficiency.

GLOSARIO

- ***ADE***: Ecuación Algebraico-Diferencial, ADE en inglés.
- **Candidato:** es el elemento o conjunto de elementos del sistema en los cuales consideramos que puede existir un error.
- ***CCEM***: Conjunto de Cadenas Evaluables Minimales.
- **CEM**: Cadena Evaluable Minimal, Sistema sobredeterminado que nos permite realizar la estimación de un valor. La minimalidad indica que no existirá otra CEM que sea subconjunto propio de la primera.
- ***CMEM***: Conjunto de Modelos Evaluables Minimales.
- **Condición:** es una expresión lógica que debe cumplirse para poder aplicar la ecuación que lleva asociada a ella.
- ***CPC***: Conjunto de Posibles Conflictos.
- ***DBC***: Diagnóstico Basado en Consistencia.
- ***DBM***: Diagnóstico Basado en Modelos.
- **Interpretación:** son cada una de la formas de las cuales puede resolverse una ecuación.
- **MEM:** Modelo Evaluable Minimal, Modelo de un subsistema que puede evaluarse mediante la resolución local de cada una de sus relaciones a partir de unas observaciones. La minimalidad indica que no hay un subconjunto propio que cumpla estas condiciones.
- ***MODELO E/S:*** Modelo Entrada/Salida.
- ***MVC:*** Modelo-Vista-Controlador.
- **PC**: Posible conflicto, conjunto de ecuaciones con redundancia analítica mínima, es decir, cualquier CEM que represente al menos un CEM.
- ***PFC***: Proyecto Fin de Carrera.
- **Residuo:** es la discrepancia que puede existir entre el valor medido del sistema real y el valor obtenido en el modelo.
- **Sistema Híbrido**: es un sistema con múltiples modos de funcionamiento continuo comandado por acciones discretas.
- ***TFG***: Trabajo de Fin de Grado.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

1.1. Contexto

Este TFG tiene como objetivo introducir una serie de mejoras en una herramienta ya realizada en PFCs y TFGs anteriores, en el lenguaje de programación Java, vinculados al ámbito del diagnóstico basado en modelos con Posibles Conflictos. Esta técnica además ha sido desarrollada en el GSI de la Universidad Valladolid.

La técnica de diagnosis basada en modelos mediante PCs se basa en realizar un análisis off-line del modelo de un sistema para intentar detectar potenciales conflictos, dada la definición un conjunto de valores (variables) y de las relaciones que las unen (ecuaciones). Las variables podrán ser de dos tipos: observadas, cuando puede medirse su valor, y no observadas cuando no. Como conflicto denominamos a la diferencia que puede existir entre el valor medido del sistema real y el valor obtenido en el modelo. Lo que buscamos es identificar los subsistemas con redundancia a partir de los cuales generamos los modelos y detectar así si existen discrepancias entre las estimaciones de variables. Para ello, usaremos la técnica de Posibles Conflictos que nos ayudará a resolverlo identificando esos subsistemas.

La herramienta en su estado actual es capaz de simular los modelos asociados a PCs y en base a ellos realizar la detección y localización de fallos analizando los residuos generados por los PCs. Además ofrece la posibilidad de visualizar los resultados de experimentos de diagnosis así como los obtenidos en los modelos ofreciéndonos una comparación entre ellos.

La funcionalidad que vamos a añadir consistirá en realizar una ampliación de la herramienta actual para que pueda usarse con sistemas híbridos. Un sistema híbrido es un sistema con múltiples modos de funcionamiento comandados por acciones discretas. Es decir, la herramienta antes obtenía los subsistemas sobre el sistema sin restricciones, ahora se le añadirán uno o varios parámetros más al sistema los cuales pueden tener dos valores cada uno. Según sea el valor del parámetro, éstos influirán completamente en el funcionamiento del sistema ya que cada ecuación del modelo tendrá asociada una condición que nos indicará cuándo puede ser usada. Así las ecuaciones sólo serán válidas en ciertas situaciones. Estas situaciones suelen modelarse como "condiciones" asociadas a cada ecuación. Si una condición se cumple en un modo de trabajo, entonces la ecuación puede intervenir en el modelo. Este hecho complica el cálculo de los PCs. Es en este aspecto del cálculo con condiciones es en el cual nos vamos a centrar en este TFG.

1.2. Introducción

Antes de comenzar vamos a introducir brevemente el marco teórico sobre el cual vamos a trabajar durante la realización del proyecto.

Se va a dar una descripción de los conceptos de diagnóstico, diagnóstico basado en modelos y posibles conflictos, así como de los conjuntos de apoyo que sirven para su cálculo y obtención que son las cadenas evaluables minimales y los modelos evaluables minimales.

Tras esto se realizará de forma breve una descripción de las herramientas que están ya realizadas y que usaremos como punto de partida que son CPCs y DxPCs.

1.3. Conceptos

Las tareas de diagnóstico son muy importantes en nuestra vida diaria y se encuentran muy presentes en ella. Ante el gran desarrollo que están sufriendo los sistemas diseñados por seres humanos, aumentando cada vez más su complejidad, las tareas de diagnóstico se vuelven cada vez más importantes. Al implementar sistemas de diagnóstico se evitan situaciones de riesgo en dichos sistemas y por lo tanto se produce una mejora en el producto final, en el proceso de fabricación o simplemente en el funcionamiento de un sistema.

El diagnóstico es según Balakrishnan y Honovar (Balakrishnan & Honavar,1998) el acto o el proceso de decidir la naturaleza de un problema tras el examen realizado de sus síntomas. Los mismos nos proponen unas cuestiones a partir de las cuales podemos hacer una división de las diferentes aproximaciones al diagnóstico. La preguntas son : ¿Cómo sabrá el sistema la relación entre los síntomas observados y el consecuente diagnóstico?, ¿Cómo representará el sistema esta relación? y ¿Cómo usará esta representación para el diagnóstico de fallos?.

A partir de estas preguntas, las técnicas de diagnóstico serán:

- Técnicas basadas en conocimiento
- Técnicas basadas en casos
- Técnicas basadas en aprendizaje automático
- Técnicas basadas en modelos

Las tres primeras no las vamos a utilizar por lo que no son de interés para este trabajo, la última, la relacionada con los modelos, es en la cual nos vamos a centrar y a la que nos vamos a referir como Diagnóstico Basado en Modelos (DBM). Hablamos de modelos como una abstracción de un sistema real al que representamos de forma simbólica, mediante un conjunto de ecuaciones, pudiendo obtener así una estimación del correcto comportamiento del mismo.

El DBM está basado en la comparación de unos datos reales obtenidos de forma experimental sobre el funcionamiento del sistema, que indican cómo se comporta el sistema real, con los datos que se han obtenido en los diferentes modelos hallados, que indican cómo debería comportarse el sistema funcionado correctamente. Cuando esos valores no coincidan, y la diferencia sea mayor que un margen, se llegará a la conclusión de que el sistema no está funcionando de forma correcta. Si la diferencia entre los valores obtenidos es menor que el margen, el sistema funciona aparentemente de forma correcta.

Entre las distintas aproximaciones al DBM vamos a centrarnos en el Diagnóstico Basado en Consistencia (DBC), que enfoca el proceso de diagnosis como un proceso iterativo de predicción del comportamiento, detección de discrepancias, generación de un conjunto de candidatos (aquellos elementos del sistema que pueden dar lugar a errores) y refinamiento de dicho conjunto.

Esta técnica tiene la ventaja de que no necesita tener información como algún conjunto de datos o información relativa a fallos ocurridos con anterioridad, sólo necesita conocer sobre los modelos de sus componentes. Sin embargo, también tiene inconvenientes como

la dificultad de obtener un modelo que represente de forma precisa el sistema así como la carga computacional que representa el tener que calcular todas las posibles diagnosis para un fallo.

El problema del diagnóstico se resolverá cuando se identifique aquellos elementos que dan lugar a los conflictos. Los conflictos son las diferencias que podríamos encontrar entre los valores observados y entre los valores predichos. Esas diferencias que podemos observar, son definidas por un conjunto de componentes, a los cuales los vamos a denominar como conjunto conflicto.

Podemos hablar de Posibles Conflictos cuando ante la ausencia de fallos estructurales, usando los modelos, podemos realizar una estimación fuera de línea, es decir sin tener en funcionamiento real del software, de aquellas partes del modelo de las cuales pueden surgir conflictos. A nivel teórico, definimos PC como un conjunto de ecuaciones con redundancia analítica mínima. A la hora de hallar estos posibles conflictos en nuestro proyecto se añadirá un elemento más a considerar que serán unas condiciones las cuales según su valor determinará si pueden usarse o no las ecuaciones del sistema dando lugar a diferentes resultados según el valor de la "condición".

La generación de estos posibles conflictos viene dado por un proceso de tres fases:

1. Se genera una representación abstracta del sistema como un hipergrafo[1]. En él, los nodos representarán variables, y los arcos las relaciones entre las mismas.
2. Búsqueda de Cadenas Evaluables Minimales (CEMs), éstas son subsistemas de relaciones sobredeterminados.
3. Búsqueda de Modelos Evaluables Minimales (MEMs), que son todas las vías posibles de evaluación de los anteriores subsistemas.

Estos subsistemas que hemos mencionado, las CEMs deben cumplir un conjunto de condiciones que debemos tener en cuenta a la hora de obtenerlos. La condiciones serán:

- Formar un hipergrafo conexo, es decir, que se pueda formar un camino desde cualquier vértice a otro.
- Tener algún valor observado, los cuáles, son variables que se pueden medir.
- Dados unos valores observados, que estos se puedan propagar localmente para hallar los no observados.
- Debe de haber tantas ecuaciones como variables desconocidas + 1
- Al ser minimales, su hipergrafo no deberá estar contenido dentro del hipergrafo de otra CEM.

Una vez obtenidas todas las CEMs, los MEMs , serán todas las formas de las que pueden evaluarse cada CEM.

Cada uno de los MEMs está representado por un grafo Y-O (grafo causal) de una CEM en el que se pueden encontrar cero, una o más formas de resolver el modelo.

Este grafo que representa los modelos, debe cumplir:

[1] Es una generalización de los grafos usados habitualmente en computación, donde se permiten relaciones n-arias entre los nodos, en lugar de restringirlos a relaciones binarias.

- Ha de tomar una y sólo una de las posibles interpretaciones asociadas a cada hiperarco (ecuación) de la CEM.
- Ha de contener un sólo nodo discrepancia (variables que son no observadas y estimadas por dos vías o variables que son observadas y estimadas por una vía).
- Los nodos hoja han de ser variables observadas.
- Los valores del nodo discrepancia han de ser obtenidos a partir de las observaciones y propagando localmente.

Una vez que ya hemos calculado el Conjunto de Modelos Evaluables Minimales (CMEMs) ya tenemos de forma inmediata el otro conjunto, el de posibles conflictos (CPCs).

Será entonces un PC el conjunto de relaciones en una CEM que contenga al menos un MEM. El proceso de cálculo de los mismos consistirá en recorrer cada CEM y ver si tiene un MEM asociado.

1.4. Herramientas

Las dos herramientas de las que disponemos son:

- **CPCs**: Es una herramienta de cálculo de posibles conflictos. Recibe como entrada un modelo que representa un conjunto de ecuaciones en formato E/S (bien existente o generado a través de la herramienta DxPCs que comentaremos a continuación). En este formato, sólo nos interesa saber qué ecuaciones hay en el modelo, qué variables tienen, si son observadas o no, y cómo se usan todas menos una, para calcular la variable restante de cada ecuación.

- **DxPCs**: Es una herramienta que realiza la diagnosis de un sistema a partir de los PCs obtenidos en la herramienta CPCs. El origen de esta herramienta es el PFC de Alberto Hernández (Hernandez, 2012) que creo la versión 1.0 para la herramienta Matlab. Posteriormente fue ampliada y traducida a Java en las prácticas de empresa de los estudiantes David Rubio y Luis Miguel Villarroel (Pulido, 2016). La herramienta está dividida en cuatro apartados:

    - SS/IO Notation converter: este apartado genera un Modelo E/S a partir de las ecuaciones que representan el sistema. Este modelo generado le utiliza la herramienta CPCs para generar los PCs.
    - Experiment generator: a partir de una serie de valores de entrada genera las salidas de las ecuaciones que representan el sistema a lo largo de un periodo de tiempo.
    - Simulation Model Generator: a partir de la salida de la herramienta CPCs, es decir, de los PCs, genera los Modelos de Simulación. Con ellos se realizará posteriormente la diagnosis.
    - Diagnosis: Realiza una diagnosis del sistema mediante los Modelos de Simulación obtenidos en el anterior apartado. Los resultados se pueden mostrar gráficamente y ver así como aumenta o disminuye el error al que la herramienta llama “residuo”. Una vez que ha calculado los residuos la herramienta puede determinar cuáles de los elementos del modelo son candidatos a fallo.

Como hemos mencionado al principio, el problema que se pretende resolver con esta herramienta es el modelado de sistemas híbridos. Sistemas híbridos son aquellos en los cuales algunas de sus partes sólo estarán activas o variarán según se den ciertas circunstancias. Ponemos como ejemplo para ilustrar, un sistema de los cuatro tanques.

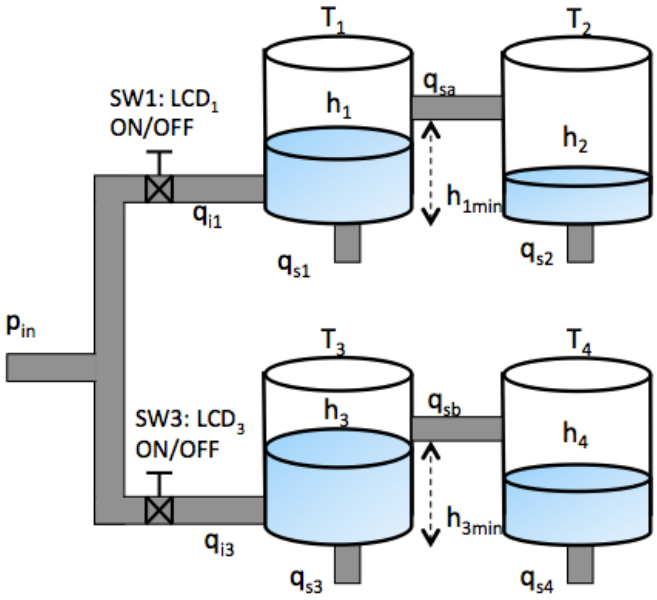


Figura 1.1 : Sistema 4 tanques

Este sistema de cuatro tanques es un sistema claramente híbrido. Como vemos tiene dos partes, la superior y la inferior. Cada una de ellas tiene un interruptor con un cierre, el cual permite o impide por completo el paso del agua, es decir, cada una de las dos partes está totalmente condicionada por el interruptor. Estos interruptores no son lo único que hace considerar híbrido a este sistema también nos encontramos con las tuberías que conectan los dos tanques inferiores y superiores. Como vemos esta tubería no se encuentra a ras de suelo por lo que la entrada de agua en ella depende totalmente de que la altura en los tanques 1 o 3 sea superior a la altura de la misma. Así la entrada del agua en los tanques 2 y 4 dependerá de que la tubería de conexión tenga caudal, como ya hemos visto condicionada por la altura de los otros dos tanques. Como vemos son varios aspectos los que hacen considerar a este sistema de 4 tanques híbrido ya que varias de sus partes están claramente condicionadas.

Entonces para poder modelar estos sistemas híbridos hay que permitir en nuestro conjunto de ecuaciones que haya condiciones de aplicación de cada una de las ecuaciones. El tener que añadir estas condiciones nos obliga a extender la gramática/sintaxis que permitimos en nuestros ficheros y a adaptarnos a la sintaxis que se usa en los ficheros que se usan para calcular PCs en presencia de condiciones (Domínguez, 2013).

2. PLAN DE DESARROLLO DE SOFTWARE

2.1. INTRODUCCIÓN

2.1.1. Propósito

El propósito de este Plan de desarrollo Software es ofrecer toda la información necesaria para controlar el desarrollo del proyecto "DxPCs 2.0 - Ampliación de la plataforma para diagnosis basada en modelos ". Nos ofrecerá una visión de alto nivel de abstracción que facilitará al equipo de trabajo la tarea de organizar el desarrollo.

Los usuarios potenciales de este apartado serán:

- Jefe de proyecto: lo usará para establecer la planificación temporal y de recursos, además de realizar el seguimiento y la evaluación del proyecto.
- Miembros del equipo de proyecto: usarán este apartado para saber qué deben hacer, cuándo lo deben hacer y qué tareas dependen de las que ellos tienen asignadas.

2.1.2. Alcance

Este Plan de Desarrollo Software describe un plan general para ser utilizado por el equipo de desarrollo para la ejecución del proyecto " DxPCs 2.0 - Ampliación de la plataforma para diagnosis basada en modelos ".

Los planes que se detallan en este capítulo se basan en la especificación de requisitos que será detallada en el siguiente capítulo de este documento.

2.1.3. Definiciones, Acrónimos, Abreviaturas

Véase el glosario general del proyecto.

2.1.4. Perspectiva General

El plan de desarrollo software contendrá:

- Perspectiva del proyecto: descripción de propósito, alcance y objetivos así como los entregables que se prevén.
- Organización del proyecto: En él se describe la estructura de organización del equipo de proyecto.
- Proceso de Gestión: define la estimación de costes en términos de tiempo y recursos, las fases del proyecto junto con sus hitos y cómo se llevará a cabo la monitorización del proyecto.

2.2. PERSPECTIVA GENERAL DEL PROYECTO

2.2.1. Propósito, alcances y objetivos

Este proyecto trata sobre la ampliación de la funcionalidad de una plataforma existente y que está fundamentada en la diagnosis basada en modelos. Las principales metas son:

- Modificar la herramienta para que admita modelos de sistemas dinámicos híbridos. Estos nuevos modelos asocian condiciones de contorno para cada ecuación de tipo ADE (Ecuación Algebraico-Diferencial, ADE en inglés). La condición no es más que una expresión lógica que debe cumplirse para poder aplicar la ecuación. De esta forma se definen los distintos modos de funcionamiento de un sistema (de ahí el término de híbrido, por contraposición a continuo).
- Realizar una propuesta de mejora de la generación de modelos evaluables asociados a Posibles Conflictos híbridos que facilite la simulación de estos Posibles Conflictos en múltiples modos de funcionamiento.

2.2.2. Suposiciones y restricciones

Para realizar la planificación, se tendrán en cuenta los siguientes factores que influirán en el desarrollo del proyecto:

- Fecha de entrega: El proyecto del deberá estar completamente finalizado antes de junio del 2016.
- Horario de trabajo: El tiempo que el equipo dedicará a la realización del proyecto estará considerablemente limitado por los cursos de idiomas que cursa el equipo. Se calcula que la dedicación a éste será de entre 10 y 20 horas semanales.
- Adiestramiento del equipo: Debido a la escasa experiencia previa del equipo de desarrollo en el lenguaje de programación a utilizar y en el método utilizado, el diagnóstico basado en modelos, se ha de tener en cuenta un período de formación y adaptación considerable.
- Recursos del proyecto: Se trabajará con unos recursos hardware y software limitados. Serán especificados en el apartado de recursos del proyecto.

2.2.3. Entregables del proyecto

En este apartado se van a indicar cada uno de los artefactos que serán generados en el proyecto, así como los entregables que se irán presentando a lo largo del proceso. Estos entregables al realizarse de forma iterativa e incremental serán susceptibles de ser modificados en una etapa posterior a la realización.

Los entregables serán:

- Gestión del proyecto:
  - Plan de desarrollo de software.
- Requisitos:
  - Introducción teórica.
  - Especificación de requisitos.

- Análisis y diseño:
  - Modelo de casos de uso.
  - Especificaciones de casos de uso.
  - Modelo de análisis.
  - Modelo de diseño.
  - Realización de los casos de uso.
- Implementación:
  - Modelo de implementación.
  - Versión funcional del apartado Conversión Modelos E/S.
  - Versión funcional del apartado Generador de Modelos de simulación.
  - Versión funcional del apartado Generador de Experimentos.
  - Versión funcional del apartado Diagnosis.
  - Versión funcional con mejoras de algunos algoritmos.
  - Código fuente
- Pruebas:
  - Batería de pruebas.
- Lanzamiento:
  - Manual de usuario.
  - Manual de instalación.
  - Producto final.

2.2.4. Evolución del plan de desarrollo

El plan de desarrollo se revisará de forma continuada y será mejorado antes de comenzar la siguiente iteración corrigiendo así posibles fallos que se hubieran podido realizar en el plan inicial.

2.3. ORGANIZACIÓN DEL PROYECTO

2.3.1. Estructura organizativa

Se trata de un proyecto que será realizado de forma individual por lo que los roles de Jefe de Proyecto, Analista, Diseñador, Programador y Personal de Pruebas van a recaer sobre la misma persona.

2.3.2. Interfaces externas

Se mantendrá un contacto de forma continua con los tutores del proyecto, con entrevistas en principio semanales, que será el que sirvan de asesoramiento y guía para cada una de las etapas de desarrollo.

2.3.3. Roles y responsabilidades

Identifica las unidades de organización del proyecto y de qué tareas se encargarán de realizar en el mismo.

Rol	Responsabilidad
Jefe de Proyecto	Se encargará de la gestión de recursos así como de la supervisión de las tareas del resto de miembros del equipo.
Analista	Su labor será realizar entrevistas para obtener requisitos y necesidades del cliente o completar los existentes. Definirá la arquitectura del sistema a muy alto nivel. Elabora diagramas de clases y secuencia para mostrar los bloques que estarán relacionados y se comunicarán entre sí. Dichos diagramas facilitarán la comprensión de los casos de uso de los usuarios.
Diseñador	Establecerá la arquitectura software para la herramienta. Estudiará la interoperabilidad con otros sistemas y la posibilidad de reutilización de software existente. Aprovechará las tecnologías disponibles. Desarrollará componentes a bajo nivel, estructuras de datos eficientes y algoritmos adecuados. También participará en detalles de la arquitectura a bajo nivel.
Programador	Se encargará de producir el código fuente de la aplicación.
Personal de Pruebas	Se encargará de diseñar y realizar los casos de prueba.

2.4. PROCESO DE GESTIÓN

2.4.1. Estimaciones de proyecto

Este apartado se centrará en las estimaciones temporales. Se van a estimar los tiempos que serán otorgados a la realización de cada tarea que se revisarán según se vaya avanzando y cambiando de una tarea a otra.

2.4.2. Plan de proyecto

2.4.2.1. Plan de fases

Fase	Nº de iteraciones	Duración (Semanas)
Concepción	1	2
Elaboración	2	4
Construcción	2	8
Transición	1	4

2.4.2.2. Objetivos de las iteraciones

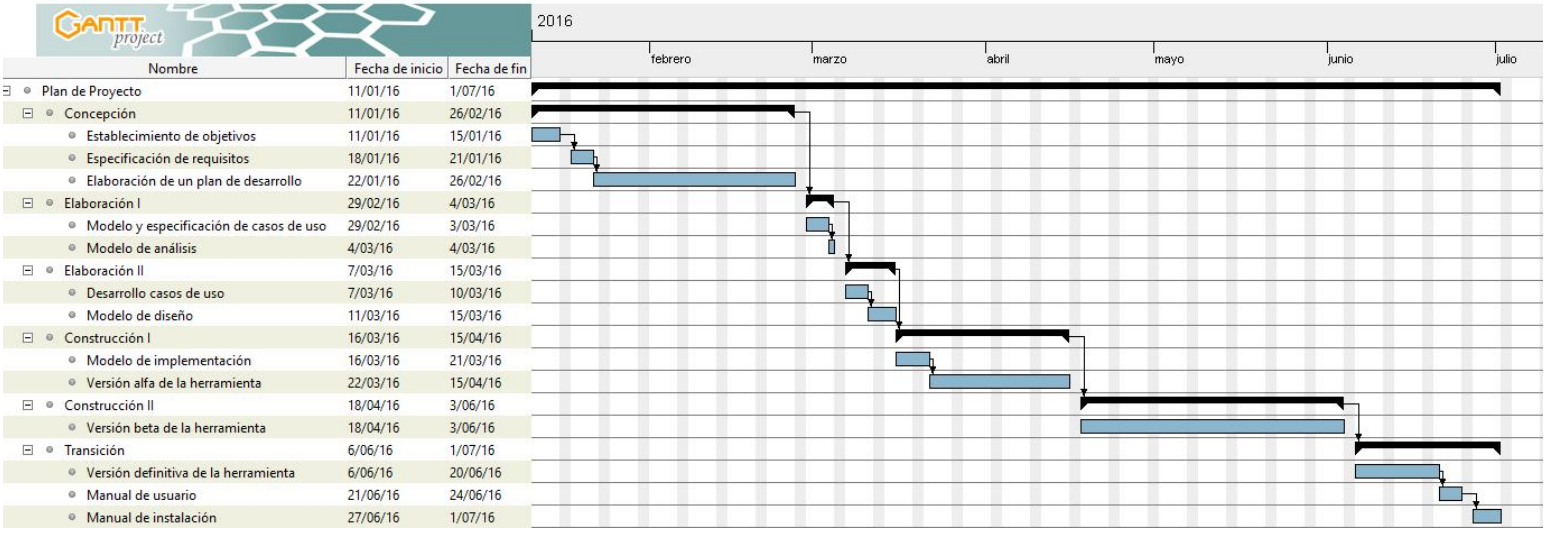
Iteración	Descripción	Hitos
Concepción	Creación del Plan de desarrollo y elaboración de Requisitos.	→ Plan de Desarrollo → Requisitos
Elaboración I	Análisis de la aplicación, elección de casos de uso y aproximación a la realización del diagrama de clases.	→ Modelo y especificación de Casos de Uso → Modelo de Análisis
Elaboración II	Diseño de la herramienta, creación del diagrama de clases, diagrama de secuencia…	→ Modelo de Diseño → Desarrollo de Casos de Uso
Construcción I	Implementación de la funcionalidad de la herramienta, desarrollo de la versión alfa.	→ Modelo de implementación → Modelado de los nuevos sistemas de entrada. → Versión alfa de la herramienta
Construcción II	Desarrollo de la versión beta.	→ Versión beta de la herramienta
Transición	Desarrollo de la versión definitiva y elaboración de manuales.	→ Versión definitiva de la herramienta → Manual de usuario → Manual de instalación

2.4.2.3. Versiones

Para cada uno de los apartados de la aplicación que vamos a realizar, se realizará:

- Versión alfa: esta versión será capaz de realizar la funcionalidad principal del apartado.
- Versión beta: añadirá a la versión anterior los cambios en la interfaz para que muestre los datos introducidos de forma correcta.
- Versión definitiva: añadirá a la versión anterior las mejoras en los algoritmos además de añadir el modo de funcionamiento automático.

2.4.2.4. Planificación temporal



Nombre	Fecha de inicio	Fecha de fin
Plan de Proyecto	11/01/16	1/07/16
Concepción	11/01/16	26/02/16
Establecimiento de objetivos	11/01/16	15/01/16
Especificación de requisitos	18/01/16	21/01/16
Elaboración de un plan de desarrollo	22/01/16	26/02/16
Elaboración I	29/02/16	4/03/16
Modelo y especificación de casos de uso	29/02/16	3/03/16
Modelo de análisis	4/03/16	4/03/16
Elaboración II	7/03/16	15/03/16
Desarrollo casos de uso	7/03/16	10/03/16
Modelo de diseño	11/03/16	15/03/16
Construcción I	16/03/16	15/04/16
Modelo de implementación	16/03/16	21/03/16
Versión alfa de la herramienta	22/03/16	15/04/16
Construcción II	18/04/16	3/06/16
Versión beta de la herramienta	18/04/16	3/06/16
Transición	6/06/16	1/07/16
Versión definitiva de la herramienta	6/06/16	20/06/16
Manual de usuario	21/06/16	24/06/16
Manual de instalación	27/06/16	1/07/16

Figura 2.1 : Diagrama de Gantt

2.4.2.5. Recursos del Proyecto

- Recursos humanos:
  - Un alumno de último curso de Grado en Ingeniería Informática mención TI.
  - Dos tutores para el seguimiento del proyecto.
- Recursos hardware:
  - Dos ordenadores portátiles personales. Para poder trabajar en paralelo pero principalmente se realizará la mayor parte con el portátil con S.O. OS X salvo para las pruebas para las cuales se necesita Matlab el cuál sólo tenemos la licencia para el portátil con S.O. Windows.
- Recursos software:
  - Sistema operativo Windows 10.
  - Sistema operativo OS X El Capitán.
  - Microsoft Office 2016 (Office 365, cuya licencia de un año la proporciona la UVA).
  - Astah professional (Licencia de un año proporcionada por la UVA).
  - Dropbox.
  - Java Runtime environment.
  - Netbeans 8.0.2.
  - REM (Requeriments manager).
  - Matlab 2010a (Licencia proporcionada por la UVA).

Todos los programas son gratuitos salvo los que se menciona que necesitan la licencia.

2.4.3. Monitorización y control de proyecto

2.4.3.1. Gestión de requisitos

Información disponible en el apartado de requisitos.

2.4.3.2. Control de calidad

Todos los entregables pasarán por un proceso de revisión adecuado. Dicho proceso deberá asegurar que se cumplan unos estándares mínimos de calidad; además de garantizar que cumplen con los requisitos especificados.

2.4.3.3. Gestión de riesgos

- Lista de riesgos

Se usarán los siguientes convenios para definir cada riesgo
Magnitud: calificará el nivel de riesgo de 1 a 10, siendo 1 lo más bajo y 10 lo más alto, basándonos en la probabilidad de que ocurra y sus consecuencias.
Descripción: breve exposición del problema.
Probabilidad: posibilidad de que ocurra el riesgo.
Impacto:

- C(Crítico): afecta a toda la funcionalidad del proyecto y a la línea base.
- A(Alto): afecta a las necesidades del usuario y a la funcionalidad.
- M(Medio): se puede contener pero es evitable en la mayoría de las veces.
- B(Bajo): se puede aceptar y es su arreglo es rápido.

RSK-01	Retraso en la entrega
Magnitud	10
Descripción	La entrega será posterior a la fecha establecida.
Probabilidad	30%
Impacto	C
Indicador	Semanas antes de la fecha tope más de 10% del proyecto está sin finalizar.
Plan de Contingencia	Dedicar más horas al proyecto.

RSK-02	Falta de conocimientos de los lenguajes de programación
Magnitud	6
Descripción	El equipo carece de experiencia en este lenguaje (Java - Matlab).
Probabilidad	65%
Impacto	A
Indicador	Retrasos a la hora de realizar la programación.
Plan de Contingencia	Formación previa de los programadores en el lenguaje.

RSK-03	Falta de conocimientos sobre la tecnología utilizada – Diagnóstico basado en modelos -
Magnitud	7
Descripción	El equipo nunca ha trabajado con la tecnología elegida.
Probabilidad	90%
Impacto	A
Indicador	Paradas en el progreso del trabajo – Retrasos.
Plan de Contingencia	Formación previa del equipo en esta tecnología.

RSK-04	Cambio en los requisitos del sistema
Magnitud	3
Descripción	Aparición de nuevas necesidades del cliente.
Probabilidad	70%
Impacto	A
Indicador	Inclusión de nuevos requisitos que dan lugar a la realización de más trabajo.
Plan de Contingencia	Concienciar al cliente de las consecuencias de realizar cambios en etapas avanzadas.

RSK-05	Disminución del tiempo de dedicación
Magnitud	5
Descripción	Los miembros del equipo ven limitado su tiempo de trabajo por circunstancias externas.
Probabilidad	30%
Impacto	M
Indicador	El avance del proyecto se ralentiza.
Plan de Contingencia	Disminución del los requisitos y/o posposición de la fecha de entrega.

RSK-06	Falta de experiencia en el proceso de planificación
Magnitud	5
Descripción	Debido a la falta de experiencia del jefe de proyecto es posible que no se realicen las estimaciones de forma adecuada.
Probabilidad	75%
Impacto	A
Indicador	Fallos constantes en la aplicación.
Plan de Contingencia	Consultar con expertos, comparación con otros proyectos similares, búsqueda de información…

RSK-07	Fallo hardware
Magnitud	8
Descripción	El PC de trabajo se estropea.
Probabilidad	10%
Impacto	A
Indicador	Fallos constantes del sistema.
Plan de Contingencia	Copias de seguridad en la nube – Disponer de un segundo PC.

RSK-08	Fallo software
Magnitud	8
Descripción	Algunos de los programas usados dejan de funcionar o falla el sistema operativo.
Probabilidad	10%
Impacto	A
Indicador	Fallos en el funcionamiento de alguno de los programas.
Plan de Contingencia	Copias de seguridad en la nube – Disponer de un segundo PC – Programas de trabajo alternativos.

RSK-09	Perdidas de trabajo
Magnitud	7
Descripción	Perdida de parte del trabajo ya realizado por algún fallo o error.
Probabilidad	15%
Impacto	A
Indicador	Faltan partes de trabajo ya realizado.
Plan de Contingencia	Copias de seguridad en la nube y/o en otro dispositivo.

RSK-10	Decisión errónea en fase específica (fallo en el planteamiento de tareas)
Magnitud	5
Descripción	Error en el planteamiento de las tareas.
Probabilidad	60%
Impacto	M
Indicador	Se producen retrasos.
Plan de Contingencia	Realización de las tareas sin prisas y con buen planteamiento evitando así errores graves.

RSK-11	Diseño complejo
Magnitud	4
Descripción	El diseño del producto puede exceder los conocimientos del equipo.
Probabilidad	20%
Impacto	M
Indicador	Retrasos y/o paradas por no saber avanzar.
Plan de Contingencia	Intentar realizar el diseño lo más simple posible.

3. REQUISITOS

3.1. INTRODUCCIÓN

En este capítulo se van a exponer todos los requisitos del software generados mediante la documentación y las entrevistas realizadas con el cliente. Estos requisitos deberán estar adecuadamente clasificados y redactados y deben darnos una idea bastante clara de las necesidades del cliente.

3.1.1. Propósito

El objetivo de este capítulo es dar una descripción de todos los requisitos que deberá cumplir el proyecto, facilitando así futuras fases del desarrollo.
Los usuarios potenciales de este apartado de la documentación serán:

- Jefe de proyecto: será su principal apoyo para la realización del Plan de Desarrollo de Software.
- Miembros del equipo de proyecto: será útil para que todos los miembros del equipo tengan claras las metas del proyecto.

3.1.2. Alcance

Este capítulo da una descripción de todos los requisitos de software para el proyecto: "DxPCs 2.0 - Ampliación de la plataforma para diagnosis basada en modelos".

3.1.3. Definiciones, Acrónimos, Abreviaturas

Véase el glosario general del proyecto.

3.1.4. Perspectiva General

El capítulo de identificación de requisitos contendrá esta información:

- Objetivos del proyecto
- Requisitos funcionales
- Requisitos de interfaz externo
- Restricciones de diseño
- Requisitos no funcionales

3.2. OBJETIVOS

A continuación se presenta una lista de los principales objetivos que se van realizar en este proyecto:

OBJ-01	Identificar aspectos a mejorar de la versión anterior
Descripción	Se realizará un análisis de la versión anterior de la herramienta y algoritmos, para encontrar posibles aspectos donde se introducirán mejoras para aumentar la eficiencia, o posibles fallos no localizados que serán corregidos.
Importancia	Alta
Comentarios	

OBJ-02	Ampliar la herramienta para que pueda trabajar con modelos discretos de sistemas híbridos
Descripción	Se realizará una modificación o nueva versión, según lo requiera, de cada uno de los algoritmos de la herramienta para que la misma funcione con modelos híbridos respetando siempre los antiguos sistemas.
Importancia	Alta
Comentarios	

OBJ-03	Ampliar la funcionalidad de la generación de modelos evaluables
Descripción	Se realizará una propuesta de mejora de la generación de modelos evaluables asociados a Posibles Conflictos híbridos que facilite la simulación de estos Posibles Conflictos en múltiples modos de funcionamiento.
Importancia	Alta
Comentarios	

3.3. REQUISITOS FUNCIONALES

En esta lista se mostrarán los requisitos funcionales de la aplicación que definen cómo debe comportarse el sistema y qué es capaz de hacer.

RF-01	**Conversión a Modelos ES**
Descripción	La herramienta debe ser capaz de transformar un conjunto de ADEs, de un sistema continuo o híbrido, en un modelo ES para la herramienta de CPC (Cálculo de Posibles Conflictos, CPC, con la que interacciona).
Importancia	Vital
Comentarios	

RF-02	Generación de experimentos
Descripción	El sistema debe ser capaz de generar un conjunto de valores de salida, valiéndose de la ejecución a través de la herramienta matemática Matlab de una función, para un conjunto de ADEs que modelen un sistema continuo o discreto.
Importancia	Vital
Comentarios	

RF-03	**Generación de Modelos de Simulación**
Descripción	El sistema debe ser capaz de generar los modelos de simulación para sistemas de ADEs que modelen un sistema continuo o discreto.
Importancia	Vital
Comentarios	

RF-04	Diagnosis
Descripción	El sistema debe ser capaz de ejecutar una diagnosis utilizando la herramienta Matlab para sistemas de ADEs que modelen un sistema continuo o discreto.
Importancia	Vital
Comentarios	

RF-05	**Fusión de Modelos Evaluables Minimales**
Descripción	El sistema debe ser capaz de fusionar aquellos Modelos con partes comunes generando un Modelo más complejo que pueda cubrir más de un modo de funcionamiento.
Importancia	Alta
Comentarios	

RF-06	**Impresión de datos**
Descripción	El sistema permitirá imprimir o guardar en distintos formatos los resultados de distintos cálculos que realiza así como los datos cargados.
Importancia	Alta
Comentarios	

RF-07	**Generación de la Función Simulación**
Descripción	El sistema debe ser capaz de generar un fichero de extensión Matlab a partir de sistemas de ADEs que modelen un sistema continuo o discreto tanto para la Generación de Experimentos como para la Diagnosis.
Importancia	Alta
Comentarios	

RF-08	**Sistema de entrada**
Descripción	El sistema debe ser compatible hacia atrás para aceptar los tipos de sistemas que aceptaba la versión DxPCs 1.0.
Importancia	Alta
Comentarios	

3.4. REQUISITOS DE INTERFAZ EXTERNO

3.4.1. Interfaces de usuario

IU-01	Interfaz del sistema
Descripción	El sistema debe ser capaz de mostrar al usuario tanto los elementos cargados como los resultados de la ejecución de la funcionalidad de cada aparatado.
Importancia	Vital
Comentarios	

3.4.2. Interfaces de software

IS-01	Formato de las ADEs y Modelos
Descripción	El sistema debe ser capaz de cargar las ADEs y los Modelos obtenidos de la herramienta de CPCs tanto con un sistema continuo como discreto.
Importancia	Vital
Comentarios	

3.5. REQUISITOS DE INFORMACIÓN

RI-01	Información de ficheros
Descripción	El sistema debe ser capaz de leer la información sobre las ADEs y los Modelos tanto con un sistema continuo como discreto.
Importancia	Vital
Comentarios	

RI-02	Información de resultados
Descripción	El sistema debe ser capaz de guardar la información sobre los resultados obtenidos de cada funcionalidad.
Importancia	Alta
Comentarios	

3.6. REQUISITOS DE DISEÑO

RD-01	Estándar de la herramienta
Descripción	El sistema debe manejar los ficheros E/S acordes al estándar de la herramienta así como los ficheros obtenidos directamente de la salida de la herramienta de CPCs.
Importancia	Vital
Comentarios	

3.7. REQUISITOS NO FUNCIONALES

RNF-01	Lenguaje de programación
Descripción	El sistema debe funcionar en el lenguaje Java ya que las versiones anteriores estaban hechas en este lenguaje.
Importancia	Alta
Comentarios	

RNF-02	Soporte multiplataforma
Descripción	El sistema debe funcionar correctamente bajo cualquier sistema operativo en el que pueda instalarse una máquina virtual Java.
Importancia	Alta
Comentarios	

RNF-03	Eficiencia
Descripción	El sistema debe realizar los cálculos de forma eficiente.
Importancia	Alta
Comentarios	

RNF-04	Fiabilidad del sistema
Descripción	El sistema debe realizar los cálculos de forma correcta.
Importancia	Alta
Comentarios	

RNF-05	Facilidad de uso
Descripción	El sistema debe ser fácil de usar para un usuario con conocimientos básicos sobre diagnóstico basado en modelos y los conceptos de Posibles Conflictos, por ejemplo, estudiantes de postgrado.
Importancia	Alta
Comentarios	

4. MODELO DE ANÁLISIS

4.1. INTRODUCCIÓN

4.1.1. Propósito

El propósito de este apartado, Modelo de Análisis, es dar una descripción de alto nivel del sistema que será la base en la cual se apoye el diseño así como la posterior implementación.
Este apartado será realizado por los analistas del equipo y será utilizado por los diseñadores para realizar el diseño de la herramienta.

4.1.2. Alcance

Este apartado proporcionará las pautas de alto nivel para realizar el proyecto "DxPCs 2.0 - Ampliación de la plataforma para diagnosis basada en modelos".

4.1.3. Definiciones, Acrónimos, Abreviaturas

Véase el glosario general del proyecto.

4.1.4. Perspectiva General

En este capítulo encontramos la siguiente información:

- Perspectiva general: descripción de las principales funcionalidades de la herramienta y las nuevas capacidades que le añade este trabajo.
- Modelo de casos de uso: contiene la información de las funciones del sistemas y los pasos que se da en cada una de ellas.
- Modelado estructural del sistema: es una primera aproximación de la estructura de las clases que contiene el sistema, almacenamiento de información y métodos.
- Modelo de comportamiento: mostrará con un diagrama el comportamiento de la herramienta.
- Modelo de iteración: describirá cómo se relacionan los elementos del sistema.

4.2. PERSPECTIVA GENERAL

Partiendo de la base de una herramienta que funciona para sistemas continuos, modelados mediante un conjunto de ADEs, nuestro objetivo principal será modificar todo aquello que sea necesario en la herramienta para que acepte modelos híbridos, modelados con ecuaciones en las que hay condiciones de uso, y se mantenga la compatibilidad con versiones anteriores. La condición asociada a una ecuación del modelo implica que el valor de cierta ecuación al que acompañe puede darse o no, o puede tener un valor u otro dependiendo de los valores de la condición.

La introducción de los sistemas híbridos implicará los siguientes cambios:

- Modificar el sistema de lectura de ficheros de sistemas híbridos.
- Modificar las estructuras que guardan la información y/o crear nuevas para guardar los modelos de los sistemas híbridos.
- Adaptar los algoritmos existentes para trabajar con sistemas híbridos además de con los continuos.
- Proponer un nuevo modo de almacenamiento de las condiciones de uso de las ecuaciones para hacer más eficiente su manejo.

4.3. MODELO DE CASOS DE USO

4.3.1. Casos de uso de la herramienta

En azul marcamos aquellos casos de uso en los cuales ha habido modificaciones, el resto con el mismo color del fondo.

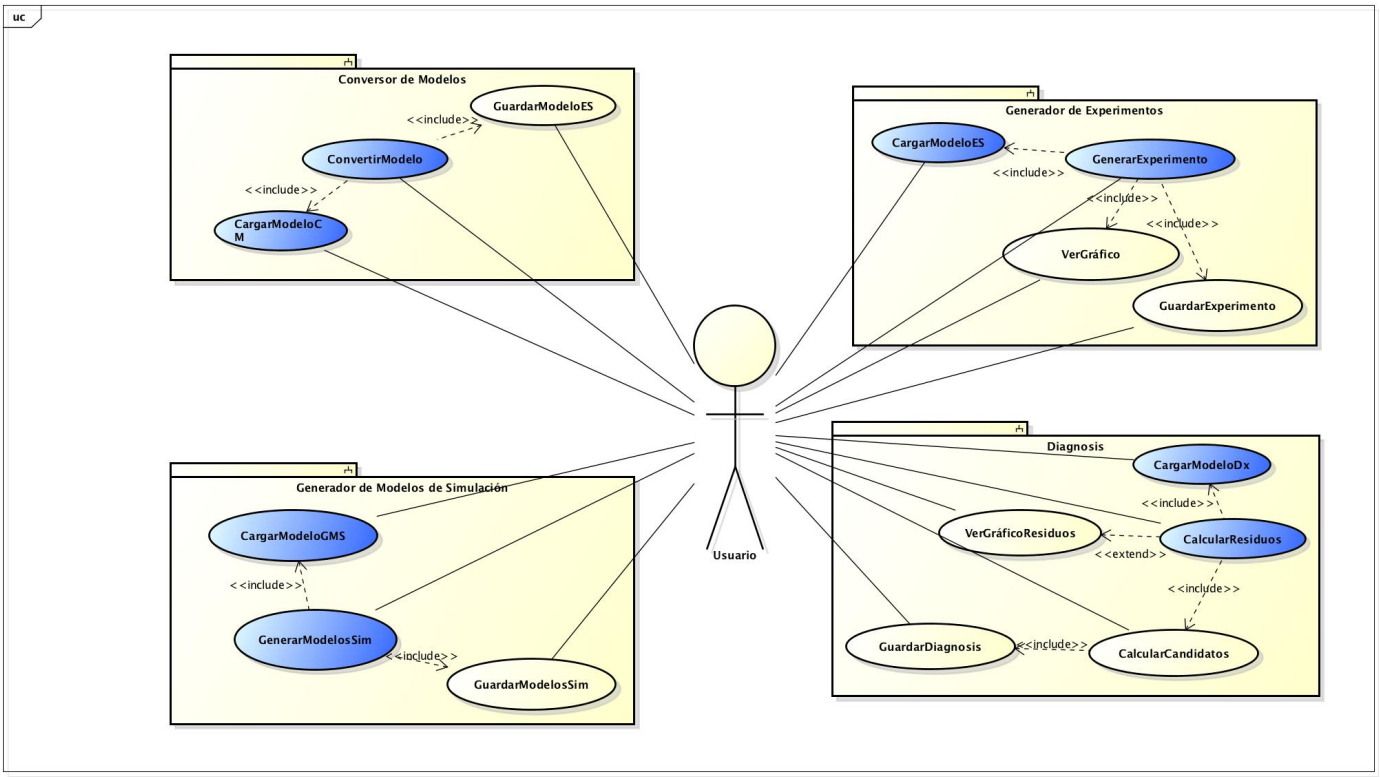


Figura 4.1 : Diagrama de casos de uso

4.3.2. Casos de uso del Conversor de Modelos E/S

CU-01	CargarModeloCM	
Descripción	El sistema debe comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee cargar los ficheros necesarios para generar un Modelo E/S.	
Precondiciones	Los ficheros están escritos de forma correcta respetando el estándar.	
Secuencia Normal	Paso	Acción
	1	El usuario selecciona la opción “Load Model”.
	2	El sistema de ofrecerá al usuario la carga “One by one” o “Using a path file”.
	3	El usuario escoge una de las dos opciones (3.1 – 3.2).
	3.1	El usuario escoge carga “One by one”.
	3.1.1	El sistema va ofreciendo uno a uno la elección de cada uno de los ficheros que necesita.
	3.1.2	El usuario elige uno a uno los ficheros.
	3.1.3	El sistema va leyendo uno a uno los ficheros elegidos.
	3.2	El usuario escoge “Using a path file”.
	3.2.1	El sistema lee el fichero de Rutas y carga los ficheros que necesita.
	4	El sistema muestra los elementos cargados.
	5	El caso de uso finaliza.
Postcondiciones	El sistema está cargado correctamente.	
Excepciones	Paso	Acción
	3.1.3	El fichero está incorrecto, se muestra un error y finaliza el caso de uso.
	3.2.1	Hay rutas o ficheros incorrectos, se muestra un error y finaliza el caso de uso.
Rendimiento	La carga del sistema se realizará de forma instantánea.	
Importancia	Alta	
Comentarios		

CU-02	Convertir Modelo	
Descripción	El sistema debe comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee transformar un sistema cargado en un Modelo E/S.	
Precondiciones	El sistema está cargado correctamente.	
Secuencia Normal	Paso	Acción
	1	El usuario selecciona la opción “Convert Model”.
	2	El sistema realizará las operaciones necesarias y mostrará el resultado.
	3	El caso de uso finaliza.
Postcondiciones	El modelo E/S resultante aparece en la interfaz.	
Excepciones	Paso	Acción
	-	-
Rendimiento	La ejecución se realizará de forma prácticamente instantánea.	
Importancia	Alta	
Comentarios		

Los siguientes casos de uso no han sido modificados sustancialmente y se conservan del prototipo anterior. Se adjuntan por complitud de la documentación.

CU-03	GuardarModeloES	
Descripción	El sistema debe comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee guardar el resultado de la conversión del Modelo ES.	
Precondiciones	El sistema ha realizado la conversión correctamente.	
Secuencia Normal	Paso	Acción
	1	El usuario selecciona la opción “Save” en la pestaña “SS/IO Notation converter”.
	2	El sistema pedirá al usuario que elija la ruta y el nombre del archivo.
	3	El usuario elige la ruta y el nombre.
	4	El sistema guarda los resultados en formato .txt.
	5	El caso de uso finaliza.
Postcondiciones	El txt está generado.	
Excepciones	Paso	Acción
	-	-
Rendimiento	La ejecución se realizará de forma prácticamente instantánea.	
Importancia	Alta	
Comentarios		

4.3.3. Casos de uso del Generador de Modelos de Simulación

CU-04	CargarModeloGMS	
Descripción	El sistema debe comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee cargar los ficheros necesarios para generar los Modelos de Simulación.	
Precondiciones	Los ficheros están escritos de forma correcta respetando el estándar.	
Secuencia Normal	Paso	Acción
	1	El usuario selecciona la opción “Load Model”.
	2	El sistema de ofrecerá al usuario la carga “One by one” o “Using a path file”.
	3	El usuario escoge una de las dos opciones (3.1 – 3.2).
	3.1	El usuario escoge carga “One by one”.
	3.1.1	El sistema va ofreciendo uno a uno la elección de cada uno de los ficheros que necesita.
	3.1.2	El usuario elige uno a uno los ficheros.
	3.1.3	El sistema va leyendo uno a uno los ficheros elegidos.
	3.2	El usuario escoge “Using a path file”.
	3.2.1	El sistema lee el fichero de Rutas y carga los ficheros que necesita.
	4	El sistema muestra los elementos cargados.
	5	El caso de uso finaliza.
Postcondiciones	El sistema está cargado correctamente.	
Excepciones	Paso	Acción
	3.1.3	El fichero está incorrecto, se muestra un error y finaliza el caso de uso.

	3.2.1	Hay rutas o ficheros incorrectos, se muestra un error y finaliza el caso de uso.
Rendimiento	La carga del sistema se realizará de forma instantánea.	
Importancia	Alta	
Comentarios		

CU-05	GenerarModeloSim	
Descripción	El sistema debe comportarse de la siguiente forma cuando el usuario desee generar los modelos de simulación asociados a los Posibles Conflictos a partir de los ficheros cargados	
Precondiciones	El sistema está cargado correctamente.	
Secuencia Normal	Paso	Acción
	1	El usuario selecciona la opción “Generate(Simulation)”
	2	El sistema realizará las operaciones necesarias, procederá primero con la fusión de los MEM cargados. Con la fusión realizada construirá uno a uno cada Modelo de Simulación.
	3	El caso de uso finaliza.
Postcondiciones	El sistema devuelve los Modelos de Simulación, al menos uno para cada Posible Conflicto en el sistema.	
Excepciones	Paso	Acción
	-	-
Rendimiento	La ejecución se realizará de forma prácticamente instantánea.	
Importancia	Alta	
Comentarios		

Los siguientes casos de uso no han sido modificados sustancialmente y se conservan del prototipo anterior. Se adjuntan por complitud de la documentación.

CU-06	GuardarModeloSim	
Descripción	El sistema debe comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee guardar el resultado de la generación de los Modelos de Simulación	
Precondiciones	El sistema ha realizado la generación correctamente.	
Secuencia Normal	Paso	Acción
	1	El usuario selecciona la opción “Save” en la pestaña “Simulation Model Generator”.
	2	El sistema pedirá al usuario que elija la ruta y el nombre del archivo.
	3	El usuario elige la ruta y el nombre.
	4	El sistema guarda los resultados en formato .txt, creando uno para cada Modelo.
	5	El caso de uso finaliza.
Postcondiciones	Los txt están generados.	
Excepciones	Paso	Acción
	-	-
Rendimiento	La ejecución se realizará de forma prácticamente instantánea.	
Importancia	Alta	
Comentarios		

4.3.4. Casos de uso del Generador de Experimentos

CU-07	CargarModeloES	
Descripción	El sistema debe comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee cargar los ficheros necesarios para generar un Experimento.	
Precondiciones	Los ficheros están escritos de forma correcta respetando el estándar.	
Secuencia Normal	Paso	Acción
	1	El usuario selecciona la opción “Load Model”.
	2	El sistema de ofrecerá al usuario la carga “One by one” o “Using a path file”.
	3	El usuario escoge una de las dos opciones (3.1 – 3.2).
	3.1	El usuario escoge carga “One by one”.
	3.1.1	El sistema va ofreciendo uno a uno la elección de cada uno de los ficheros que necesita.
	3.1.2	El usuario elige uno a uno los ficheros.
	3.1.3	El sistema va leyendo uno a uno los ficheros elegidos.
	3.2	El usuario escoge “Using a path file”.
	3.2.1	El sistema lee el fichero de Rutas y carga los ficheros que necesita.
	4	El sistema muestra los elementos cargados.
	5	El caso de uso finaliza.
Postcondiciones	El sistema está cargado correctamente.	
Excepciones	Paso	Acción
	3.1.3	El fichero está incorrecto, se muestra un error y finaliza el caso de uso.
	3.2.1	Hay rutas o ficheros incorrectos, se muestra un error y finaliza el caso de uso.
Rendimiento	La carga del sistema se realizará de forma instantánea.	
Importancia	Alta	
Comentarios		

CU-08	GenerarExperimento	
Descripción	El sistema debe comportarse de la siguiente forma cuando el usuario desee generar un experimento para un sistema determinado. Consiste en simular el comportamiento del sistema durante un tiempo dado, para unas señales de entrada dadas y en presencia de cero o más cambios de modo y de cero o más modos de fallo.	
Precondiciones	El sistema ya está cargado correctamente.	
Secuencia Normal	Paso	Acción
	1	El usuario selecciona la opción “Generate Experiment”.
	2	El sistema realiza las operaciones necesarias para generar el experimento a través de Matlab y muestra al usuario los resultados.
	3	El caso de uso finaliza.
Postcondiciones	El sistema devuelve el resultado del experimento.	
Excepciones	Paso	Acción

	2	No se puede establecer conexión con Matlab entonces el caso de uso finaliza.
Rendimiento	Dependerá del tiempo que tarde el pc en ejecutar Matlab ya que el tiempo que utiliza la herramienta en operaciones propias es casi instantáneo.	
Importancia	Alta	
Comentarios		

Los siguientes casos de uso no han sido modificados sustancialmente y se conservan del prototipo anterior. Se adjuntan por complitud de la documentación.

CU-09	GuardarExperimento	
Descripción	El sistema debe comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee guardar el resultado generación del experimento.	
Precondiciones	El sistema ha realizado la generación correctamente.	
Secuencia Normal	Paso	Acción
	1	El usuario selecciona la opción “Save” en la pestaña “Experiment Generator”.
	2	El sistema pedirá al usuario que elija la ruta y el nombre del archivo.
	3	El usuario elige la ruta y el nombre.
	4	El sistema guarda los resultados en formato .txt.
	5	El caso de uso finaliza.
Postcondiciones	El txt está generado.	
Excepciones	Paso	Acción
	-	-
Rendimiento	La ejecución se realizará de forma prácticamente instantánea.	
Importancia	Alta	
Comentarios		

CU-10	VerGrafico	
Descripción	El sistema debe comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee ver las gráficas de variables del resultado de generación del experimento.	
Precondiciones	El sistema ha realizado la generación correctamente.	
Secuencia Normal	Paso	Acción
	1	El usuario selecciona la opción “Display Graph” en la pestaña “Experiment Generator”.
	2	El sistema pedirá al usuario que elija las variables que desee ver la gráfica.
	3	El usuario selecciona las variables.
	4	El sistema muestra la gráfica.
	5	El caso de uso finaliza.
Postcondiciones	La gráfica ha sido generada.	
Excepciones	Paso	Acción
	-	-
Rendimiento	La ejecución se realizará de forma prácticamente instantánea.	
Importancia	Alta	
Comentarios		

4.3.5. Casos de uso de Diagnosis

CU-11	CalcularResiduos	
Descripción	El sistema debe comportarse de la siguiente forma cuando el usuario desee cargar los ficheros necesarios para calcular los residuos. El residuo es la diferencia entre el valor medido y el valor real y se puede calcular de tres formas: - Differences Absolute Values : calculando las diferencias de los valores absolutos de los resultados obtenidos. - Differences : calculando las diferencias entre los resultados obtenidos. - Euclidean Distances : calculando las diferencias entre los resultados obtenidos utilizando la fórmula de la distancia euclidea. -	
Precondiciones	Los ficheros están escritos de forma correcta respetando el estándar.	
Secuencia Normal	Paso	Acción
	1	El usuario selecciona la opción “Calculate Residuals”
	2	El sistema muestra 3 opciones posibles para el cálculo
	3	El usuario escoge una de las 3 opciones
	3.1	El usuario escoge “Differences Absolute Values”
	3.2	El usuario escoge “Differences”
	3.3	El usuario escoge “Euclidean Distances”
	4	El sistema realiza las operaciones necesarias para calcular los residuos a través de Matlab y muestra al usuario los resultados.
	5	El caso de uso finaliza.
Postcondiciones	Como resultado se calculan los residuos para los diferentes PCs activos en el sistema.	
Excepciones	Paso	Acción
	4	No se puede establecer conexión con Matlab entonces el caso de uso finaliza.
Rendimiento	Dependerá del tiempo que tarde el ordenador en ejecutar Matlab ya que el tiempo que utiliza la herramienta en operaciones propias es casi instantáneo.	
Importancia	Alta	
Comentarios	Una vez dados los residuos para cada PC y su descripción estructural se calculan los candidatos a fallo.	

CU-12	CargarModeloDX	
Descripción	El sistema debe comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee cargar los ficheros necesarios para generar una Diagnosis.	
Precondiciones	Los ficheros están escritos de forma correcta respetando el estándar.	
Secuencia Normal	Paso	Acción
	1	El usuario selecciona la opción “Load Model”.
	2	El sistema de ofrecerá al usuario la carga “One by one” o “Using path file”.
	3	El usuario escoge una de las dos opciones (3.1 – 3.2).
	3.1	El usuario escoge carga “One by one”.
	3.1.1	El sistema va ofreciendo uno a uno la elección de cada uno de los ficheros que necesita.
	3.1.2	El usuario elige uno a uno los ficheros.
	3.1.3	El sistema va leyendo uno a uno los ficheros elegidos.
	3.2	El usuario escoge “Using path file”.
	3.2.1	El sistema lee el fichero de Rutas y carga los ficheros que necesita.
	4	El sistema muestra los elementos cargados.
	5	El caso de uso finaliza.
Postcondiciones	El sistema está cargado correctamente.	
Excepciones	Paso	Acción
	3.1.3	El fichero está incorrecto, se muestra un error y finaliza el caso de uso.
	3.2.1	Hay rutas o ficheros incorrectos, se muestra un error y finaliza el caso de uso.
Rendimiento	La carga del sistema se realizará de forma instantánea.	
Importancia	Alta	
Comentarios		

Los siguientes casos de uso no han sido modificados sustancialmente y se conservan del prototipo anterior. Se adjuntan por complitud de la documentación.

CU-13	GuardarDiagnosis	
Descripción	El sistema debe comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee guardar el resultado de la diagnosis.	
Precondiciones	El sistema ha realizado el calculo de residuos y de candidatos correctamente.	
Secuencia Normal	Paso	Acción
	1	El usuario selecciona la opción “Save Results” en la pestaña “Diagnosis”.
	2	El sistema pedirá al usuario que elija la ruta y el nombre del archivo.
	3	El usuario elige la ruta y el nombre.
	4	El sistema guarda los resultados en formato .txt.
	5	El caso de uso finaliza.
Postcondiciones	El txt está generado.	
Excepciones	Paso	Acción

	-	-
Rendimiento	La ejecución se realizará de forma prácticamente instantánea.	
Importancia	Alta	
Comentarios		

CU-14	VerGraficoResiduos	
Descripción	El sistema debe comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee ver las gráficas de los residuos que generan cada uno de los PCs.	
Precondiciones	El sistema ha realizado la generación correctamente.	
Secuencia Normal	Paso	Acción
	1	El usuario selecciona la opción “Display Graph” en la pestaña “Diagnosis”.
	2	El sistema pedirá al usuario que elija el nodo de discrepancia del cual desea ver el residuo.
	3	El usuario selecciona el nodo.
	4	El sistema muestra la gráfica.
	5	El caso de uso finaliza.
Postcondiciones	La gráfica ha sido generada.	
Excepciones	Paso	Acción
	-	-
Rendimiento	La ejecución se realizará de forma prácticamente instantánea.	
Importancia	Alta	
Comentarios		

CU-15	CalcularCandidatos	
Descripción	El sistema debe comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee cargar los ficheros necesarios para realizar el calculo de candidatos.	
Precondiciones	Se ha realizado previamente el cálculo de residuos.	
Secuencia Normal	Paso	Acción
	1	El usuario selecciona la opción “Fault Candidates Calculation”.
	2	El sistema pide al usuario que escoja el cálculo por “Equations” o “Components”
	3	El usuario escoge una de las dos opciones (3.1 – 3.2).
	3.1	El usuario escoge cálculo por “Equations”
	3.2	El usuario escoge cálculo por “Components”
	4	El sistema pide al usuario la ruta del fichero de PCs
	5	El usuario selecciona el fichero de PCs.
	6	El sistema pide al usuario que elija la forma de calcular los candidatos, “Run Z-test” , “Use the global value defined in the configuration file”, “Insert threshold value for each Possible Conflict”.
	6.1	El usuario escoge la opción “Run Z-test”.
	6.1.1	El sistema pide al usuario los datos para la ejecución del z-test.
	6.1.2	El sistema ejecuta el z-test.
	6.2.	El usuario escoge la opción “Use the global value defined in the configuration file”.

	6.2.1	El sistema realiza la buscada por el umbral definido en el fichero de configuración.
	6.3	El usuario escoge la opción “Insert threshold value for each Possible Conflict”.
	6.3.1	El sistema pide al usuario los umbrales de búsqueda para cada PC.
	6.3.2	El usuario introduce el umbral de búsqueda para cada PC.
	6.3.4	El sistema realiza la búsqueda por cada umbral definido.
	7	El sistema muestra los resultados del cálculo.
	8	El caso de uso finaliza.
Postcondiciones	Se ha mostrado los resultados del calculo de candidatos.	
Excepciones	Paso	Acción
	-	-
Rendimiento	El tiempo de ejecución será variable dependiendo del volumen de datos.	
Importancia	Alta	
Comentarios		

4.4. MODELADO ESTRUCTURAL DEL SISTEMA

En este apartado daremos una primera aproximación a la estructura de clases, almacenamiento de la información y métodos que va a tener nuestro sistema. Ésta será una visión estática del mismo en que no se plantearán interacciones entre los diferentes elementos o módulos del sistema.

4.4.1. Diagrama de clases

En este apartado vamos a mostrar el diagrama de clases del sistema el que se verán diferenciadas las clases modificadas con respecto a la versión original en un color más oscuro y dentro de las mismas se mostrará en color azul aquellos atributos y operaciones nuevos o modificados.

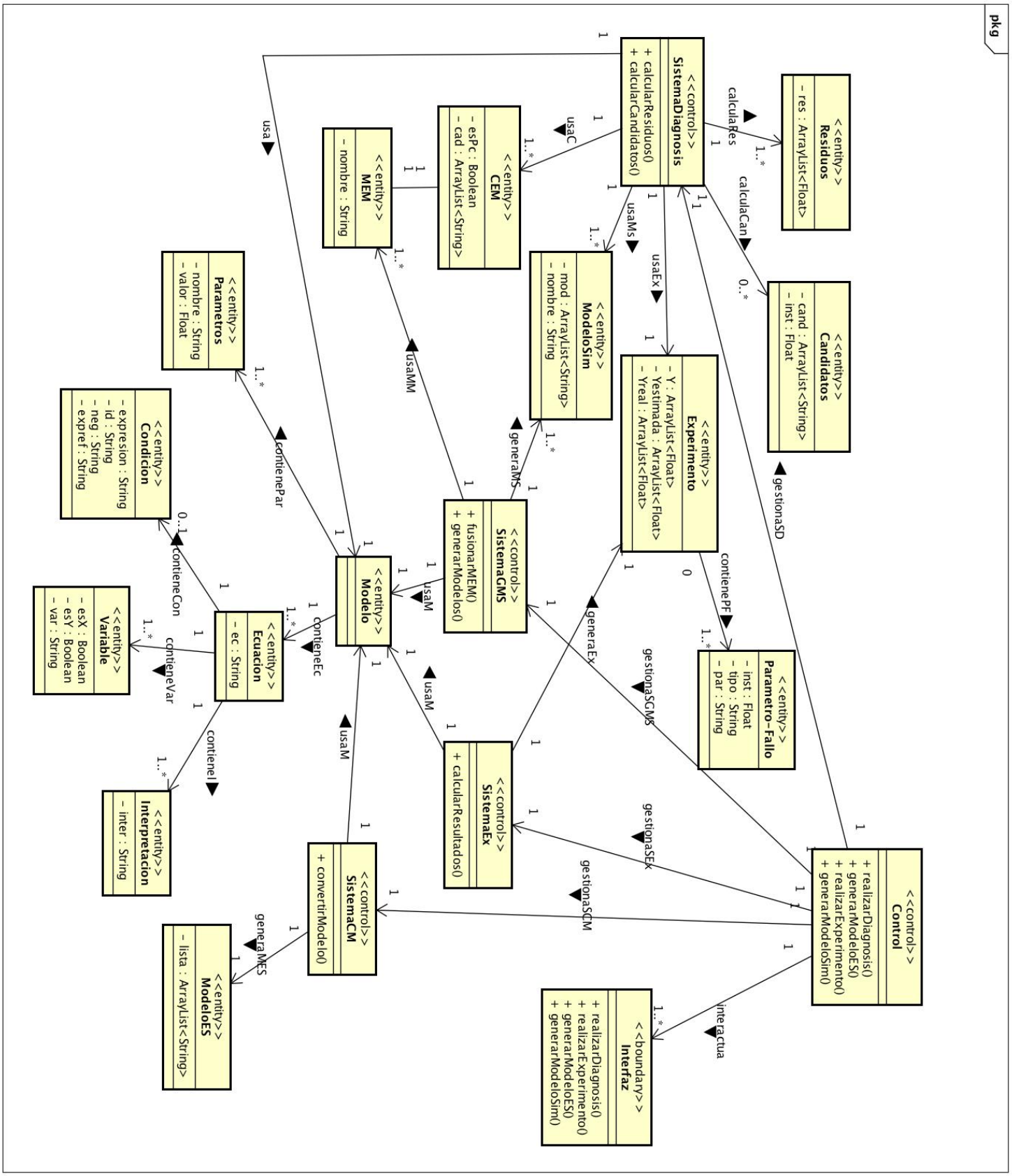


Figura 4.2 : Diagrama de clases

Para comprender mejor el diagrama de clases vamos a realizar unos aportes explicando la funcionalidad de cada una de las clases que se han mencionado en el diagrama.

- Modelo: es la clase que almacenará toda la información relativa al sistema que se introducirá en la aplicación para trabajar con él.
- Ecuacion: es la clase que almacenará las ecuaciones que componen los modelos de los sistemas.
- Condicion: es la clase que almacenará los datos asociados a una condición, es decir, la expresión que indica la validez de una ecuación, la expresión en forma prefija, su identificador y si la hubiera, la expresión de la condición del else.
- Variable: es la clase que almacenará cada una de las variables que contienen las ecuaciones del modelo del sistema.
- Parametro-Fallo: es la clase que almacenará la información relativa al error o errores que se puedan introducir en el sistema.
- CEM: es la clase que almacenará la información relativa a los elementos que forman la cadena evaluable minimal y además se podrá definir si la cadena es o no no es Posible Conflicto.
- MEM: CEM: es la clase que almacenará la información relativa a los Modelos evaluables Minimales a partir de los cuales que sirven de base para la generación de los Modelos de Simulación.
- Interpretación: es la clase que almacenará cada una de las distintas formas de interpretación de las ecuaciones del modelo del sistema.
- Interfaz: es la clase a través de la cual el usuario se comunica con el control del programa para realizar órdenes y obtiene los resultados de estos.
- Control: es la clase que coordina toda la actividad del sistema, se encarga de la comunicación con la interfaz y de lanzar las operaciones que realiza la aplicación encargando a los 4 subsistemas las tareas que convengan según la necesidad del usuario.
- SistemaCM: es uno de los 4 subsistemas de control. Cuando el usuario requiera un conversión de un sistema a la forma del Modelo Entrada/Salida, ES, será él el encargado de realizar las operaciones de conversión.
- ModeloES: es la clase que almacenará Modelo ES, resultado de la conversión que realiza el SistemaCM por medio de los datos que almacena Modelo.
- SistemaGMS: es uno de los 4 subsistemas de control. Cuando el usuario requiera la generación de los Modelos de Simulación, será él el encargado de realizar las operaciones de generación y opcionalmente, fusión.
- ModeloSim: es la clase que almacenará la información de los Modelos de Simulación que genera el SistemaGMS y que usará el SistemaDiagnosis para el cálculo de residuos.
- SistemaEx: es uno de los 4 subsistemas de control. Cuando el usuario requiera la generación de experimentos de diagnosis, donde se generarán datos asociados a un modo de funcionamiento para unas ciertas entradas; será él el encargado de realizar las operaciones necesarias.
- Experimento: es la clase que almacenará la información sobre los resultados generados por el SistemaEx.
- SistemaDiagnosis: es uno de los 4 subsistemas de control. Cuando el usuario requiera la generación de una diagnosis para un sistema y un experimento en concreto, será el encargado de realizarla. Primero mediante los Modelos de Simulación y los resultados del Experimento generará los residuos y una vez con ellos generados se encargará de buscar los posibles candidatos a fallo.

4.5. MODELO DE COMPORTAMIENTO

En este apartado se va a mostrar mediante un diagrama de secuencia los pasos de funcionamiento de la herramienta en un ciclo de funcionamiento básico.

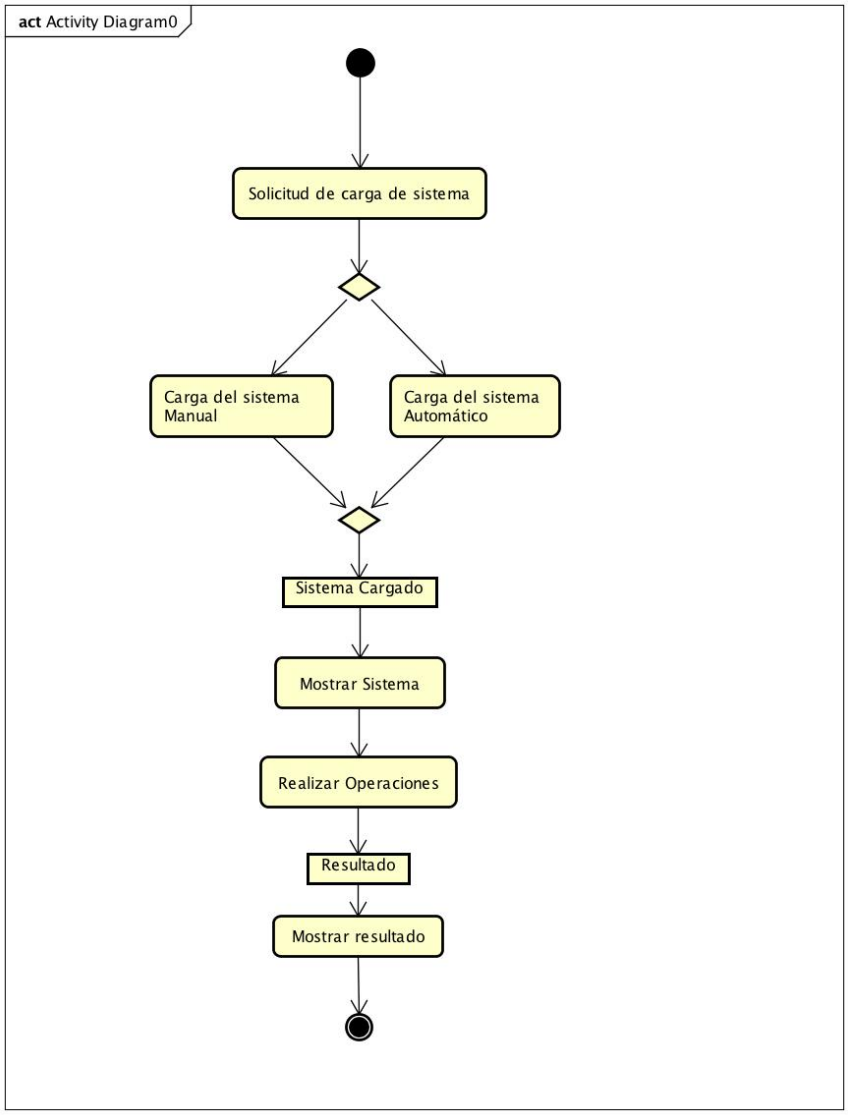


Figura 4.3 : Diagrama de actividad

Como vemos la secuencia de pasos es muy simple, primero se introduce la información pertinente en el sistema y con la información cargada ejecutaríamos alguna de las posibles operaciones de las cuales el sistema nos mostraría el resultado.

4.6. MODELO DE ITERACCIÓN

En esta sección se dará una visión general de la interacción que existe entre los elementos que se describieron en el modelo estructural. Para ello se mostrará un diagrama de secuencia para cada caso de uso que vimos en el punto apartado Modelado de Casos de Uso. Los elementos de los diagramas marcados el rojo son aquellos nuevos o que han sufrido modificaciones.

4.6.1. Escenarios en la aplicación

4.6.1.1. Escenarios en la carga de la información

En este caso los cuatro casos de usos tienen un escenario común, realizan operaciones distintas pero para el usuario el escenario será el mismo.

- Caso de uso CargarModeloCM
- Caso de uso CargarModeloES
- Caso de uso CargarModeloGMS
- Caso de uso CargarModeloDX

El usuario solicitará al sistema iniciar la carga del sistema pulsando en la opción "Load Model", el sistema le pedirá que decida entre la carga manual y la carga automática. El usuario decidirá entonces si quiere carga manual, donde tendrá que ir eligiendo uno a uno los ficheros que le pida el sistema o la carga automática donde deberá seleccionar el fichero donde se encuentran las rutas al resto de archivos necesarios para la carga. El sistema irá guardando toda la información según vaya leyendo cada fichero que se le vaya pasando y realizando las transformaciones de datos si lo necesitaran. Ya cargado la información relativa al sistema mostrará la información que se le ha suministrado en el panel izquierdo de la aplicación.

4.6.1.2. Escenarios en la ejecución de funciones

- Caso de uso ConvertirModelo:
  El usuario elegirá la opción "Convert Model", entonces el sistema generará a partir de la información que se ha cargado anteriormente un nuevo Modelo ES. Tras generar el sistema se lo mostrará al usuario en el panel derecho de la aplicación.

- Caso de uso GenerarExperimento:
  El usuario elegirá la opción "Generate Experiment", entonces el sistema tomará los datos introducidos para generar una función de Matlab que ejecutará en el mismo Matlab, obtendrá los resultados y se los mostrará en el panel derecho de la aplicación en forma de tabla al usuario.

- Caso de uso GenerarModeloSim:
  El usuario elegirá la opción "Generate(Simulation)", entonces el sistema mediante la información introducida en la carga obtendrá los Modelos de Simulación del sistema introducido fusionando aquellos que sean comunes. Tras tener los Modelos finales se los mostrará al usuario en el panel derecho de la aplicación.

- Caso de uso CalcularResiduo:
  El usuario elegirá la opción "Calculate Residuals", entonces el sistema le dará al usuario tres opciones para el cálculo. El usuario seleccionará uno de los modos, entonces el sistema tomará los datos introducidos para generar una función de Matlab que ejecutará en el mismo

Matlab, obtendrá los resultados y se los mostrará en el panel derecho de la aplicación en forma de tabla al usuario.

4.6.2. Diagramas de secuencia de la aplicación

En este apartado se van a mostrar los diagramas de secuencia del sistema del apartado de análisis que son los siguientes.

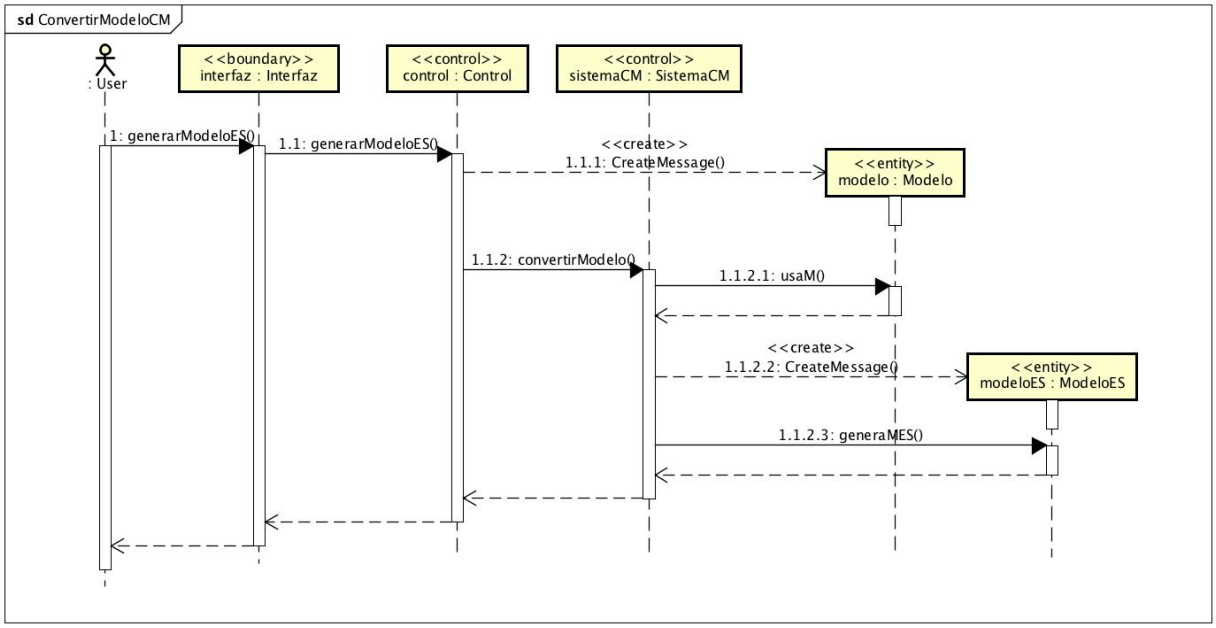


Figura 4.4 : Diagrama de secuencia ConvertirModelo

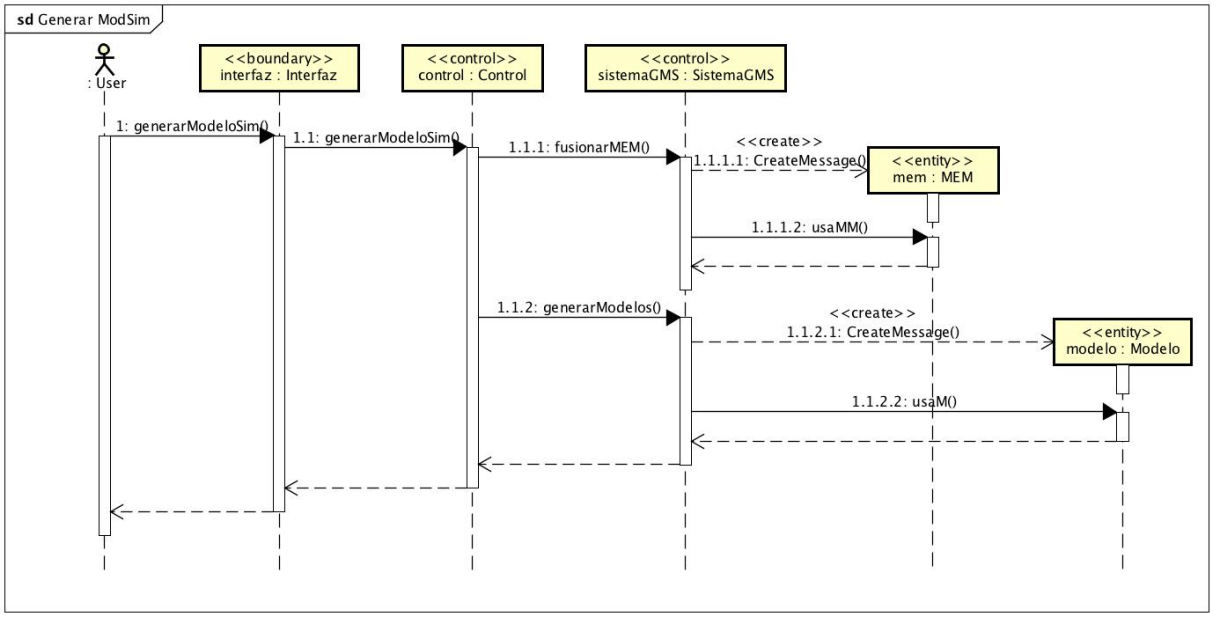


Figura 4.5 : Diagrama de secuencia GenerarModeloSim

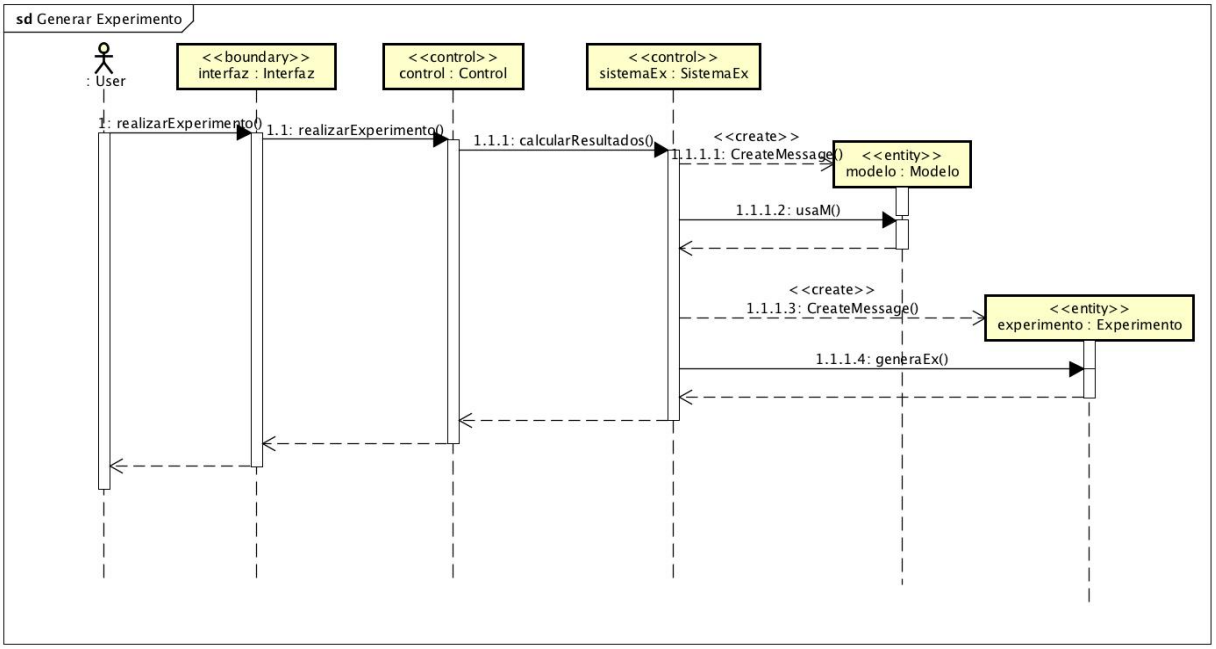


Figura 4.6 : Diagrama de secuencia GenerarExperimento

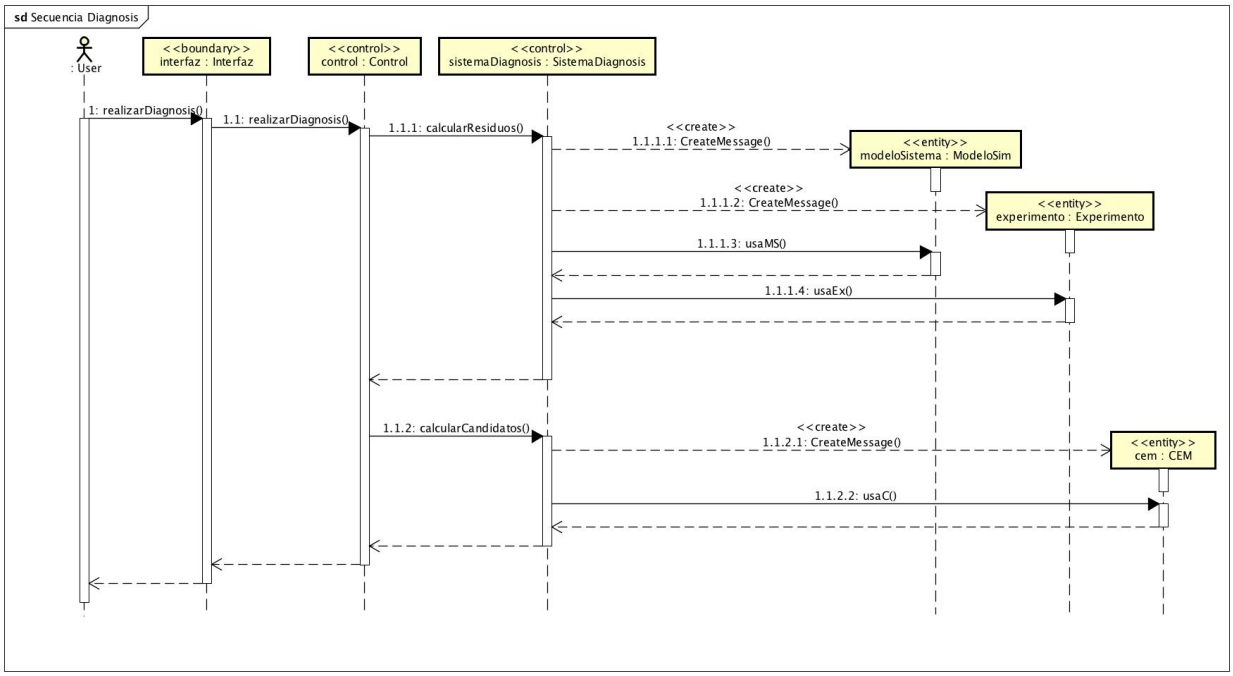


Figura 4.7 : Diagrama de secuencia Diagnosis

5. MODELO DE DISEÑO

5.1. INTRODUCCIÓN

5.1.1. Propósito

El propósito del modelo de diseño es definir de forma precisa la forma en que se va a llevar a cabo la implementación, de forma que el trabajo de los programadores sea lo más sencillo e inmediato posible. Además, un buen diseño será imprescindible para realizar una correcta división del trabajo en caso de existir varios equipos de desarrollo.

Los usuarios potenciales de este apartado de documentación serán los diseñadores, como autores de dicho apartado, y los programadores, puesto que ésta será su principal guía sobre cómo llevar a cabo la implementación.

5.1.2. Alcance

Este apartado proporcionará la forma de la cual debe llevarse a cabo la implementación del proyecto "DxPCs 2.0 - Ampliación de la plataforma para diagnosis basada en modelos".

5.1.3. Definiciones, Acrónimos, Abreviaturas

Véase el glosario general del proyecto.

5.1.4. Perspectiva General

Dentro de este capítulo podremos encontrar lo siguiente:

- Arquitectura del sistema: descripción general sobre cómo estará construido el sistema.
- Realización de casos de uso: refinamiento de los casos de uso vistos en el modelo de análisis para que éstos se adecúen al código que va a ser generado.
- Diseño de las clases: refinamiento de los diagramas de clases vistos en el modelo de análisis para que éstos se adecúen al código que va a ser generado.
- Modelo de interacción: refinamiento de los diagramas de secuencia vistos en el modelo de análisis para que éstos se adecúen al código que va a ser generado.
- Diagramas de secuencia: diagramas de secuencia de diseño de los casos de usos en los cuales hemos hecho modificaciones en nuestra aplicación.

5.2. ARQUITECTURA DEL SISTEMA

En esta sección se dará una idea general de cómo se va a construir el sistema. Puesto que se trata de una aplicación interactiva, dotada de una interfaz gráfica, como en la inmensa mayoría de este tipo de aplicaciones se va a hacer uso del patrón de diseño "Modelo-Vista-Controlador" (MVC).

5.2.1. El patrón MVC

Este patrón se llama Modelo-Vista-Controlador y se usa en aquellas aplicaciones que poseen una interfaz gráfica. Como su nombre indica las clases se agrupan en los tres paquetes que dan el nombre a este patrón.

- Modelo: en él se almacenará toda la información relativa a los datos con los que trabajará la aplicación.
- Vista: es la encargada de la comunicación con el usuario. Se encarga de hacer de intermediaria entre Usuario y Controlador, reconociendo las órdenes que le manda el usuario y pasándoselas al Controlador.
- Controlador: es el encargado de ejecutar las órdenes que le pasa la Vista y trasladar los cambios producidos al Modelo o sobre la propia Vista.

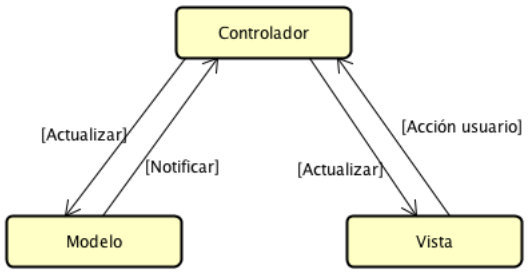


Figura 5.1 : Esquema del patrón MVC

Cualquier interacción que se produjera en el sistema seguiría estos pasos:

1. El Usuario manda la acción al paquete Vista.
2. El paquete Vista la identifica y envía su correspondiente orden al Controlador.
3. Mediante los datos del paquete Modelo el Controlador realiza las acciones necesarias.
4. Tras finalizar, el paquete Controlador envía los resultados al paquete Vista que se los muestra al Usuario.

5.3. DISEÑO DE SISTEMAS

5.3.1. SISTEMAS HÍBRIDOS

Para utilizar los nuevos sistemas híbridos primero vamos a decidir cómo definiremos las ecuaciones del sistema. Por ello un sistema podrá tener:

- Ecuaciones simples.
- Ecuaciones condicionadas: aquellas cuyo valor dependerá de una expresión con un condicionante. Serán del tipo:

```
if  <expresión condicionante>
        <ecuación>
[else
        <ecuación>]
end
```

Estas ecuaciones podrán llevar o no la parte del else según convenga en el sistema.
Las ecuaciones deberán tener la sintaxis conforme a la herramienta Matlab ya que de ella nos valemos para realizar cálculos posteriormente (Gilat, 2006).
Las expresiones también tendrán sintaxis conforme a Matlab. Se escribirán en forma infija tal que (x || y), (!x), (x && y)… y será la herramienta la cuál transforme las ecuaciones de forma infija a prefija facilitando la labor al usuario y evitando fallos en la escritura en forma prefija, algo más compleja. Los cambios que realizará la herramienta de forma infija a prefija son:

- X + Y → SM(X,Y)
- X - Y → RT(X,Y)
- X / Y → DV(X,Y)
- X * Y → MT(X,Y)
- X >= Y → GE(X,Y)
- X > Y → GT(X,Y)
- X <= Y → LE(X,Y)
- X < Y → LT(X,Y)
- X = Y → EQ(X,Y)
- -Y → NEG(X)
- X ~= Y → NEQ(X,Y)
- X || Y → or(X,Y)
- X && Y → and(X,Y)
- !X → not(X)

No todas estas funciones existen en Matlab por lo que algunas de ellas se escribirán y se añadirán directamente a Matlab de forma manual.

5.3.2. MODELO E/S

En este apartado se explica cómo la herramienta transformará nuestro sistema de ecuaciones en un Modelo E/S para la herramienta CPCs. Lo primero, cuando el sistema tenga la lista de las condiciones en forma prefija, comenzará la realización del catálogo. El catálogo es un medio que genera la herramienta automáticamente que almacena las condiciones denominándolas con el termino Sw_x , siendo x su identificador. A la hora de crear el modelo facilita el trabajo al ya tener cada condición asignada su nombre del modo Sw. El método de creación del catálogo será, para cada condición se deberá realizar una comparación con todas las demás condiciones guardadas y se podrán dar tres posibilidades.

1. La primera corresponde a que al realizar la comparación exista una condición igual, entonces a ésta se la dará el mismo nombre que a su igual.
2. La segunda será que exista una condición que sea su negación por lo que a esta condición se la dará el nombre de la otra negado. Por ejemplo si fuera igual a la condición SW1 sería !SW1.
3. Por último se podrá dar que sea una condición diferente al resto por lo cual se la dará un nuevo nombre. Si fuese la primera del catálogo sería SW1.

Una vez creado el catálogo, se comenzará con la transformación de las ADEs condicionadas al Modelo E/S. Los Modelos de E/S son la entrada para el cálculo de los PCs. Estos modelos están compuestos por relaciones que tienen dos partes: una parte estructural (que llamamos ecuación) y una parte causal (que llamamos interpretaciones y que son una de las formas en las que se puede resolver la ecuación, esto es, cómo se puede resolver una ecuación con una variable independiente cuando todas las demás variables se suponen conocidas o calculadas por parte de otra ecuación).
Se irán transformando una a una las ecuaciones con dos posibilidades:

1. Para cada ecuación simple el sistema considerará dos tipos de variables, las observadas y las no observadas:

   - Variables no observadas: las pertenecientes a los conjuntos X y W.
   - Variables observadas: las pertenecientes a los conjuntos Y y U.

   Tras realizar la clasificación de sus variables, se obtendrán sus interpretaciones, que pueden ser una o más y se obtiene entonces la ecuación en formato ES. Con estas consideraciones obtenemos que las ecuaciones simples que forman el modelo ES serán de la forma.

   El modelo completo sería un conjunto de ecuaciones tal que:

   <modelo> := { <ecuación>
   $}+
   $

   Y cada una de las ecuaciones sin condición del modelo:

   <ecuación> := <id-ecuacion>
   { <variables no observadas> }
   $
   { <variables observadas> }
   $
   { <Interpretación>
   $}+
   $

   Un modelo es una o más ecuaciones, separadas por $ y terminando la última con otro $. Cada uno de los dos conjuntos de variables, observadas y no observadas, se dividirá colocando una sola variable por línea. Las interpretaciones estarán formadas por la variable cabeza, la variable dependiente, con sus variables de cola, esto es las variables que se usan para calcular la cabeza, detrás, una por línea.. El orden de las variables de cola debajo de la cabeza no influye en el resultado.

2. Para las ecuaciones con condiciones debemos clasificar también sus variables de cola en observadas y en no observadas pero en este caso a estos dos conjuntos debemos añadirle también aquellas variables observadas o no observadas que pudieran aparecer en la condición de la ecuación. Tras clasificar las variables se colocarán las interpretaciones de

las ecuaciones según su condición sea la original o su negación, aunque la segunda parte, la de la negación que corresponde al else, puede darse el caso de no aparecer ya que es opcional. Por lo que las ecuaciones condicionadas que forman el modelo ES serán de la forma:

<ecuación> := <id-ecuacion>
{ <variables no observadas> }
$
{ <variables observadas> }
$
{ <Interpretación Sw_x>
#
Sw_x
$}+
$

Tras obtener la conversión de todas las ecuaciones el Modelo de E/S se terminará añadiendo un $ más al final de la última ecuación transformada que indica el fin del Modelo E/S.
Cabe destacar cómo se forma el conjunto de variables intermedias que denominamos W. Este conjunto contiene aquellas variables que no pertenecen a ninguno de los conjuntos predefinidos en el fichero de variables. Es un conjunto de variables no observadas. En la versión anterior este conjunto se generaba según se iba construyendo el modelo por lo que el orden de escritura de las ecuaciones era crucial ya que un orden incorrecto de la mismas podía hacer que no tuviéramos una variable de ese conjunto W en un momento determinado. Ahora este conjunto se generará antes de comenzar la conversión haciendo una búsqueda de sus variables ecuación a ecuación. El sistema pasará por todas las ADEs, para cada ecuación, se considera su variable cabeza, si ésta no pertenece a los conjuntos del fichero de variables, ni ya ha sido añadida al conjunto W anteriormente ya que formase parte de otra ecuación, la añadimos al mismo. Con esto facilitamos la tarea del usuario ya que ahora no tiene que estar pendiente del orden en que deberían de ir las ecuaciones cuando las escriba.

5.3.3. MODELOS DE SIMULACIÓN

En este apartado se explica cómo la herramienta creará valiéndose de la salida de la aplicación de CPC los nuevos Modelos de Simulación que ahora pueden ser híbridos. Se ha definido una nueva estructura para representar aquellas nuevas partes del Modelo de Simulación. En los nuevos Modelos de Simulación se podrán observar dos tipos de ecuaciones:

- Ecuaciones simples
- Ecuaciones condicionadas: aquellas cuyo valor dependerá de una expresión con un condicionante. Serán del tipo:

if <expresión condicionante>
    <ecuación>
end

Es decir sólo habrá ecuaciones condicionadas del tipo if-end. Se transformarán todas aquellas ecuaciones del tipo if-else-end ya que se ha considerado más conveniente a la hora de ver toda la información asociada a cada ecuación de manera más precisa.

Además se creará una nueva funcionalidad de mejora que consistirá en la fusión de aquellos Modelos de Simulación con partes equivalentes reduciendo así el número de los mismos y mejorando la visión de los resultados.

5.4. REALIZACIÓN DE CASOS DE USO

Los casos de uso que se corresponden a este apartado corresponden con los que ya se han mostrado anteriormente en anteriormente en el apartado de análisis.

5.5. DISEÑO DE LAS CLASES

En este apartado vamos a mostrar cómo quedará nuestro diagrama de clases al integrarlo en el patrón MVC que hemos descrito anteriormente. Al igual que en el caso anterior pondremos en azul todo aquello que hemos modificado o es nuevo.

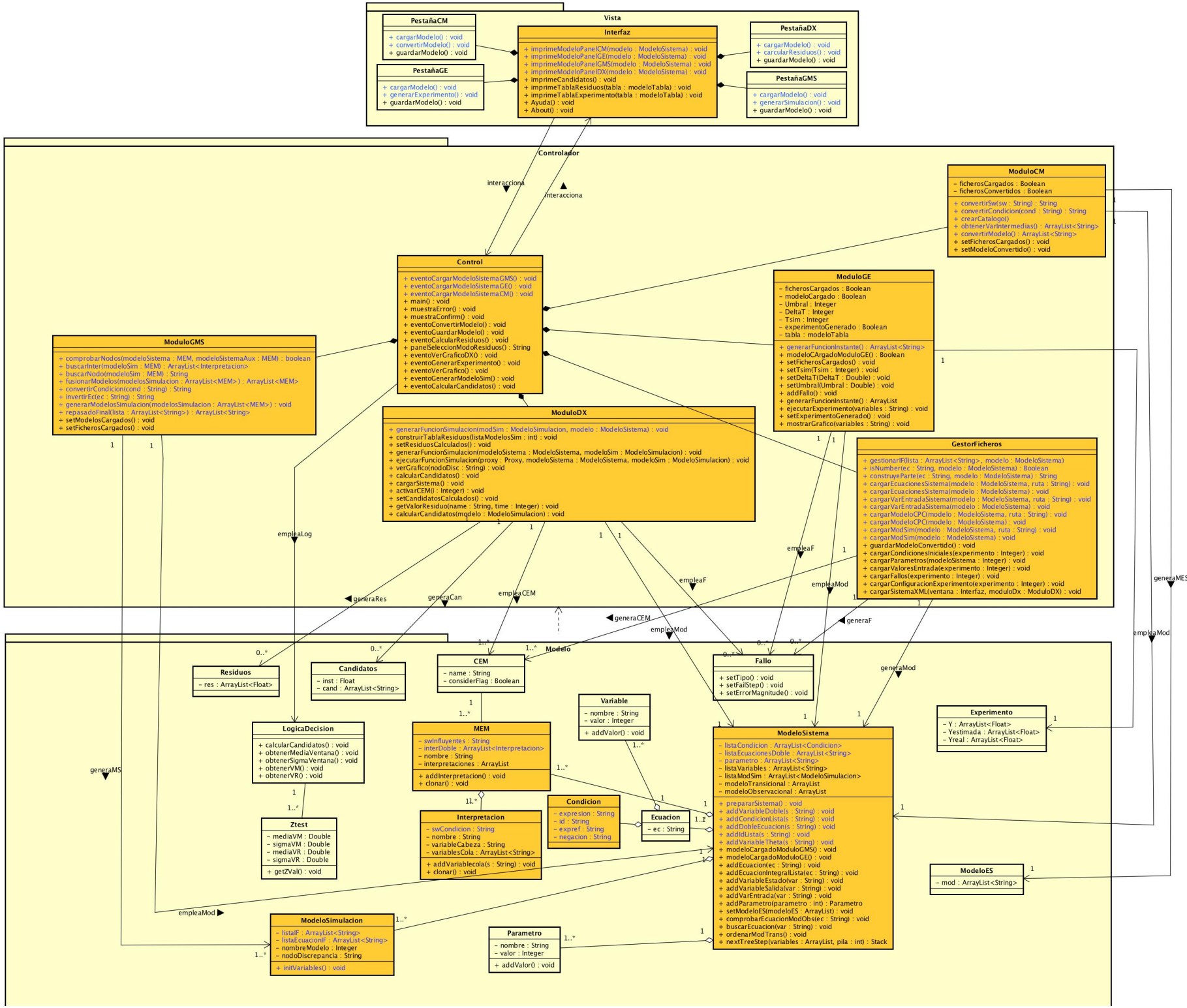


Figura 5.2 : Diagrama de clases - Patrón MVC

Ahora vamos explicar las clases modificadas explicando cada uno de los elementos modificados o añadidos en ellas.

ModeloSistema	
Responsabilidades	
Clase que almacena los datos sobre el sistema.	
Atributos	
Nombre(Tipo)	**Descripción**
listaCondicion(Arraylist<Condicion>)	Lista de los elementos condicion del sistema.
listaEcuacionesDoble(Arraylist<String>)	Lista de las ecuaciones con un elemento condición asociado.
parametros(Arraylist<String>)	Lista de variables Theta.
Operaciones	
prepararSistema()	
Objetivos	Construye los Modelos Observacional y Transicional.
Entrada	-
Salida	-
addCondicionLista()	
Objetivos	Añade una condición a Condicion.
Entrada	La condición a cargar.
Salida	-
addDobleEcuacion()	
Objetivos	Añade una ecuación del else a Condicion.
Entrada	La ecuación a cargar.
Salida	-
addIdLista()	
Objetivos	Añade un identificador a Condicion.
Entrada	El identificador a cargar.
Salida	
addVariablesTheta()	
Objetivos	Añade un nuevo parámetro a parámetros.
Entrada	La variable a cargar.
Salida	-

Condicion	
Responsabilidades	
Clase que almacena los datos sobre el sistema.	
Atributos	
Nombre(Tipo)	**Descripción**
expresion(String)	Expresión de una condición.
id(String)	Identificador de condición.
expref(String)	Expresión de una condición en forma prefija.
negacion(String)	Expresión de la condición del else.

Interpretacion	
Responsabilidades	
Clase que almacena los datos una interpretación.	
Atributos	
Nombre(Tipo)	**Descripción**
swCondicion(String)	Nombre del Sw que le afecta.

MEM	
Responsabilidades	
Clase que almacena los datos de un MEM.	
Atributos	
Nombre(Tipo)	**Descripción**
swInfluyentes(String)	Nombre del Sw o de los Sw que le afectan.
interDoble(ArrayList<Interpretacion>)	Lista de las interpretaciones extra que se añaden al modelo base tras la fusión.

ModeloSimulacion	
Responsabilidades	
Clase que almacena los datos de un Modelo de Simulación.	
Atributos	
Nombre(Tipo)	**Descripción**
listaIF(ArrayList<String>)	Lista de las condiciones.
listaEcuacionesIF(ArrayList<String>)	Lista de las ecuaciones asociadas a una condición.
Operaciones	
initVariables()	
Objetivos	Actualiza las variables del Modelo de Simulación.
Entrada	-
Salida	-

ModuloDX	
Responsabilidades	
Clase que ejecuta las funciones del modulo de diagnosis.	
Operaciones	
generarFuncionSimulacion()	
Objetivos	Genera una función para ejecutar la diagnosis a través de Matlab
Entrada	La información sobre el modelo del sistema y sobre el modelo de simulación.
Salida	-

ModuloGE	
Responsabilidades	
Clase que ejecuta las funciones del modulo de generación de experimentos.	
Operaciones	
generarFuncionInstante()	
Objetivos	Genera una función para ejecutar un experimento a través de Matlab.
Entrada	-
Salida	Lista con las variables del experimento.

Interfaz	
Responsabilidades	
Clase que imprime los datos cargados y los resultados.	
Operaciones	
imprimeModeloPanelCM()	
Objetivos	Muestra la información cargada en la pestaña Conversor de Modelos.
Entrada	La información sobre el modelo del sistema.
Salida	-
imprimeModeloPanelGE()	
Objetivos	Muestra la información cargada en la pestaña Generador de Experimentos.
Entrada	La información sobre el modelo del sistema.
Salida	-
imprimeModeloPanelGMS()	
Objetivos	Muestra la información cargada en la pestaña Generador de Modelos de Simulación.
Entrada	La información sobre el modelo del sistema.
Salida	-
imprimeModeloPanelIDX()	
Objetivos	Muestra la información cargada en la pestaña Diagnosis.
Entrada	La información sobre el modelo del sistema.
Salida	-

Control	
Responsabilidades	
Clase de control de la aplicación que gestiona los eventos.	
Operaciones	
eventoCargarModeloSistemaGMS()	
Objetivos	Evento de gestión de la carga del modelo de sistema en el apartado Generar Modelos de Simulación.
Entrada	-
Salida	-
eventoCargarModeloSistemaGE()	
Objetivos	Evento de gestión de la carga del modelo de sistema en el apartado Generador de Experimentos.
Entrada	-
Salida	-
eventoCargarModeloSistemaCM()	
Objetivos	Evento de gestión de la carga del modelo de sistema en el apartado Conversor de Modelos.
Entrada	-
Salida	-

ModuloCM	
Responsabilidades	
Clase que ejecuta las funciones del modulo de conversión de modelos.	
Operaciones	
convertirSw()	
Objetivos	Transforma un Sw en su inverso.
Entrada	El Sw a transformar.
Salida	El Sw transformado.
convertirCondicion()	
Objetivos	Transforma una condición en su inversa.
Entrada	La condición a transformar.
Salida	La condición transformada.
crearCatalogo()	
Objetivos	Crea el catálogo de los distintos Sw.
Entrada	-
Salida	-
obtenerVarIntermedias()	
Objetivos	Realiza una pasada sobre el sistema y obtiene la lista de variables intermedias para su posterior uso.
Entrada	-
Salida	La lista de variables intermedias.
convertirModelo()	
Objetivos	Crea a partir de la información del sistema el Modelo ES.
Entrada	-
Salida	El Modelo ES.

ModuloGMS	
Responsabilidades	
Clase que ejecuta las funciones del modulo de generador de modelos de simulación.	
Operaciones	
comprobarNodos()	
Objetivos	Comprueba si dos MEM tienen el mismo nodo de discrepancia.
Entrada	Los dos MEM.
Salida	True o False según el resultado de la comprobación.
buscarInter()	
Objetivos	Busca el nodo de discrepancia de un MEM (las ecuaciones).
Entrada	El MEM donde se va a buscar.
Salida	Un array con las discrepancias.
buscarNodo()	
Objetivos	Busca el nodo de discrepancia de un MEM (la variable).
Entrada	El MEM donde se va a buscar.
Salida	El nodo de discrepancia.
obtenerVarIntermedias()	
Objetivos	Realiza una pasada sobre el sistema y obtiene la lista de variables intermedias para su posterior uso.
Entrada	-
Salida	La lista de variables intermedias.
fusionarModelos()	
Objetivos	Fusiona aquellos modelos de simulación que son considerados equivalentes.

Entrada	La lista de los Modelos de Simulación.
Salida	La lista de los Modelos de Simulación fusionados.
generarModelosSimulacion()	
Objetivos	Genera los Modelos de Simulación.
Entrada	La lista de los Modelos de Simulación.
Salida	-
invertirEc()	
Objetivos	Transforma una ecuación en su inversa.
Entrada	El nombre de la ecuación a transformar.
Salida	El nombre de la ecuación transformado.
repasadoFinal()	
Objetivos	Función de apoyo, se encarga de repasar el resultado de generarModelosSimulacion() buscando partes del modelo incorrectas y corrigiéndolas.
Entrada	Un array con la lista que genera generarModelosSimulacion().
Salida	El array con las pertinentes correcciones.

GestorFicheros	
Responsabilidades	
Clase que se encarga de gestionar la información introducida al sistema.	
Operaciones	
gestionarIF()	
Objetivos	Transforma las condiciones de forma infija a prefija.
Entrada	La información sobre el modelo del sistema y la lista de las condiciones.
Salida	-
isNumber()	
Objetivos	Se encarga de averiguar si el parámetro pasado es un integer.
Entrada	La información sobre el modelo del sistema y el string que queremos ver si es un integer.
Salida	Un boolean true o false con el resultado.
construyeParte ()	
Objetivos	Función de apoyo a gestionarIF, es la que realiza realmente la transformación.
Entrada	La condición a transformar y los datos sobre el modelo de sistema.
Salida	-
cargarEcuacionesSistema()	
Objetivos	Carga las ecuaciones del sistema, tiene dos versiones, automática y manual.
Entrada(Manual)	La información sobre el modelo del sistema.
Entrada(Automática)	La información sobre el modelo del sistema y la ruta del fichero de rutas para el acceso a la información de forma automática.
Salida	-
cargarVarEntradaSistema()	
Objetivos	Carga las variables del sistema, tiene dos versiones, automática y manual.
Entrada(Manual)	La información sobre el modelo del sistema.
Entrada(Automática)	La información sobre el modelo del sistema y la ruta del fichero de rutas para el acceso a la información de forma automática.
Salida	-

cargarModeloCPC()	
Objetivos	Carga los MEM que le pasemos, tiene dos versiones, automática y manual.
Entrada(Manual)	La información sobre el modelo del sistema.
Entrada(Automática)	La información sobre el modelo del sistema y la ruta del fichero de rutas para el acceso a la información de forma automática.
Salida	-
cargarModSim()	
Objetivos	Carga las los Modelos de Simulación, tiene dos versiones, automática y manual.
Entrada(Manual)	La información sobre el modelo del sistema.
Entrada(Automática)	La información sobre el modelo del sistema y la ruta del fichero de rutas para el acceso a la información de forma automática.
Salida	-

5.6. MODELO DE INTERACCIÓN

En este apartado vamos a mostrar las interacciones que ocurren entre los elementos del sistema, como hicimos en el apartado de análisis pero con mucho más detalle en las clases implicadas.

5.6.1. Escenarios en la aplicación

5.6.1.1. Escenarios en la carga de la información

En este caso los cuatro casos de usos tienen un escenario común, realizan operaciones distintas pero para el usuario el escenario será el mismo.

- Caso de uso **CargarModeloCM**
- Caso de uso **CargarModeloES**
- Caso de uso **CargarModeloGMS**
- Caso de uso **CargarModeloDX**

Este caso de uso comienza con una petición al usuario de cargar un sistema, dicha petición se pasará al objeto de tipo control. Primero creará los objetos necesarios para guardar la información, el objeto modelo y según sea el tipo de carga creará el objeto que corresponda para la posterior ejecución. Si es **CargarModeloCM** creará un objeto tipo **ConversorModelos**, si es **CargarModeloGE** creará un objeto tipo **GeneradorExperimentos**, si es **CargarModeloGMS** creará un objeto tipo **GenradorModelosSimulacion** y si fuera **CargarModeloDX** creará un objeto tipo Diagnosis. Después control mediante un objeto tipo *GestorFicheros* mandará la ejecución de la carga del modelo pasando el control a *GestorFicheros* que cargará los datos en el objeto modelo en diferentes llamadas al objeto. Tras la carga, control mandará la transformación de las condiciones si las hubiera que también ejecutará *GestorFicheros*.

5.6.1.2. Escenarios en la ejecución de funciones

- Caso de uso **ConvertirModelo**:
  Este caso de uso comienza con una petición del usuario para ejecutar la conversión de modelo, dicha petición se pasará al objeto *control*. *Control* mediante el objeto creado en la carga, de tipo *ConversorModelos*, ejecuta la conversión que pasa el control a ese objeto. Entonces comienza la conversión. Lo primero que hace es crear el catálogo y generar el conjunto de variables intermedias. Una vez tenga toda la información que necesita comienza a generar el Modelo E/S. Se analizará ecuación a ecuación obteniendo la conveniente transformación. Con el Modelo E/S generado vuelve a *control* que manda la ejecución al objeto *ventana*, de tipo *Interfaz*, que muestre los resultados.

- Caso de uso **GenerarExperimento**:
  Este caso de uso comienza con una petición de usuario para ejecutar la generación de un experimento, dicha petición se pasará al objeto control. *Control* mediante el objeto creado en la carga, de tipo *GeneradorExperimento*, ejecuta primero la generación de la función para Matlab que pasa el control a ese objeto y el mismo crea la función de Matlab. La siguiente llamada de control será al mismo objeto que ha generado la función para que la ejecute a través de Matlab. Una vez realizada la ejecución esta vez control mandará al objeto ventana, de tipo Interfaz, que muestre los restultados.

- Caso de uso **GenerarModeloSim**:
  Este caso de uso comienza con una petición del usuario para ejecutar la generación de modelos de simulación, dicha petición se pasará al objeto *control*. Lo primero que se hace es mandar la ejecución del método *prepararSistema* mediante el objeto *modelo*, que lo que hace es generar los modelos observacional y transicional. Después mediante el objeto de tipo *GeneradorModelosSimulacion*, creado en la carga de datos, manda la ejecución de la generación de modelos que pasa el control a ese objeto. Entonces comienza la generación. Lo primero que hace es realizar la fusión de los modelos mediante el método *fusionModelos*. Una vez que se ha producido la fusión comienza la generación de los modelos que crea una lista con los modelos generados. Teniendo la lista, llama al método *repasadoFinal* para buscar errores en la lista y los modifica. Con la lista repasada se vuelve a control que mandará al objeto *ventana*, de tipo *Interfaz*, que muestre los resultados.

- Caso de uso **CalcularResiduo**:
  Este caso de uso comienza con una petición del usuario para ejecutar el cálculo de residuos, dicha petición se pasará al objeto *control*. *Control* mediante el objeto creado en la carga, de tipo *Diagnosis*, ejecuta primero la generación de la función para Matlab que pasa el control a ese objeto y el mismo crea la función de Matlab. La siguiente llamada de control será al mismo objeto que ha generado la función para que la ejecute a través de Matlab. Una vez realizada la ejecución

esta vez *control* mandará al objeto *ventana*, de tipo *Interfaz*, que muestre los resultados.

5.6.2. Diagramas de secuencia de la aplicación

En este apartado se van a mostrar los diagramas de secuencia del sistema. Se mostrarán en rojo las operaciones que se han modificado o nuevas. Para el apartado de carga de datos vamos a mostrar las secuencias del caso de uso **CargarModeloCM** y **CargarModeloGMS** suficientes para ver las modificaciones realizadas. Los casos de uso **CargarModeloES** y **CargarModeloDX** funcionan igual que estos mencionados anteriormente por lo que se han elegido mostrar los anteriores, en los que se ven todas las operaciones que habría en estos últimos, evitando así repeticiones de información.

En cuanto a la ejecución de funciones, se va a mostrar el caso de uso **CovertirModelo** y **GenerarModeloSim** de forma completa. De los casos de uso **GenerarExperimento** y **CalcularResiduo** se va a mostrar parte del caso de uso, la que ha sido modificada y que es común en ambos, la que genera la función de simulación para Matlab.

Como podemos ver en la figura 5.8, es la parte que hemos modificado del caso de uso calcular residuos y que es equivalente a **GenerarExperimento**, se ha modificado la función *write(equation)*. Lo que hacemos en este caso es para cada ecuación comprobamos el tipo de ecuación, con o sin condiciones. Si tiene condición asociada se escribe la ecuación sino, si tiene condición, el sistema buscará toda la información relativa a la ecuación y la escribirá de la forma correcta.

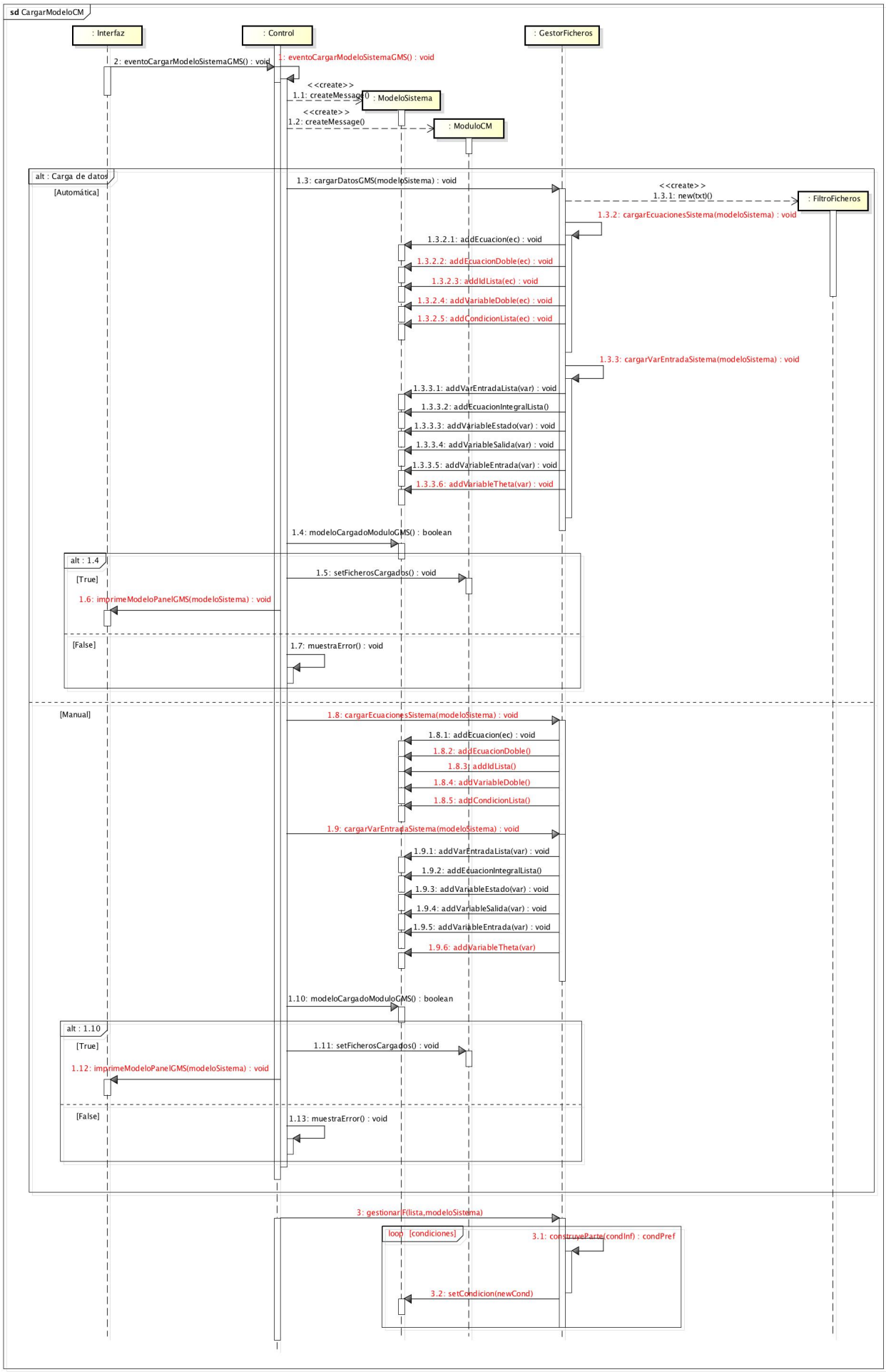


Figura 5.3 : Diagrama de secuencia CargarModeloCM

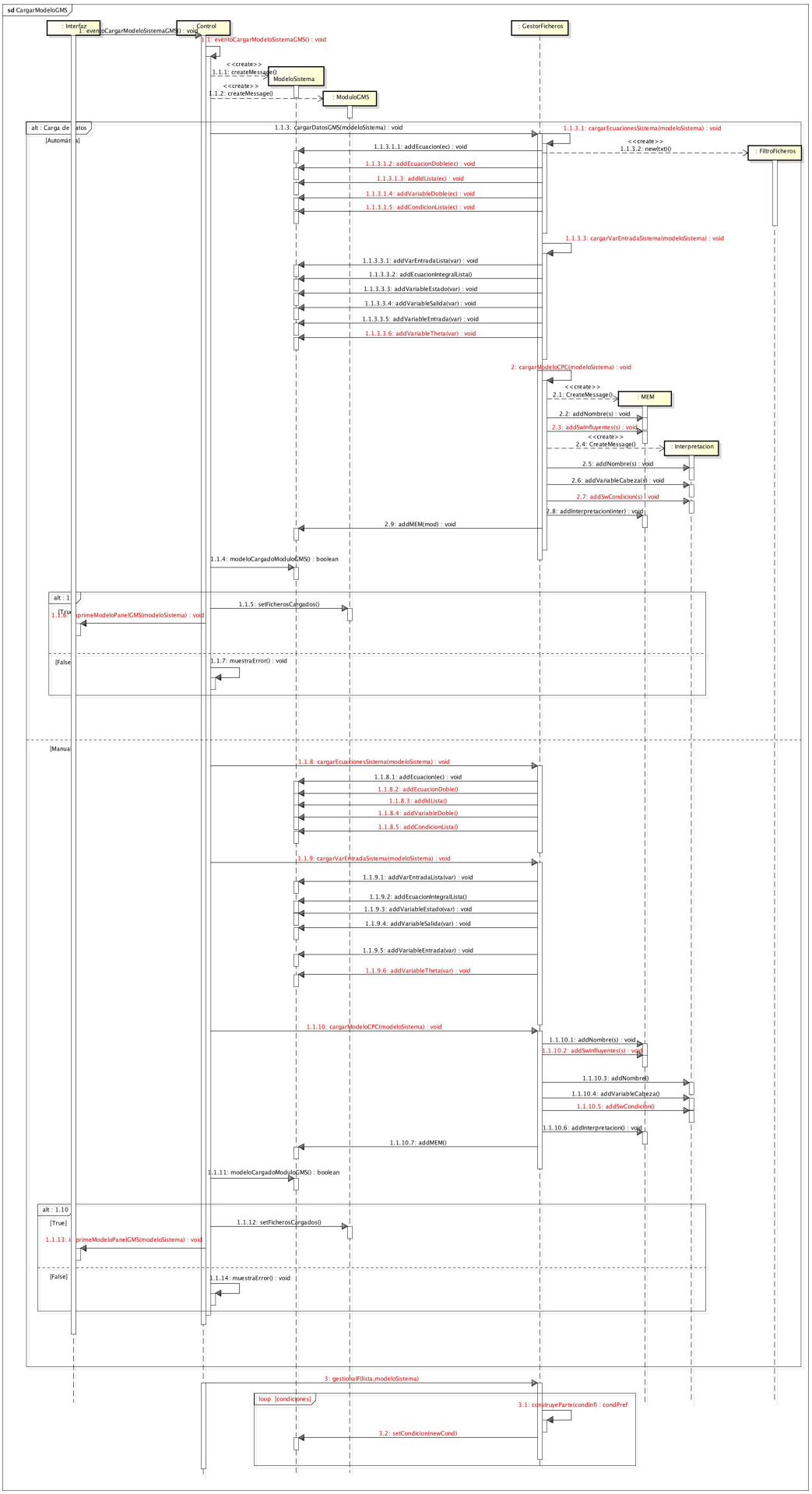


Diagrama 5.4 : Diagrama de secuencia CargarModeloGMS

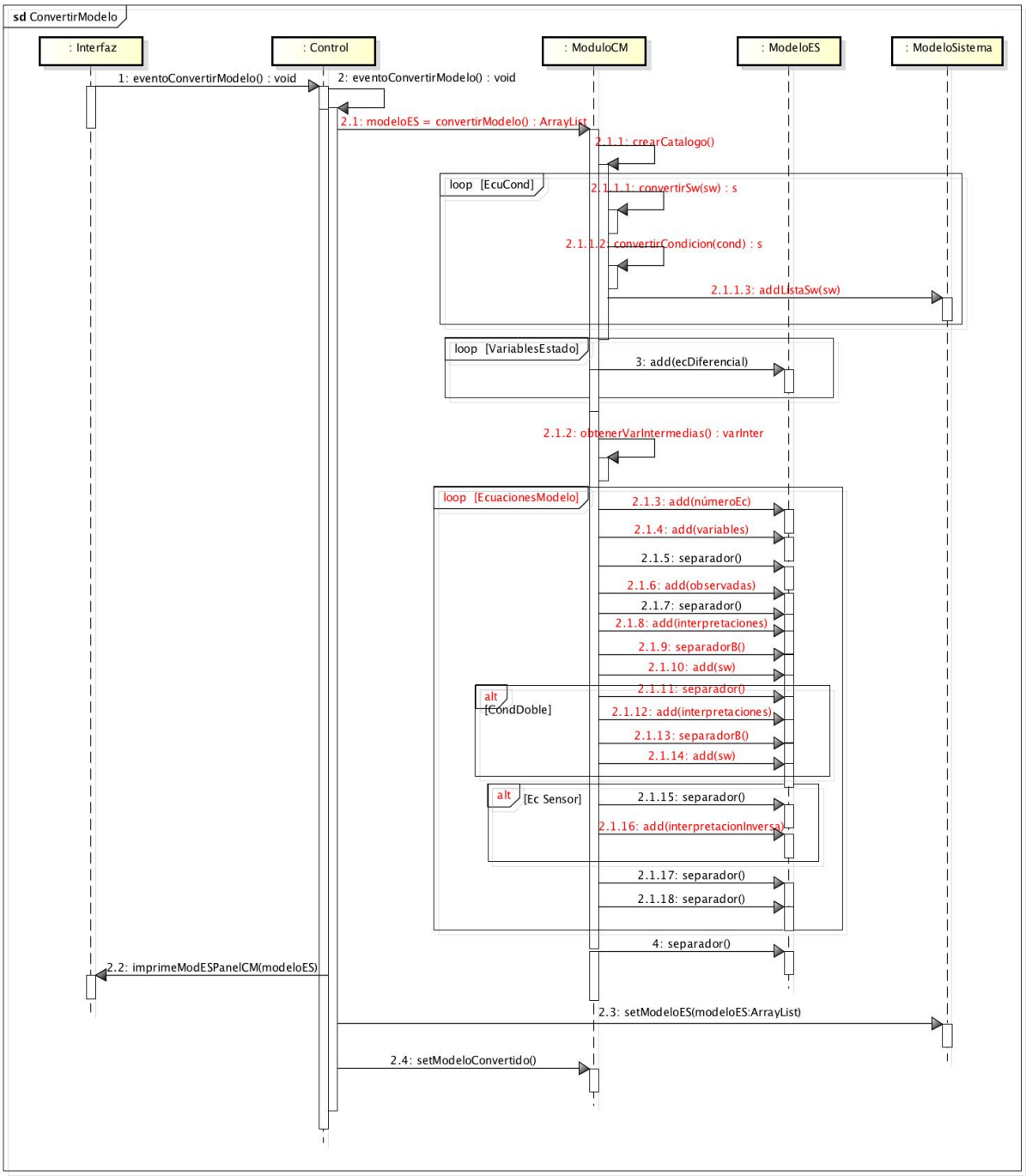


Figura 5.5 : Diagrama de secuencia ConvertirModelo

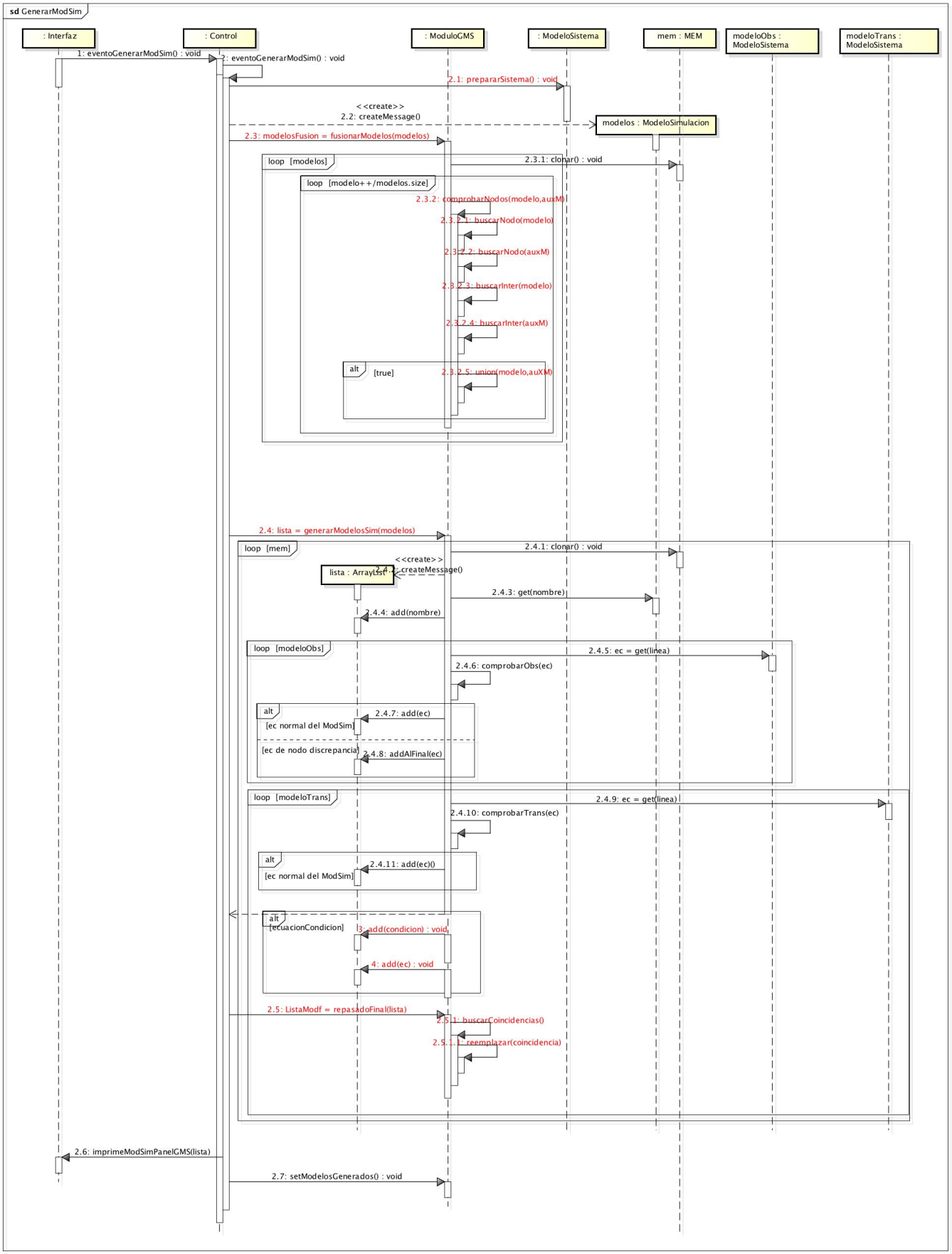


Figura 5.6 : Diagrama de secuencia GenerarModeloSim

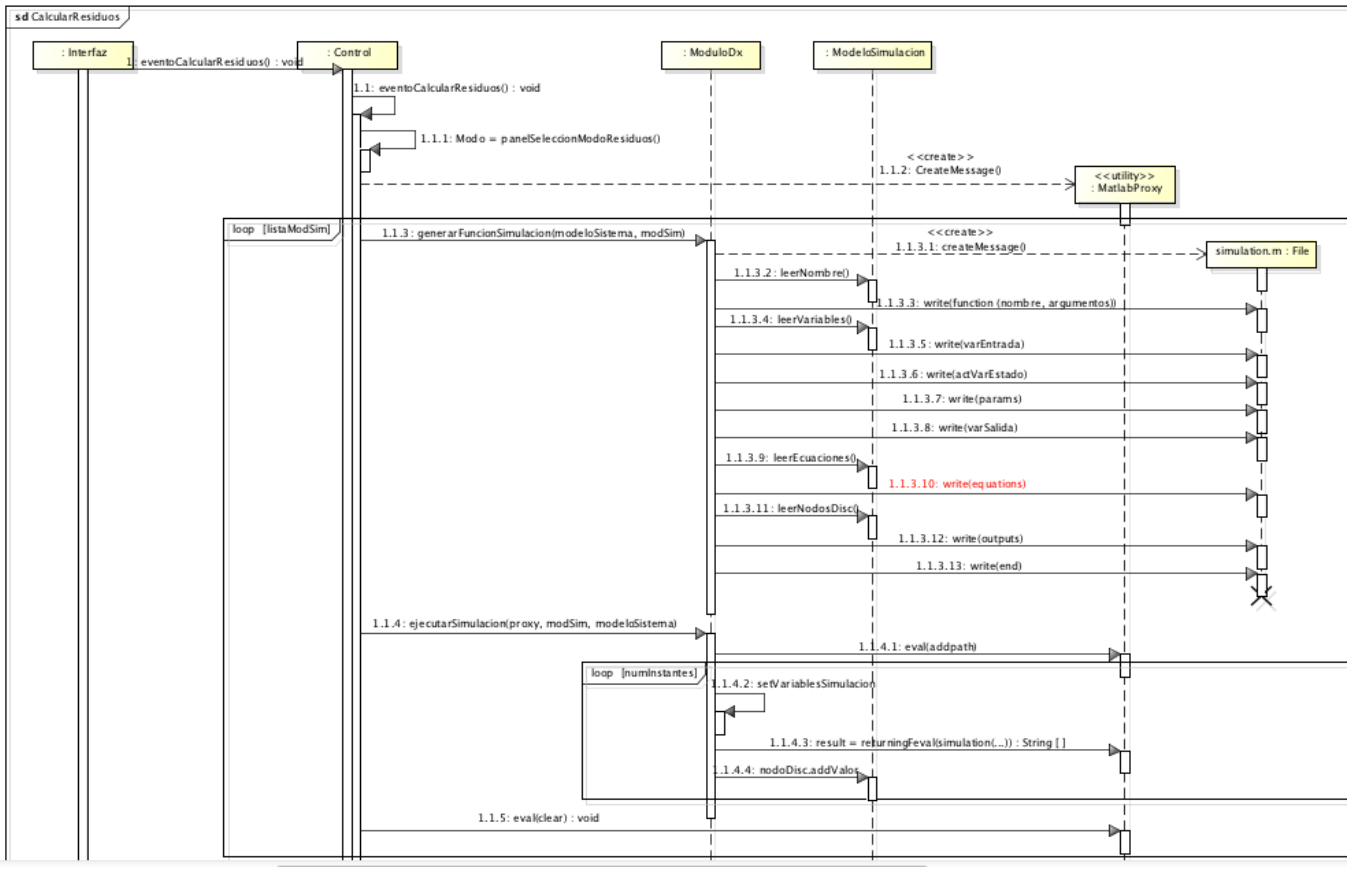


Figura 5.7 : Diagrama de secuencia GeneracionFuncionMatlab

6. MODELO DE IMPLEMENTACIÓN

6.1. INTRODUCCIÓN

6.1.1. Propósito

El propósito del modelo de implementación es describir los procesos y las decisiones que se han llevado a cabo en la implementación del proyecto.
Los usuarios potenciales de este apartado de documentación serán, principalmente, los programadores (además serán estos mismos los que lo generen).

6.1.2. Alcance

Este apartado proporcionará información sobre el proceso de implementación del proyecto "DxPCs 2.0 - Ampliación de la plataforma para diagnosis basada en modelos".

6.1.3. Definiciones, Acrónimos, Abreviaturas

Véase el glosario general del proyecto.

6.1.4. Perspectiva General

Este capítulo constará de las siguientes partes:

- Descripción de las herramientas de trabajo.
- Aportaciones: Estructuras y algoritmos nuevos: en este apartado se describirán los cambios realizados en la aplicación de CPCs y los algoritmos implementados.

6.2. Descripción de las herramientas de trabajo

6.2.1. Lenguaje de programación Java

El lenguaje de programación elegido para llevar a cabo la implementación será Java, en su última versión (a fecha de la realización de este documento, la versión más reciente es la 8). El principal motivo de esta decisión es que la aplicación de la que partimos ya había sido desarrollada en este lenguaje y, puesto que debíamos mantener la compatibilidad con ésta, haber llevado a cabo el desarrollo en otro lenguaje habría sido tremendamente costoso.

6.2.2. Entorno de desarrollo Netbeans

El entorno de desarrollo que se va a utilizar será Netbeans. Se trata de un software de código abierto con el que el equipo de desarrollo ya había trabajado con anterioridad.

6.2.3. Sistemas operativos Windows y OS X

Se va a realizar la mayor parte del trabajo, la realización del documento y programación, en el S.O. OS X que es el portátil con el que cuenta el equipo. Debido a que la licencia que poseemos de Matlab es para el S.O. Windows las pruebas de la herramienta se realizarán en este otro sistema.

6.3. Aportaciones a la herramienta

6.3.1. Cambios en el almacenamiento de datos

El cambio que vamos a introducir en la herramienta, será una ampliación para que la herramienta pueda trabajar también con modelos discretos de modelos híbridos. Ésto conlleva modificar aquellas estructuras en la cuales se guardan los datos de los mismos. Por ello van a realizarse cambios en esas estructuras para poder almacenar la nueva información que conlleva trabajar con estos sistemas.

- *ModeloSistema*
    - *listaCondicion:*
      Será un arrayList que almacena el conjunto de elementos de tipo condición.
    - *listaEcuacionesDoble:*
      Será un arrayList con ecuaciones. Estas ecuaciones serán aquellas que tienen un elemento condición asociado.
    - *parametros*:
      Será un *arrayList* con los nombres de los parámetros. En ocasiones no necesitamos cargar toda la información que contiene el fichero de parámetros sino solamente sus nombres. Por ello con este array se facilitará el trabajo al no tener que cargar otro fichero completo sino solo los datos del conjunto Theta del fichero variables. Será cargado por *GestorFicheros* en la carga de datos.
- *Condicion*
    - *id:* Será un String que guardará el identificador, de la forma *swx*, de la expresión de la condición. Será cargado por GestorFicheros en la carga de datos.
    - *expresion:* Será un String que guardará la expresión de la condición de la ecuación asociada. Será cargado por GestorFicheros en la carga de datos.
    - *expref:* Será un String que guardará la expresión de la condición en forma prefija. Será cargado por GestorFicheros en la carga de datos tras realizar la conversión.

- *negacion:* Será un String que guardará la expresión de la condición del else, si la hubiera. Será cargado por GestorFicheros en la carga de datos.

- *Interpretación*

  - *swCondicion*:
    Guardará el identificador de la condición que afecta a la interpretación. Será cargado por GestorFicheros en la carga de datos.

- *ModeloSimulacion*

  - *listaIF*:
    Será un *arrayList* con la lista de las expresiones de las condiciones de un modelo de simulación. Será cargado por GestorFicheros en la carga de datos.

  - *listaEcuacionesIF*:
    Será un *arrayList* con la lista de las expresiones de las ecuaciones asociadas a una condición. Su condición asociada tendrá la misma posición en listaIF. Será cargado por GestorFicheros en la carga de datos.

- *MEM*

  - *swInfluyentes:*
    Sera un *string* que guardará los identificadores de las condiciones, en la forma *swx*, que afectan a ese *MEM*. Será cargado por *GestorFicheros* en la carga de datos.

  - *interDoble*:
    Será un *arrayList* que guardará una lista de interpretaciones. Cuando se fusionan dos o más modelos se usará uno de base, las ecuaciones añadidas al mismo de forma suplementaria al modelo base a la hora de la fusión serán almacenadas en este array. Será cargado en la operación de fusión de modelos de simulación.

6.3.2. Algoritmos nuevos

6.3.2.1. Conversión de Modelos

En el apartado de conversión de modelos se han añadido principalmente dos algoritmos además de la modificación del algoritmo que ya había. Estos algoritmos son los equivalentes a las funciones *crearCatalogo* y *obtenerVarIntermedias*. El primero crear catálogo crea una lista de los sw que hay en nuestro sistema comprobando si hay sw repetidos u opuestos para que la función de conversión trabaje posteriormente más fácilmente. El otro *obtenerVarIntermedias* se crea

debido a un problema encontrado en el código anterior; en la versión anterior se detectaban las variables intermedias al analizar cada ecuación y si éstas no habían sido calculadas previamente ocasionaban un error si las ecuaciones no estaban escritas en el orden adecuado para simular. Ahora, con este nuevo tratamiento, se evita tener que dar las ecuaciones en el orden exacto de simulación. Ésta función crea el conjunto de variables intermedias antes de comenzar la conversión consiguiendo así que el orden de las ecuaciones sea indiferente.

```
ALGORITMO obtenerVarIntermedias()
   i:=0;
   MIENTRAS i < modelo.listaEcuaciones.tamaño() HACER
       cabeza := listaEcuaciones.getCabeza(i);
       SI NOT modelo.variablesSalida.contains(cabeza) ENTONCES
          SI NOT variablesIntermedias.contains(cabeza) ENTONCES
               variablesIntermedias.add(cabeza);
          FIN SI
       FIN SI
   i:=i+1;
   FIN MIENTRAS
FIN

ALGORITMO crearcatalogo()
   i:=0;
   cont := 1;
   MIENTRAS i < modelo.listaCondicion.tamaño() HACER
       igual := false;
       z:=0;
       condicion := modelo.listaCondicion.condicion.expref.get(i);
       MIENTRAS z < catalogo.tamaño() && igual == true HACER
          itemcatalogo := catalogo.get(z);
          SI itemcatalogo == condicion ||
          itemcatalogo == convertirCondicion(condicion) ENTONCES
               igual := true;
          FIN SI
          z:= z+1;
       FIN MIENTRAS
       SI igual == false ENTONCES
          modelo.condicion.id.add("sw"+ cont);
          cont := cont + 1;
       FIN SI
   i := i+1;
   FIN MIENTRAS
FIN
```

Además de las dos funciones principales que acabamos de comentar hay otras funciones de apoyo más simples como lo son *convertirCondicion* y *convertirSw* que sirven para obtener las formas negadas de las condiciones y de los sw.

6.3.2.2. Conversión de forma infija a prefija

Lo primero, tenemos el algoritmo *gestionarIF* que es la base para el cambio. Este algoritmo leerá línea a línea el sistema y cuando detecte una condición la transformará llamando a *construyeParte*.

```
ALGORITMO gestionarIF(listaSistema , modeloSistema)
    MIENTRAS i<listaSistema.tamaño() HACER
           elemento := listaSistema.get(i);
           SI elemento.empiezapor("if") ENTONCES
                  elemento := contruyeParte(elemento,modelo);
           FIN SI
    FIN MIENTRAS
FIN
```

El algoritmo *construyeParte* es un algoritmo recursivo basado en un case con varias opciones. Irá leyendo elemento a elemento la condición, si encuentra paréntesis llamará a la misma función de forma recursiva del interior del paréntesis, sino se identifica la posible función, número, si es un espacio y se guarda la información en una pila para el posterior montaje de la condición en forma prefija.

```
ALGORITMO construyeParte(elemento,modelo)
    z := 1;
    MIENTRAS i<elemento.tamaño() HACER
       aux := elemento.substring(z);
       SEGUN aux HACER
           CASO "(":
              construyeParte(elementoEntreParentesis, modelo);
           CASO funcion/numero/espacio:
              identificarElemento();
               realizarTransformacion();
              guardarEnPila();
       FIN SEGUN
    FIN MIENTRAS
    generarNuevaCondicion(pila);
FIN
```

También tenemos una función de apoyo a la que hemos denominada *isNumber* que comprueba si el *string* pasado es un número.

6.3.2.3. Generación de Modelos de Simulación

Una vez que tenemos los modelos fusionados que veremos en el apartado siguiente, 6.3.2.4. Con los modelos fusionados se llama a la antigua función *generarModelosSimulacion* que como veremos después, la modificación, solo añade las condiciones además de aquellas ecuaciones complementarias al antiguo funcionamiento que generaba los modelos de los sistemas continuos. Por ello aun nos faltaría ver si aquellas variables de estado, de esas condiciones y ecuaciones complementarias, están escritas correctamente o hay que modificarlas por su valor "_ant". De esto se encarga la función *repasadoFinal*, que pasa por cada uno de los elementos de la lista generada por *generarModelosSimulacion* buscando ese posible error.

```
ALGORITMO repasadoFinal(lista)
   i:=0;
   MIENTRAS i < lista.tamaño() HACER
       SI esCondicion() ENTONCES
              PARA CADA variableDeLaCondicion HACER
                 SI variable.noPerteneceACoincidencias() ENTONCES
                     reeemplazar("_ant");
                 FIN SI
              FIN PARA
       FIN SI
       SI esEcuacion() ENTONCES
              coincidencias.add(cabezaEcuacion);
              PARA CADA variableDeLaEcuacion HACER
                 SI variable.noPerteneceACoincidencias() ENTONCES
                     reeemplazar("_ant");
                 FIN SI
              FIN PARA
       FIN SI
       i := i+1;
   FIN MIENTRAS
FIN
```

6.3.2.4. Fusión de Modelos de Simulación

El algoritmo de fusión se encargará de unir aquellos Modelos de Simulación que son comunes haciendo que posteriormente los resultados de la diagnosis de vean de una forma más completa y simple. El algoritmo se llama *fusionarModelos*. Éste se vale de otros algoritmos para su funcionamiento que son: *buscarNodo*, *buscarInter* y *comprobarNodos*.

El algoritmo *buscarNodo* lo que hace simplemente es buscar el nodo del *MEM* que se le ha pasado. Su funcionamiento consiste en encontrar aquella variable que está dos veces definida tanto por el valor del sensor como por el de su ecuación. Buscará repeticiones de variables valiéndose de una pila.

```
ALGORITMO buscarNodo(mem)
       i := 0;
       MIENTRAS i < interpretaciones.tamaño() HACER
              SI inter.cabeza.terminapor(mes) ENTONCES
                     SI inter.variablecola.estaEnPila() ENTONCES
                            nodo := inter.variablecola;
                     SINO
                            inter.variablecola.añadirPila();
                     FIN SI
              SINO
                     SI inter.variablecabeza.estaEnPila() ENTONCES
                            nodo := inter.variablecabeza;
                     SINO
                            inter.variablecabeza.añadirPila();
                     FIN SI
              FIN SI
       i := i + 1;
       FIN MIENTRAS
FIN
```

Tras encontrar el nodo lo siguiente será extraer las interpretaciones en la cuales el nodo está en nodo, que serán discrepantes. Para ellos usamos *buscarInter* que extraerá esas interpretaciones por medio del nodo que le da la ecuación anterior.

```
ALGORITMO buscarInter(mem)
       nodo = buscarNodo(mem);
       MIENTRAS i< mem.interpretaciones.tamaño() HACER
              SI coincidencia(nodo) ENTONCES
                     lista.añadir(inter);
              FIN SI
       FIN MIENTRAS
FIN
```

El algoritmo *comprobarNodos* como su nombre indica comprueba si dos *MEM* tienen el mismo nodo y tras esto si con el mismo nodo las interpretaciones que extrajo *buscarInter* sean las mismas.

```
ALGORITMO comprobarNodos(mem1,mem2)
       iguales:= false;
       nodo1 := buscarNodo(mem1);
       nodo2 := buscarNodo(mem2);
       SI comprobarNodosIguales() ENTONCES
              listainter1 := buscarInter(mem1);
              listainter2 := buscarInter(mem2);
              SI comprobarListasIguales() ENTONCES
                     iguales := true;
              FIN SI
       FIN SI
FIN
```

Con estas funciones de apoyo ya podemos realizar la fusión. Primero se comprobará que se trata de un modelo equivalente y tras ello añadirá al modelo base las ecuaciones que están en los otros modelos equivalentes pero no en el base.

```
ALGORITMO fusionarModelos(modelosSimulacion)
       i:=0;
       MIENTRAS (i < modelosSimulacion.tamaño()) HACER
          m := modelosSimulacion.get(i);
          z := i + 1;
          MIENTRAS (z < modelosSimulacion.tamaño()) HACER
              maux := modelosSimulacion.get(z);
              SI comprobarMismaCadena() && comprobarSw() ENTONCES
                     SI comprobarNodos(m,maux) ENTONCES
                            buscarEcuacionesExtra();
                            añadirEcuacionesExtraModeloBase();
                     FIN SI
              FIN SI
              Z := z+1;
          FIN MIENTRAS
       FIN MIENTRAS
FIN
```

El funcionamiento se basa en la comparación; para cada modelo como vemos lo compara con el resto de modelos siguientes en la lista de modelos y una vez que sabe que se trata de modelos comunes se encargará de buscar las ecuaciones distintas al modelo base y añadírselas al mismo creando así el modelo fusionado que sustituirá a aquellos modelos de los cuales parte. Para buscar las interpretaciones distintas lo que hará será comparar cada interpretación del modelo auxiliar con todas las del modelo base, añadiéndolas en el caso no estar ya presentes.

Finalmente podemos ver en un ejemplo como sería una fusión en la cual 4 MEM pasan a estar representados por uno solo. Como se puede ver en la figura 6.1 la principal diferencia entre los cuatro MEMs del PC es que q_sa se calcula de dos formas distintas, según la condición sw2 esté activada o negada.

La otra mínima diferencia estriba en que el residuo se calcule asociado a la diferencia entre h2 estimada y medida o h2 estimada por dos vías. En ambos casos la información de diagnosis es la misma y se puede fusionar en el modelo que aparece en la nueva figura 6.2.

```
Modelo 1(Cadena 1): ; {sw2}
q_f2 := ec10_1(h_2)
d_h_2 := ec15_1(q_s2, q_f2, q_sa)
h_1 := ec17_2(h1_mes)
h2_mes := ec18_1(h_2)
q_sa := ec22_1(h_1, h_2); {sw2}
h_2 := ec2_1(d_h_2)
q_s2 := ec6_1(h_2)

Modelo 2(Cadena 1): ; {not sw2}
q_f2 := ec10_1(h_2)
d_h_2 := ec15_1(q_s2, q_f2, q_sa)
h_1 := ec17_2(h1_mes)
h2_mes := ec18_1(h_2)
q_sa := ec22_2(h_1, h_2); {not sw2}
h_2 := ec2_1(d_h_2)
q_s2 := ec6_1(h_2)

Modelo 3(Cadena 1): ; {sw2}
q_f2 := ec10_1(h_2)
d_h_2 := ec15_1(q_s2, q_f2, q_sa)
h_1 := ec17_2(h1_mes)
h_2 := ec18_2(h2_mes)
q_sa := ec22_1(h_1, h_2); {sw2}
h_2 := ec2_1(d_h_2)
q_s2 := ec6_1(h_2)

Modelo 4(Cadena 1): ; {not sw2}
q_f2 := ec10_1(h_2)
d_h_2 := ec15_1(q_s2, q_f2, q_sa)
h_1 := ec17_2(h1_mes)
h_2 := ec18_2(h2_mes)
q_sa := ec22_2(h_1, h_2); {not sw2}
```

Figura 6.1 MEMs de un PC

```
Modelo 1(Cadena 1): ; {sw2} {not sw2}
q_f2 := ec10_1(h_2)
d_h_2 := ec15_1(q_s2, q_f2, q_sa)
h_1 := ec17_2(h1_mes)
h2_mes := ec18_1(h_2)
q_sa := ec22_1(h_1, h_2); {sw2}
h_2 := ec2_1(d_h_2)
q_s2 := ec6_1(h_2)
q_sa := ec22_2(h_1, h_2); {not sw2}
```

Figura 6.2 Fusión de MEMs

6.3.3. Algoritmos modificados

6.3.3.1. Lectura y visualización de datos

Al introducir la posibilidad de que una ecuación lleve asociada una condición, además de los nuevos MEM, como hemos visto, se han añadido nuevos elementos a las clases antiguas para guardar en ellas esta nueva información. Por esto se han cambiado ciertas funciones que leían la información de los ficheros y las que lo mostraban en pantalla.

Lo primero fue cambiar la lectura de las ecuaciones. Para ello se modificó la antigua función para que ahora reconozca cuándo existe una condición asociada a una condición y la guarde correctamente.

Se ha añadido la lectura de parámetros Theta del fichero de variables de forma equivalente a los otros conjuntos. Antes no se realizaba esta carga y en ocasiones

necesitamos solamente los nombres de las variables, sin sus valores. Así evitamos toda la carga del fichero parámetros.
La lectura de los MEM es la última en la cual se han hecho cambios ya que ahora pueden llevar nueva información. Lo principal ha sido cambiar el *parser* creando uno que funcionase para todos los posibles MEM.
La visualización de la información guardada, en los distintos paneles de la interfaz será la operación inversa, extrayendo la nueva información tal y como la hemos introducido anteriormente.

6.3.3.2. Conversor de Modelos

El código que se ha modificado en este apartado es el referente a la antigua función de conversión de *Modelos*. Lo principal ha sido añadir a la función la búsqueda de las variables no observadas y observadas en las condiciones asociadas a ecuaciones si se dieran el caso de que existieran. Además de añadir la nueva parte del *Modelo ES*, la asociada a representar cada condición, denominada con un Sw, dentro del *Modelo*.

6.3.3.3. Generador de experimentos

El código modificado de este apartado es el relativo a la función que se genera para ejecutar en Matlab. Al igual que en la impresión de datos en la interfaz se ha modificado para que a la hora de escribir una ecuación compruebe si tiene datos asociados como una condición y en ese caso los escriba.

6.3.3.4. Generador de Modelos de Simulación

El código modificado en este apartado ha sido el relativo a la función que generaba los modelos de simulación. A la hora de crear los *Modelos* según vamos añadiendo las ecuaciones se va comprobando si cada una de ellas tiene más datos asociados como condiciones y se van introduciendo tal cual. Las modificaciones que necesitan estas expresiones de condición y las expresiones de ecuaciones del else para generar el Modelo de Simulación, las realiza *repasadoFina*l.

6.3.3.5. Diagnosis

Al igual que en Generador de Experimentos, el código modificado de este apartado es el relativo a la función que se genera para ejecutar en Matlab. Sin embargo en este caso vamos a crear no solo una función, sino una para cada Modelo de Simulación, es decir cada Modelo tendrá su propio fichero independiente. Según vamos introduciendo cada ecuación se va comprobando si cada una de ellas tiene más datos asociados como condiciones y se van introduciendo para generar cada función de simulación de Matlab.

Cabe destacar los cambios realizados en la función que se encarga de decidir si un residuo se está desviando lo suficiente como para dar lugar a una decisión mediante el test estadístico denominado z-test. En la versión anterior había un error que hacía que no funcionase el mismo. Se busco el error que se encontraba en el cálculo del valor de la desviación estándar. Además se cambió la forma de introducir los valores del Z-test, pasando de introducir instantes de ejecución a

unidades de tiempo, es decir, si se estuvieran realizando los cálculos en segundos, introducir directamente el tiempo en segundos.

7. PLAN DE PRUEBAS

7.1. INTRODUCCIÓN

7.1.1. Propósito

El propósito de este apartado será recoger información sobre las pruebas realizadas para comprobar aspectos del producto finalizado tales como eficiencia, fiabilidad y robustez. Dichas pruebas se llevarán a cabo tanto durante como después de haber concluido la fase de implementación.

7.1.2. Alcance

Este capítulo proporcionará información sobre las pruebas realizadas en la aplicación sobre la que se basa el proyecto "DxPCs 2.0 - Ampliación de la plataforma para diagnosis basada en modelos".

7.1.3. Definiciones, Acrónimos, Abreviaturas

Véase el glosario general del proyecto.

7.1.4. Perspectiva General

En este documento encontramos información sobre las pruebas realizadas divididas en dos apartados:

- Pruebas sobre los sistemas continuos
- Pruebas sobre los sistemas híbridos

A lo largo del desarrollo del proyecto se encontraron algunos fallos sobre el código antiguo. Estos fallos se subsanaron, y por ello ahora realizaremos pruebas de forma más extensa sobre varios sistemas.

7.2. Pruebas realizadas

Como ya hemos dicho se van a realizar pruebas sobre dos tipos de sistemas, continuos e híbridos. Dentro cada uno de estos tipos se realizarán pruebas con distintos casos, varios sistemas continuos y varios híbridos. Cabe destacar en la pruebas:

- En el caso de la generación de experimentos se han simulado varios modos de funcionamiento. Se han cambiado los tiempos de simulación, se han probado los sistemas con diferentes entradas y valores de parámetros e introducido distintos tipos de fallo.

- En la Diagnosis de han probado las distintas formas de calcular la detección, con cada una de los modos de simulación probados en la generación de experimentos. Se han probado para el cálculo de candidatos los 3 test de los

que dispone la herramienta cada uno de ellos probados con distintos umbrales de fallo.

7.2.1. Pruebas de la herramienta DxPCs 1.0

Antes de comenzar a trabajar se van a realizar pruebas sobre varios sistemas previos, los sistemas continuos, en cada una de los 4 apartados de la herramienta para comprobar que todo funciona de forma correcta.

CP-01	Conversor de Modelos
Objetivo	Cargar un sistema continuo y realizar la conversión.
Procedimiento	Realizar la carga de uno de los sistemas continuos tanto de forma manual como automática y después proceder con la conversión.
Salida Esperada	Se deberá mostrar el sistema cargado. Dicha información deberá coincidir con la de los ficheros de entrada. Se deberá mostrar el resultado del sistema convertido.
Resultado	El resultado obtenido es correcto cuando el orden de las ecuaciones es tal que primero se colocan en el fichero aquellas que poseen las variables intermedias. Se creará una función que genere ese conjunto antes de la conversión para evitar ese error debido al orden. Véase el apartado de implementación.

CP-02	Generación de Experimentos
Objetivo	Cargar un sistema continuo y realizar la ejecución.
Procedimiento	Realizar la carga de uno de los sistemas continuos tanto de forma manual como automática y después proceder con la ejecución.
Salida Esperada	Se deberá mostrar el sistema cargado. Dicha información deberá coincidir con la de los ficheros de entrada. Se deberá mostrar el resultado de la ejecución de Matlab.
Resultado	El resultado obtenido es correcto.

CP-03	Generador de Modelos de Simulación
Objetivo	Cargar un sistema continuo y realizar la generación.
Procedimiento	Realizar la carga de uno de los sistemas continuos tanto de forma manual como automática y después proceder con la generación.
Salida Esperada	Se deberá mostrar el sistema cargado. Se procederá al cálculo de los Modelos de Simulación y tras ello se visualizaran cada uno de los modelos obtenidos.
Resultado	El resultado obtenido es correcto.

CP-04	Diagnosis – Cálculo de Residuos
Objetivo	Cargar un sistema continuo y realizar la ejecución.
Procedimiento	Realizar la carga de uno de los sistemas continuos tanto de forma manual como automática y después proceder con la ejecución.
Salida Esperada	Se deberá mostrar el sistema cargado. Se deberá de visualizar el resultado de la ejecución así como mostrar en la interfaz los resultados del experimento anterior y los cálculos de los residuos actuales.
Resultado	El resultado obtenido es correcto.

CP-05	Diagnosis – Cálculo de candidatos
Objetivo	Realizar el cálculo de los candidatos a fallo.
Procedimiento	Una vez realizado el cálculo de residuos se procederá con el cálculo de los candidatos.
Salida Esperada	Se deberá mostrar de forma correcta el instante en el cuál se produce el fallo introducido y visualizar por pantalla los conjuntos de candidatos que se van generando según se consiguen las activaciones.
Resultado	El resultado obtenido es incorrecto. Existe un fallo en la ejecución del z-test. Véase el apartado de implementación.

7.2.2. Pruebas de la herramienta DxPCs 2.0

7.2.2.1. Pruebas con sistemas continuos

A la hora de modificar la herramienta antigua queríamos añadir la posibilidad de añadir sistemas híbridos. Sin embargo los cambios debían hacer que la herramienta siguiera funcionando con los sistemas continuos por ello realizamos pruebas sobre distintos sistemas, con sus correspondientes modelos, comprobando que los cambios hechos respetan el funcionamiento de ambos tipos de comportamiento, tanto continuo como híbrido. En este apartado cada una de las imágenes que se usa para la ilustración corresponde con un sistema de 3 tanques continuo.

CP-01	Carga de sistemas continuos (Manual – Automático)
Objetivo	Cargar los sistemas continuos y variables de ambas formas.
Procedimiento	Seleccionar la opción “Load Model” y cargar los sistemas continuos de forma manual y automática.
Salida Esperada	Se deberá mostrar el sistema cargado. Dicha información deberá coincidir con la de los ficheros de entrada.
Resultado	El resultado obtenido es correcto.

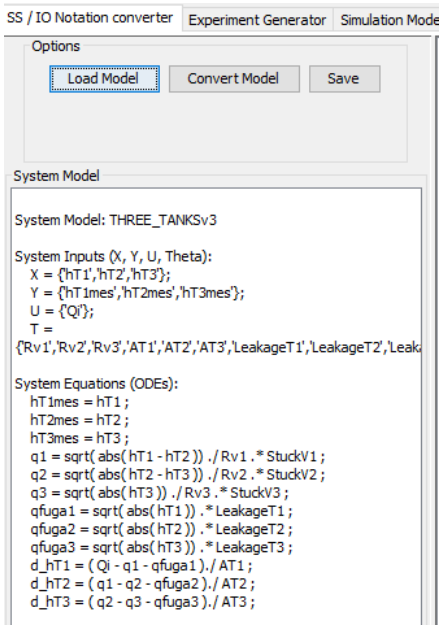


Figura 7.1 Carga de un sistema continuo

CP-02	Carga de los MEMs de sistemas continuos.
Objetivo	Cargar los MEMs generados por la herramienta de CPCs en el apartado “Simulation Model Generator” de sistemas continuos.
Procedimiento	Seleccionar la opción “Load Model” dentro de “Simulation Model Generator” y cargar uno de los sistemas continuos.
Salida Esperada	Se deberá mostrar los MEM de forma correcta. Dicha información deberá coincidir con la de los ficheros de entrada.
Resultado	El resultado obtenido es correcto.

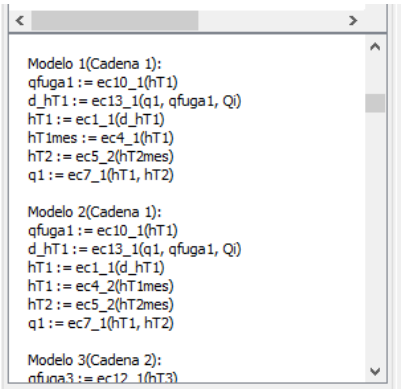


Figura 7.2 Carga de los MEM

CP-03	Carga de los Modelos de Simulación de sistemas continuos
Objetivo	Cargar los modelos de simulación que ha generado la herramienta de sistemas continuos.
Procedimiento	Seleccionar la opción “Load Model” dentro de “Diagnosis” y cargar uno de los sistemas continuos.
Salida Esperada	Se deberá mostrar los modelos de simulación de forma correcta. Dicha información deberá coincidir con la de los ficheros de entrada.
Resultado	El resultado obtenido es correcto.

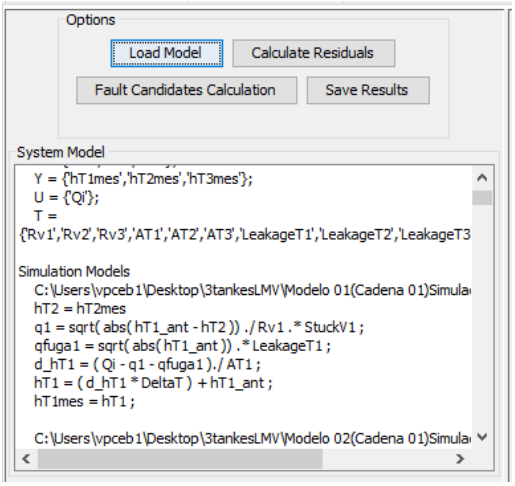


Figura 7.3 Carga Modelos Simulación

CP-04	Generación de un Modelo E/S de sistemas continuos
Objetivo	Generar de forma correcta el Modelo E/S de sistemas continuos.
Procedimiento	Con el sistema ya cargado, pulsar sobre la opción “Convert Model” en “SS/IO Notation Converter”.
Salida Esperada	Se deberá mostrar el Modelo E/S de forma correcta.
Resultado	El resultado obtenido es correcto.

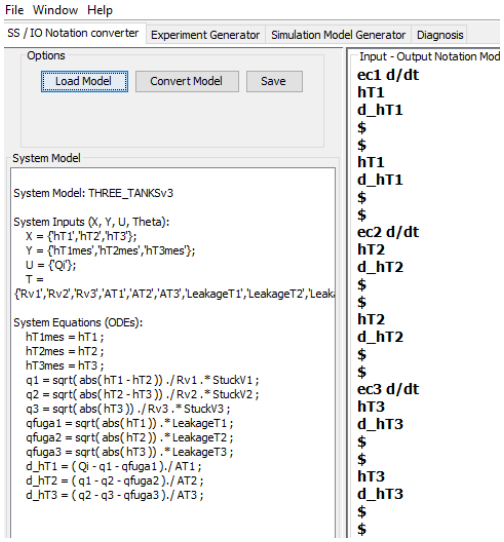


Figura 7.4 Modelo E/S

CP-05	Generación de experimentos de sistemas continuos
Objetivo	Generar un experimento de forma correcta de sistemas continuos.
Procedimiento	Con el sistema ya cargado, pulsar sobre la opción “Generate Experiment”.
Salida Esperada	Se deberá mostrar el resultado de la ejecución de Matlab, es decir, el cálculo de las los valores de cada variable en cada instante.
Resultado	El resultado obtenido es correcto.

CP-06	Generación de los Modelos de Simulación de sistemas continuos
Objetivo	Generar los Modelos de Simulación para sistemas continuos de forma correcta.
Procedimiento	Con el sistema ya cargado, pulsar sobre la opción “Generate(Simulation)” .
Salida Esperada	Se deberán mostrar los modelos generados.
Resultado	El resultado obtenido es correcto.

CP-07	Fusión en la generación de Modelos de Simulación de sistemas continuos.
Objetivo	Que se produzca la fusión de modelos en la generación.
Procedimiento	La fusión se realiza de forma automática en la generación.
Salida Esperada	Se deberán mostrar los modelos generados fusionados.
Resultado	El resultado obtenido es correcto.

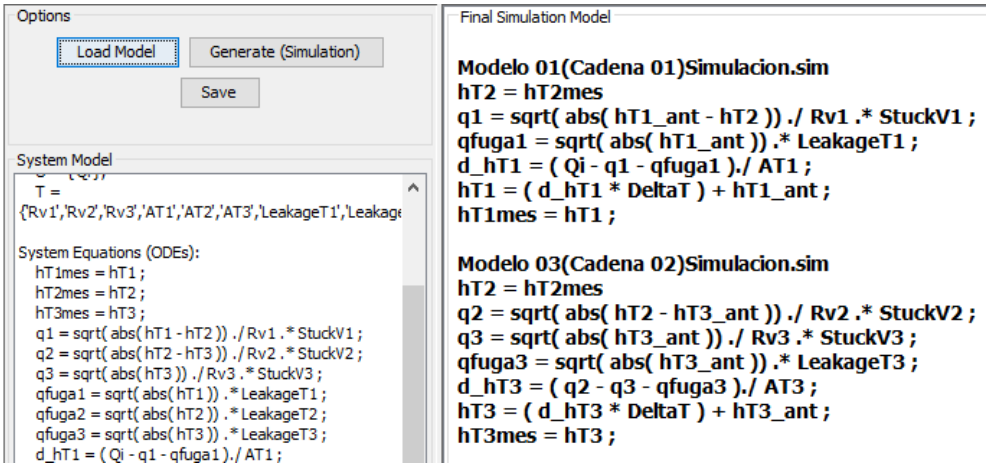


Figura 7.5 : Modelos de simulación fusionados

Para ver como ha funcionado la fusión de la figura 7.5 ilustramos con la siguiente imagen, la 7.6, de los dos modelos de los cuales parte el Modelo 01(Cadena 01) para acabar con esa fusión.

```
hT2 = hT2mes
q1 = sqrt( abs( hT1_ant- hT2)) ./ Rv1 .* StuckV1 + sqrt( abs( hT1_ant)) .* LeakageT1 ;
d_hT1 = ( Qi - q1 )./ AT1 ;
hT1 = ( d_hT1 * DeltaT ) + hT1_ant ;
hT1mes = hT1 ;
```

```
hT1_2 = hT1mes
hT2 = hT2mes
q1 = sqrt( abs( hT1_ant- hT2)) ./ Rv1 .* StuckV1 + sqrt( abs( hT1_ant)) .* LeakageT1 ;
d_hT1 = ( Qi - q1 )./ AT1 ;
hT1 = ( d_hT1 * DeltaT ) + hT1_ant ;
```

Figura 7.6 : Modelos de simulación sin fusionar

CP-08	Cálculo de residuos para sistemas continuos
Objetivo	Realizar el cálculo de los residuos de forma correcta de sistemas continuos.
Procedimiento	Con el sistema ya cargado, pulsar sobre la opción “Calculate Residuals” .
Salida Esperada	Se deberán mostrar los resultados del cálculo.
Resultado	El resultado obtenido es correcto.

CP-09	Cálculo de candidatos para sistemas continuos
Objetivo	Realizar el cálculo de los candidatos de forma correcta de sistemas continuos.
Procedimiento	Con el cálculo de residuos realizado, pulsar sobre la opción “Fault Candidates Calculation” .
Salida Esperada	Se deberán mostrar los resultados del cálculo.
Resultado	El resultado obtenido es correcto.

7.2.2.2. Pruebas con sistemas híbridos

Se van a realizar pruebas con varios sistemas híbridos y se van ilustrar con los dos sistemas híbridos que hemos creado para la herramienta, el de 3 tanques y el de 4 tanques que mencionamos en el apartado de introducción. Se realizan varios casos, para ver que el sistema no funciona solamente con el caso de prueba. El sistema de 4 tanques ya ha sido ilustrado con la Figura 1.1. El sistema de 3 tanques será el siguiente:

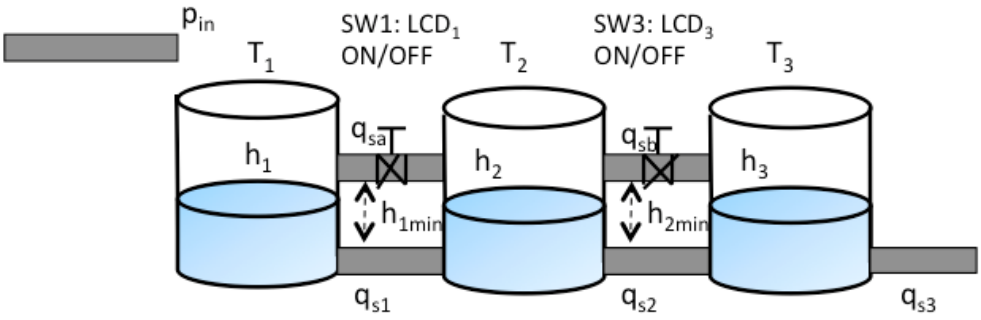


Figura 7.7 : Sistema de 3 tanques híbrido

CP-10	Carga de sistemas híbridos (Manual – Automático)
Objetivo	Cargar de sistemas híbridos y variables de ambas formas.
Procedimiento	Seleccionar la opción “Load Model” y cargar el nuevo sistema de 3 tanques de forma manual y automática.
Salida Esperada	Se deberá mostrar el sistema cargado. Dicha información deberá coincidir con la de los ficheros de entrada.
Resultado	El resultado obtenido es correcto.

CP-11	Conversión de condiciones de forma infija a prefija
Objetivo	Comprobar que la función de conversión de condiciones funciona de forma correcta.
Procedimiento	Tras la carga del sistema, se realiza la transformación de forma automática.
Salida Esperada	Se deberá mostrar las condiciones del sistema en forma prefija.
Resultado	El resultado obtenido es correcto.

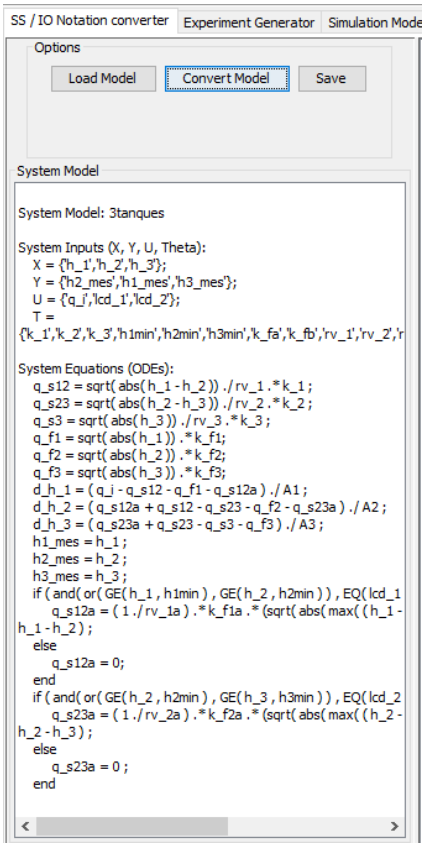


Figura 7.8 : Carga sistema 3 tanques híbrido – Conversión de condiciones

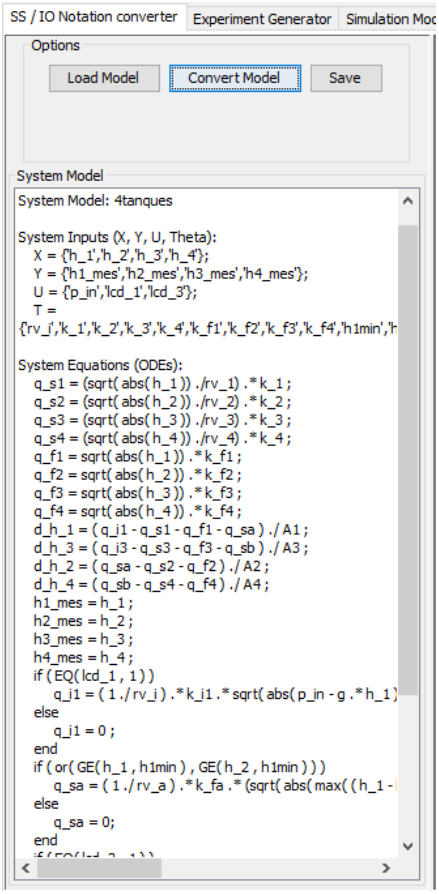


Figura 7.9 : Carga sistema 4 tanques híbrido – Conversión de condiciones

CP-12	Generación de un Modelo E/S del sistemas híbridos
Objetivo	Generar de forma correcta el Modelo E/S de sistemas híbridos.
Procedimiento	Con el sistema ya cargado, pulsar sobre la opción “Convert Model” en “SS/IO Notation Converter”.
Salida Esperada	Se deberá mostrar el Modelo E/S de forma correcta.
Resultado	El resultado obtenido es correcto.

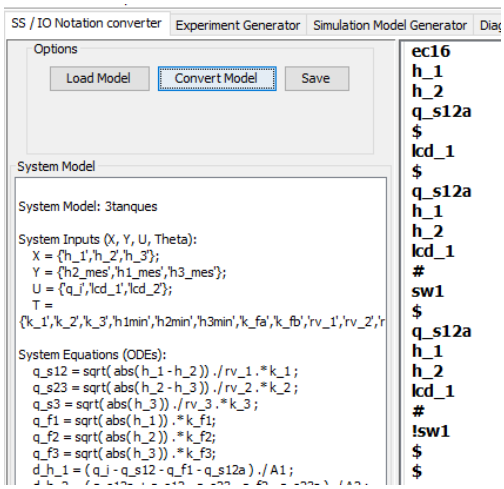


Figura 7.10 : Modelo E/S 3 tanques híbrido

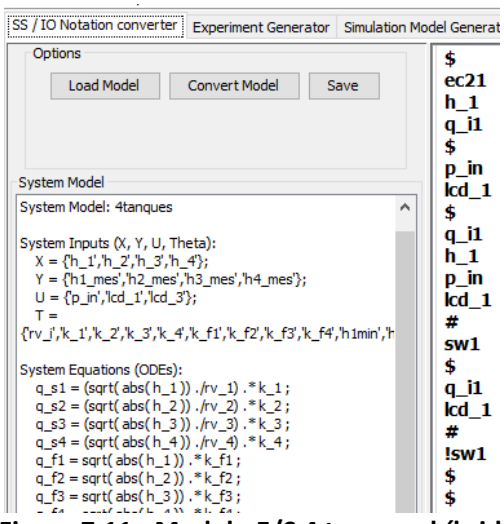


Figura 7.11 : Modelo E/S 4 tanques híbrido

CP-13	Generación de experimentos de sistemas híbridos
Objetivo	Generar un experimento de forma correcta de sistemas híbridos.
Procedimiento	Con el sistema ya cargado, pulsar sobre la opción “Generate Experiment” .
Salida Esperada	Se deberá mostrar el resultado de la ejecución de Matlab, es decir, el cálculo de las los valores de cada variable en cada instante.
Resultado	El resultado obtenido es correcto.

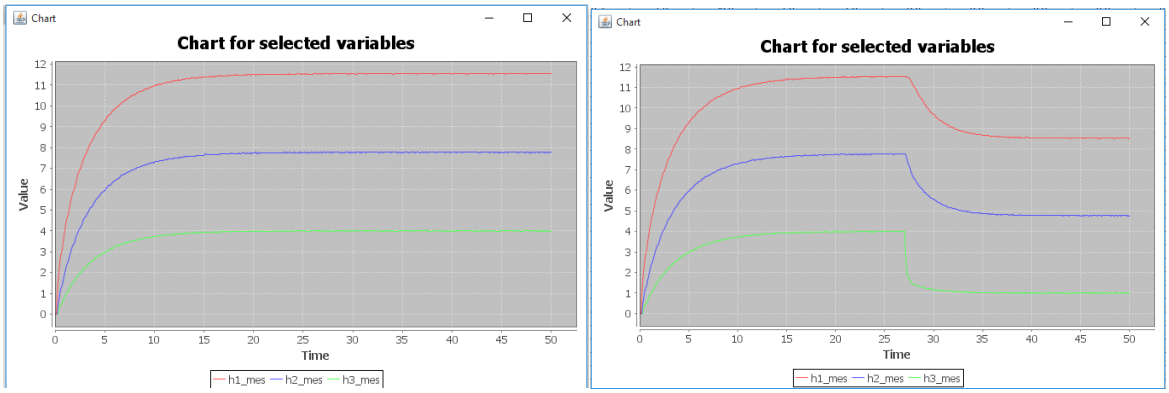


Figura 7.12 Experimentos sin fallo y con fallo en sistema de 3 tanques híbrido

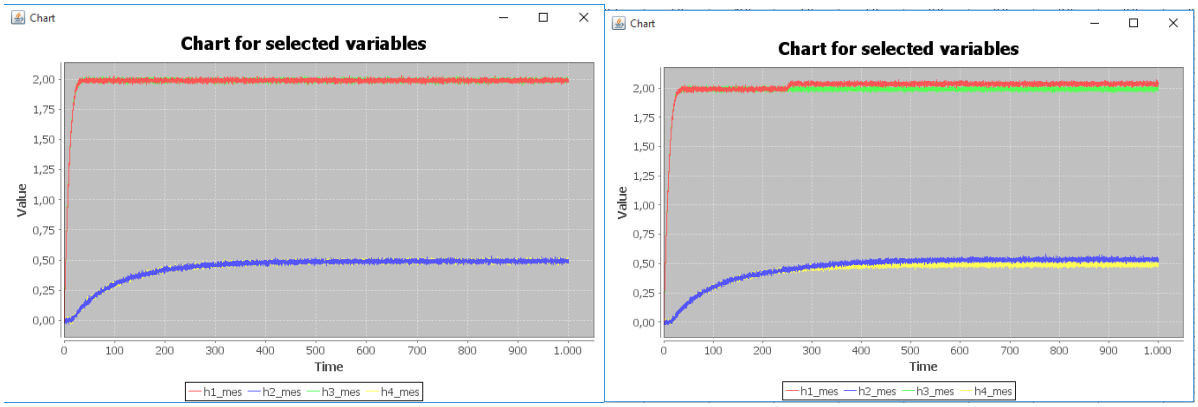


Figura 7.13 Experimentos sin fallo y con fallo en sistema de 4 tanques híbrido

CP-14	Carga de los MEMs de sistemas híbridos.
Objetivo	Cargar los MEMs generados por la herramienta de CPCs en el apartado "Simulation Model Generator" de sistemas híbridos.
Procedimiento	Seleccionar la opción "Load Model" dentro de Simulation Model Generator y cargar el sistema.
Salida Esperada	Se deberá mostrar los MEM de forma correcta. Dicha información deberá coincidir con la de los ficheros de entrada.
Resultado	El resultado obtenido es correcto.

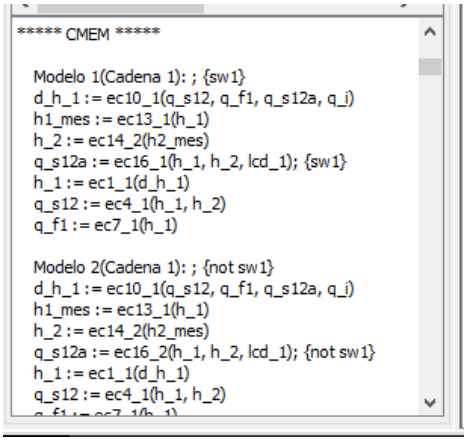


Figura 7.14 : Carga de MEM sistema 3 tanques híbrido

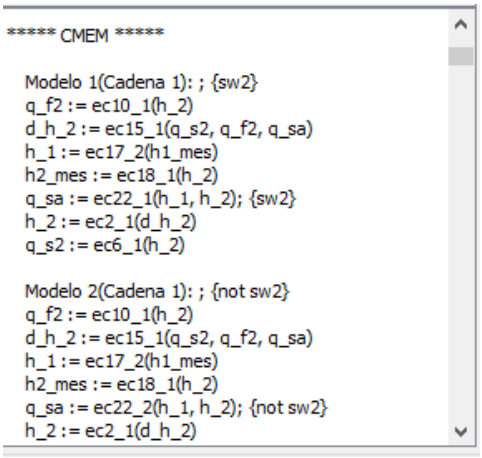


Figura 7.15 : Carga de MEM sistema 4 tanques híbrido

CP-15	Generación de los Modelos de Simulación de sistemas híbridos.
Objetivo	Generar los Modelos de Simulación para un sistema de 3 tanques híbrido forma correcta.
Procedimiento	Con el sistema ya cargado, pulsar sobre la opción "Generate(Simulation)" .
Salida Esperada	Se deberán mostrar los modelos generados.
Resultado	El resultado obtenido es correcto.

CP-16	Fusión en la generación de Modelos de Simulación de sistemas híbridos.
Objetivo	Que se produzca la fusión de modelos en la generación.
Procedimiento	La fusión se realiza de forma automática en la generación.
Salida Esperada	Se deberán mostrar los modelos generados fusionados.
Resultado	El resultado obtenido es correcto.

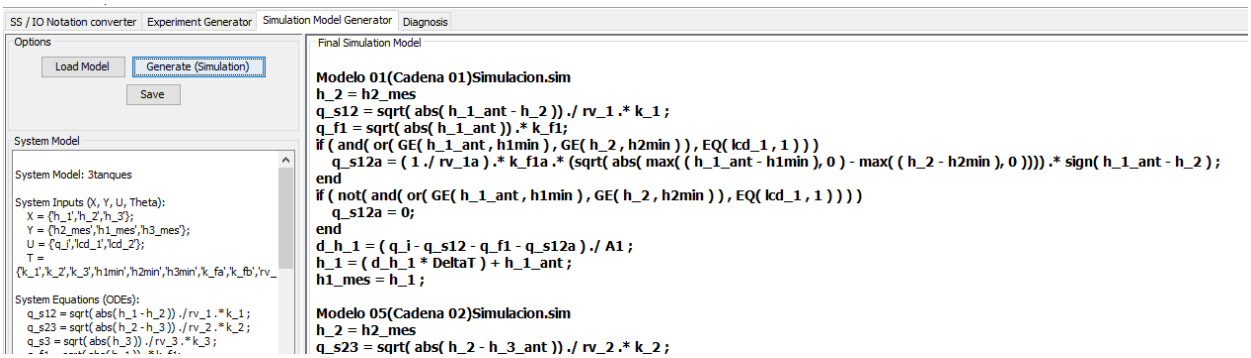


Figura 7.16 : Modelos de Simulación fusionados sistema 3 tanques híbrido

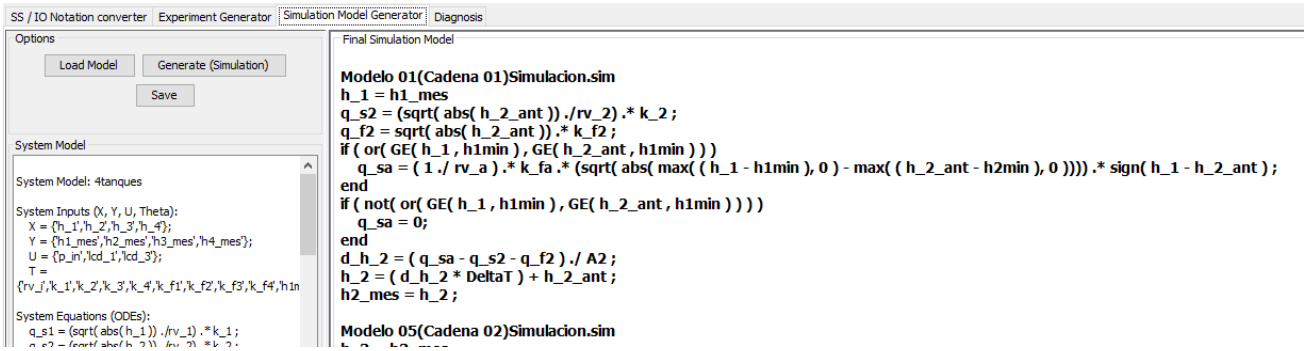


Figura 7.17 : Modelos de Simulación fusionados sistema 3 tanques híbrido

Para ver como ha funcionado la fusión de la figura 7.17 ilustramos con la siguiente imagen, la 7.18, de los 4 modelos de los cuales parte el Modelo 01(Cadena 01) para acabar con esa fusión.

```
h_1 = h1_mes
q_s2 = (sqrt( abs( h_2_ant )) ./rv_2) .* k_2 ;
q_f2 = sqrt( abs( h_2_ant )) .* k_f2;
if ( or( GE( h_1 , h1min ) , GE( h_2_ant , h1min ) ) )
    q_sa = ( 1 ./ rv_a ) .* k_fa .* (sqrt( abs( max( ( h_1 - h1min ), 0 ) - max( ( h_2_ant - h2min ), 0 )))) .* sign( h_1 - h_2_ant ) ;
end
d_h_2 = ( q_sa - q_s2 - q_f2 ) ./ A2 ;
h_2 = ( d_h_2 *DeltaT ) + h_2_ant ;
h2_mes = h_2 ;

h_1 = h1_mes
q_s2 = (sqrt( abs( h_2_ant )) ./rv_2) .* k_2 ;
q_f2 = sqrt( abs( h_2_ant )) .* k_f2;
if ( not( or( GE( h_1 , h1min ) , GE( h_2_ant , h1min ) ) ) )
    q_sa = 0;
end
d_h_2 = ( q_sa - q_s2 - q_f2 ) ./ A2 ;
h_2 = ( d_h_2 *DeltaT ) + h_2_ant ;
h_2_2 = h2_mes ;

h_1 = h1_mes
q_s2 = (sqrt( abs( h_2_ant )) ./rv_2) .* k_2 ;
q_f2 = sqrt( abs( h_2_ant )) .* k_f2;
if ( or( GE( h_1 , h1min ) , GE( h_2_ant , h1min ) ) )
    q_sa = ( 1 ./ rv_a ) .* k_fa .* (sqrt( abs( max( ( h_1 - h1min ), 0 ) - max( ( h_2_ant - h2min ), 0 )))) .* sign( h_1 - h_2_ant ) ;
end
d_h_2 = ( q_sa - q_s2 - q_f2 ) ./ A2 ;
h_2 = ( d_h_2 *DeltaT ) + h_2_ant ;
h2_mes = h_2 ;

h_1 = h1_mes
q_s2 = (sqrt( abs( h_2_ant )) ./rv_2) .* k_2 ;
q_f2 = sqrt( abs( h_2_ant )) .* k_f2;
if ( not( or( GE( h_1 , h1min ) , GE( h_2_ant , h1min ) ) ) )
    q_sa = 0;
end
d_h_2 = ( q_sa - q_s2 - q_f2 ) ./ A2 ;
h_2 = ( d_h_2 *DeltaT ) + h_2_ant ;
h_2_2 = h2_mes ;
```

Figura 7.18: Modelos de Simulación sin fusionar

CP-17	Carga de los Modelos de Simulación de sistemas híbridos.
Objetivo	Cargar los modelos de simulación que ha generado la herramienta de sistemas híbridos.
Procedimiento	Seleccionar la opción “Load Model” dentro de “Diagnosis” y cargar el sistema híbrido.
Salida Esperada	Se deberá mostrar los modelos de simulación de forma correcta. Dicha información deberá coincidir con la de los ficheros de entrada.
Resultado	El resultado obtenido es correcto.

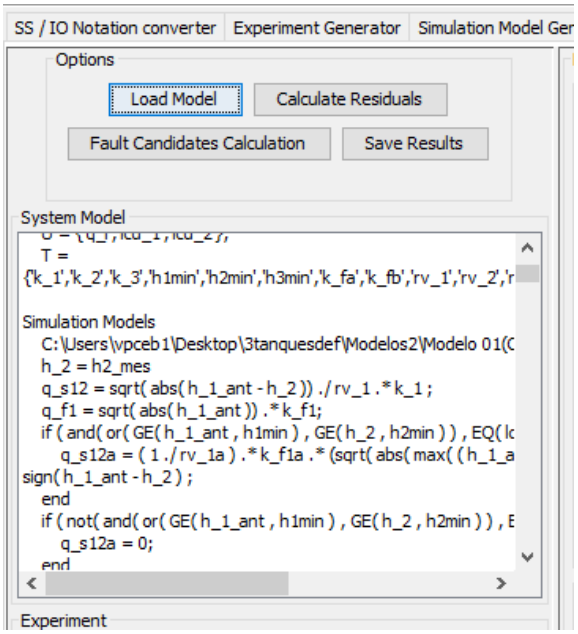


Figura 7.19 : Modelo de Simulación cargado 3 tanques híbrido

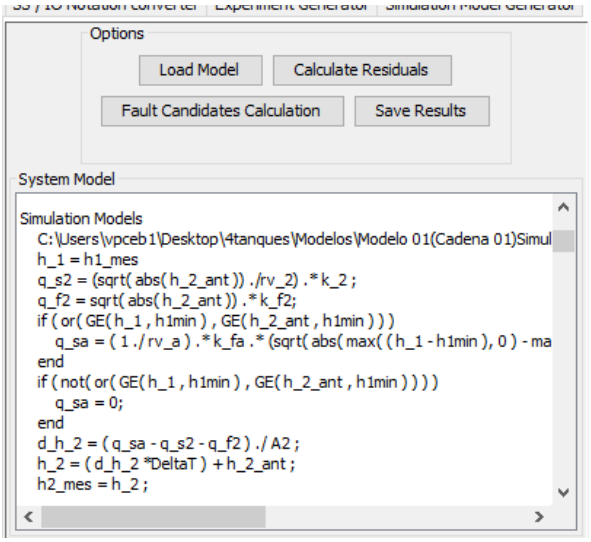


Figura 7.20 : Modelo de Simulación cargado 4 tanques híbrido

CP-18	Cálculo de residuos para sistemas híbridos.
Objetivo	Realizar el cálculo de los residuos de forma correcta de sistemas híbridos.
Procedimiento	Con el sistema ya cargado, pulsar sobre la opción “Calculate Residuals” .
Salida Esperada	Se deberán mostrar los resultados del cálculo.
Resultado	El resultado obtenido es correcto.

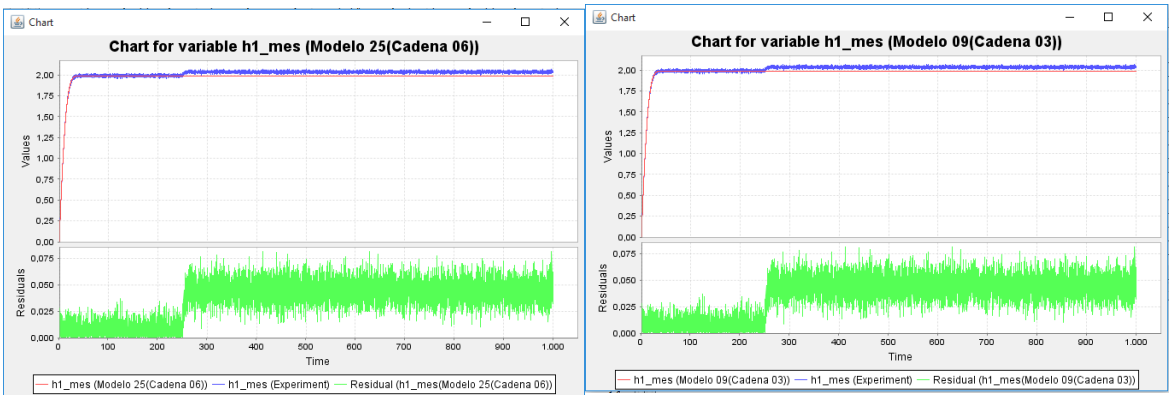


Figura 7.21 : Cálculo de residuos de Modelos de Simulación con fallo del sistema

CP-19	Cálculo de candidatos para sistemas híbridos
Objetivo	Realizar el cálculo de los candidatos de forma correcta de sistemas híbridos.
Procedimiento	Con el cálculo de residuos realizado, pulsar sobre la opción “Fault Candidates Calculation” .
Salida Esperada	Se deberán mostrar los resultados del cálculo.
Resultado	El resultado obtenido es correcto.

Para ilustrar el CP-19 se van a colocar imágenes del resultado del z-test para los residuos mostrados en la figura 7.21 con diferentes umbrales, veremos así como el instante de detección es más o menos preciso según que parámetros le introducimos para el cálculo:

- Mismos umbrales diferente nivel de significancia:

Prueba A	Caso 1	Caso 2	Caso 3
Reference Windows Size	40	40	40
Sample Windows Size	40	40	40
Window Mismatch	20	20	20
Significance Level(alpha)	0.01	0.05	0.1

```
Starting candidates calculation
------------------------

Instant: 287.9
Activated PC: Modelo 09(Cadena 03)
Current candidate set: [ec17] [ec18] [ec22] [ec1] [ec13] [ec21] [ec5] [ec9]

Instant: 287.9
Activated PC: Modelo 25(Cadena 06)
Current candidate set: [ec17] [ec22] [ec1] [ec13] [ec21] [ec5] [ec9] [ec18 ec10] [ec18 ec15] [ec18 ec2] [ec18 ec6]
```

```
Starting candidates calculation
------------------------

Instant: 283.1
Activated PC: Modelo 09(Cadena 03)
Current candidate set: [ec17] [ec18] [ec22] [ec1] [ec13] [ec21] [ec5] [ec9]

Instant: 283.1
Activated PC: Modelo 25(Cadena 06)
Current candidate set: [ec17] [ec22] [ec1] [ec13] [ec21] [ec5] [ec9] [ec18 ec10] [ec18 ec15] [ec18 ec2] [ec18 ec6]
```

```
Starting candidates calculation
-------------------------

Instant: 283.1
Activated PC: Modelo 09(Cadena 03)
Current candidate set: [ec17] [ec18] [ec22] [ec1] [ec13] [ec21] [ec5] [ec9]

Instant: 283.1
Activated PC: Modelo 25(Cadena 06)
Current candidate set: [ec17] [ec22] [ec1] [ec13] [ec21] [ec5] [ec9] [ec18 ec10] [ec18 ec15] [ec18 ec2] [ec18 ec6]
```

Figura 7.22: Cálculo de candidatos Prueba A: caso 1 - caso 2 - caso 3

- Misma significancia diferentes umbrales:

Prueba B	Caso 1	Caso 2	Caso 3
Reference Windows Size	20	60	80
Sample Windows Size	20	60	80
Window Mismatch	10	30	40
Significance Level(alpha)	0.1	0.1	0.1

```
Starting candidates calculation
-------------------------

Instant: 264.8
Activated PC: Modelo 09(Cadena 03)
Current candidate set: [ec17] [ec18] [ec22] [ec1] [ec13] [ec21] [ec5] [ec9]

Instant: 264.8
Activated PC: Modelo 25(Cadena 06)
Current candidate set: [ec17] [ec22] [ec1] [ec13] [ec21] [ec5] [ec9] [ec18 ec10] [ec18 ec15] [ec18 ec2] [ec18 ec6]
```

```
Starting candidates calculation
-------------------------

Instant: 291.2
Activated PC: Modelo 09(Cadena 03)
Current candidate set: [ec17] [ec18] [ec22] [ec1] [ec13] [ec21] [ec5] [ec9]

Instant: 291.2
Activated PC: Modelo 25(Cadena 06)
Current candidate set: [ec17] [ec22] [ec1] [ec13] [ec21] [ec5] [ec9] [ec18 ec10] [ec18 ec15] [ec18 ec2] [ec18 ec6]
```

```
Starting candidates calculation
-------------------------

Instant: 304.7
Activated PC: Modelo 09(Cadena 03)
Current candidate set: [ec17] [ec18] [ec22] [ec1] [ec13] [ec21] [ec5] [ec9]

Instant: 304.7
Activated PC: Modelo 25(Cadena 06)
Current candidate set: [ec17] [ec22] [ec1] [ec13] [ec21] [ec5] [ec9] [ec18 ec10] [ec18 ec15] [ec18 ec2] [ec18 ec6]
```

Figura 7.23: Cálculo de candidatos Prueba B: caso 1 - caso 2 - caso 3

8. MANUAL DE INSTALACIÓN

En el CD de entrega se encuentra el archivo de instalación “DxPCsInstaller”, ejecutándole y siguiendo los pasos requeridos se instalará la herramienta. Una vez instalada, para un correcto funcionamiento, se debe ejecutar la herramienta en modo administrador.

Debe tener instalado en su equipo previamente una versión de Matlab reciente que puede descargar de http://es.mathworks.com/products/matlab/?requestedDomain=www.mathworks.com . Además se deberá de introducir en Matlab la carpeta “funciones” que va dentro de la carpeta del software en versión fuente, dentro del CD de entrega.

Debe tener instalado en su equipo previamente una versión de Java reciente que puede descargar de https://www.java.com/es/download/.

9. MANUAL DE USUARIO

Como no existe ningún manual anterior ni ningún proyecto que hable sobre esta herramienta, se va a realizar un manual sobre la totalidad de la misma para que el usuario sea capaz de usarla sin problema alguno. El usuario tipo de esta herramienta es un estudiante o investigador que ha recibido previamente una formación en diagnóstico basado en modelos mediante PCs y que entiende el modelado de sistemas híbridos. Primero se van a definir los diferentes tipos de ficheros que usa la herramienta y después se va a explicar el funcionamiento de la misma.

9.1. Tipos de ficheros

- **Fichero de ecuaciones**
  El fichero que contiene las ecuaciones es un fichero con extensión .m. Para ver cómo debemos escribirlo usamos la siguiente ecuación que modela el comportamiento híbrido del caudal de salida entre los tanques 3 y 4 como se puede ver en la figura 1.1.

```
if ( ( h_3 >= h3min ) || ( h_4 >= h4min ) )
q_sb = ( 1 ./ rv_b ) .* k_fb .* ( sqrt( abs( max( ( h_3 -
h3min ), 0 ) - max( ( h_4 - h4min ), 0 )))) .* sign( h_3 -
h_4 ) ;
else
q_sb = 0 ;
end
```

  Cada elemento se escribe en una línea distinta. Sólo las ecuaciones acaban en punto y coma. Tanto variables como funciones respetan el espaciado. En el caso de las condiciones cabe tener en cuenta que se va a realizar una transformación de forma infija a prefija por lo cual es importante respetar los paréntesis ya que de ellos se guía la función de la transformación. Destacar también que si en una de las condiciones hubiera un numero negativo iría entre paréntesis tal que (-1).

  Otro punto será cómo se deben escribir las condiciones: éstas deben ir escritas en forma infija, más sencilla para el usuario, ya que es la herramienta la que se encarga de la transformación posteriormente.

- **Fichero de variables**
  Es un fichero en texto plano ASCII. En él se colocan los conjuntos de las variables cada uno en una línea. Los conjuntos de variables son: X, Y, U, T. El orden de los conjuntos será: X, Y, U, T. Usamos el siguiente conjunto como ejemplo:

```
X = {'h_1','h_2','h_3','h_4'};
```

  O en caso de ir vacío:

```
X = { };
```

Primero se nombra el conjunto y después se colocan las variables cada una entre comillas simples, separados por comas y todo entre llaves. Al final acabamos la línea con ";".

- **Fichero de condiciones iniciales**
  Es un fichero con los valores que tienen las variables en el primer instante marcado, en texto plano ASCII. Se escribirán los datos en dos líneas, en la primera los nombres de todas las variables y en la segunda los valores de las mismas. Las posiciones de los valores de la segunda línea corresponden con las posiciones de los nombres de la primera. Cada nombre y valor irán separados por un carácter tabulador. El tiempo de inicio se escribirá con el nombre "Time" en la primera posición. Usamos el siguiente ejemplo:

Time	*p_in*	*lcd_1*	*lcd_3*
0	0	0	0

- **Fichero de configuración**
  Es un fichero en el cual se define el tiempo y condiciones de la simulación, con extensión .config, en texto plano ASCII. Las separaciones entre elementos son un carácter tabuladore. Contiene 3 parámetros:

  - *Tsim*: número de instantes de simulación.
  - *DeltaT*: paso de integración de las ecuaciones diferenciales.
  - *Umbral*: umbral de activación de PCs en el cálculo de candidatos.

  Usamos el siguiente ejemplo:

Tsim	*DeltaT*	*Umbral*
50	0.1	0.25

- **Fichero de fallos**
  Es un fichero en el cual se marca donde se va a introducir un fallo o varios en el sistema si se quisiera, en texto plano ASCII. Las separaciones entre elementos son un carácter tabulador. Contiene 4 parámetros:

  - *NumParam*: será un número que indica la posición en el fichero variables del parámetro que va a fallar.
  - *Tipo*: indica la forma de fallo que será aditivo, en cuyo caso pondremos ADD, o multiplicativo, en cuyo caso pondremos MULT.
  - *FailStep*: define el instante en el cual se producirá el fallo.
  - *ErrorMagnitude*: es el factor de actualización del fallo. Es un valor real de doble precisión y no un entero.

  Usamos el siguiente ejemplo que significa: En el primer caso es un fallo de tipo aditivo en el parámetro 7, en el instante 10 y de valor 1, es decir, en el instante 10 sumará 1 al valor del parámetro 7. En el segundo es un fallo de tipo multiplicativo en el parámetro 3 en el instante 20 y de valor 0.5, es decir, en el instante 20 multiplicará el valor del parámetro 3 por 0.5.

NumParam	*Tipo*	*FailStep*	*ErrorMagnitude*
7	ADD	10	1.0
3	MULT	20	0.5

- **Valores de entrada**
  Es un fichero en el cual se marcan los valores que van a tomar las variables de entrada en cada instante, en texto plano ASCII. Ejemplo si tuviéramos que *Tsim* es 1, *DeltaT* 0.25 y las variables de entrada X e Y con valor 1 y 0 constante respectivamente, el fichero quedaría:

  ```
  X       Y
  1.0     0.0
  1.0     0.0
  1.0     0.0
  1.0     0.0
  end
  ```

  El tiempo de simulación 1 con paso 0.25 nos deja 4 valores. Las columnas van separadas por un espacio de tabulador.

- **Rutas**
  Este fichero, en texto plano ASCII, sirve para proporcionar a la aplicación los nombres de todos los ficheros necesarios, evitando la carga manual. En él se marcan todos los ficheros que puede usar la aplicación poniendo delante un nombre y luego la ruta donde se encuentran. Usamos el siguiente ejemplo:

  Model: *C:\Users\4tanques.m*
  Variables: *C:\Users\variables.txt*
  Initial Conditions: *C:\Users\CondicionesIniciales.txt*
  Configuration Setup: *C:\Users\configuracionexperimento.config*
  CPCs Model: *C:\Users\PCs.txt*
  Input Data: *C:\Users\valoresdeentrada.txt*
  Parameters: *C:\Users\parametros.txt*
  Fails: *C:\Users\fallo.txt*
  Experiment Results File: *C:\Users\Experimento.txt*
  Simulation MEMs: *C:\Users\Modelos*

  Hemos marcado en negrita los nombres que hay que introducir para que la herramienta encuentre la ruta absoluta. La ruta de ejemplo sería si nuestros ficheros estuvieran en Users en C. Destacar que en los modelos de simulación habrá que marcar la carpeta, no los ficheros, en la cual se encuentren los modelos.

- **Otros ficheros**
  Hay otros ficheros que también usará la herramienta pero que no los generamos nosotros de forma manual sino que son el resultado de alguna de las funciones de nuestra herramienta como los resultados de experimentos, los Modelos E/S o los Modelos de Simulación. También usará otros como el resultado del cálculo de PCs de la herramienta CPCs tanto en formato .txt como en .xml.

9.2. Funcionalidad de la herramienta

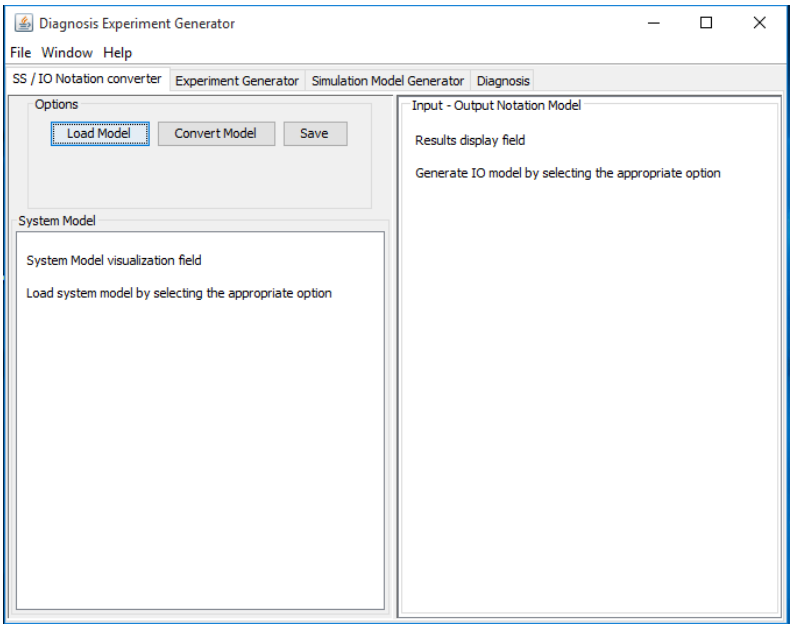


Figura 9.1 : Interfaz herramienta

En la figura 9.1 se puede ver la interfaz de la herramienta. Como vemos tiene una parte superior donde hay unas pestañas que dividen la funcionalidad. El resto se divide en dos partes, la izquierda con una parte superior con botones y la inferior donde se ven los elementos cargados, y la derecha donde se ven los resultados.

En la parte superior tenemos 3 opciones, “File”, “Window” y “Help”. “Window” es otra forma de cambiar entre pestañas y “Help” es un simple botón de ayuda. El importante es “File”, desde el que tenemos dos opciones para generar de forma automática un fichero de entrada y uno de fallos.

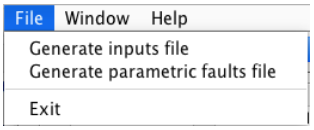


Figura 9.2: Botón File

El primero, “Generate inputs file” en primer lugar nos pedirá seleccionar el archivo de variables del sistema y después seguir los pasos:

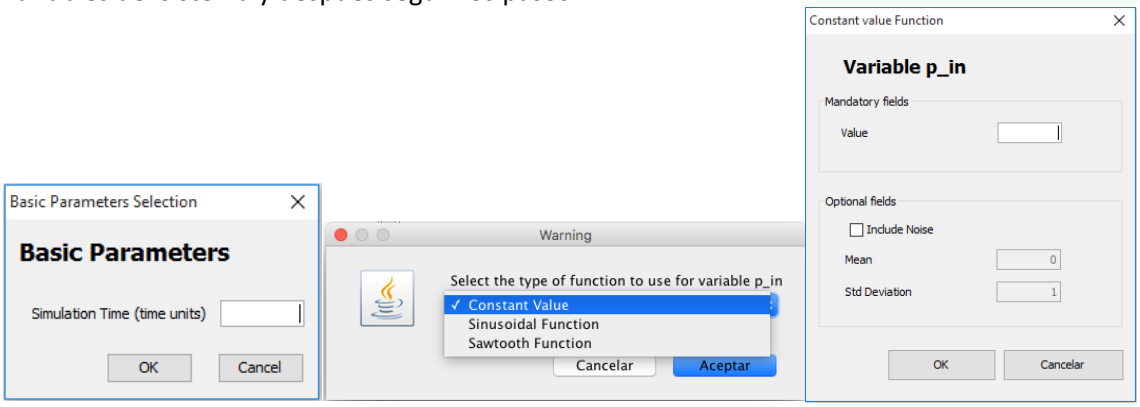


Figura 9.3: Pasos de creación del fichero de entradas automático

Primero indicamos el tiempo de simulación, después elegimos el tipo de función, en el caso elegido constante, y después elegimos el valor de la variable o si fuera otro tipo los valores que nos pida. Tras eso elegiríamos dónde guardar el nuevo fichero y el nombre del mismo.

En el segundo caso podemos generar los fallos de forma automática para lo cual lo primero que hará será pedirnos el fichero de parámetros y después seguir los pasos:

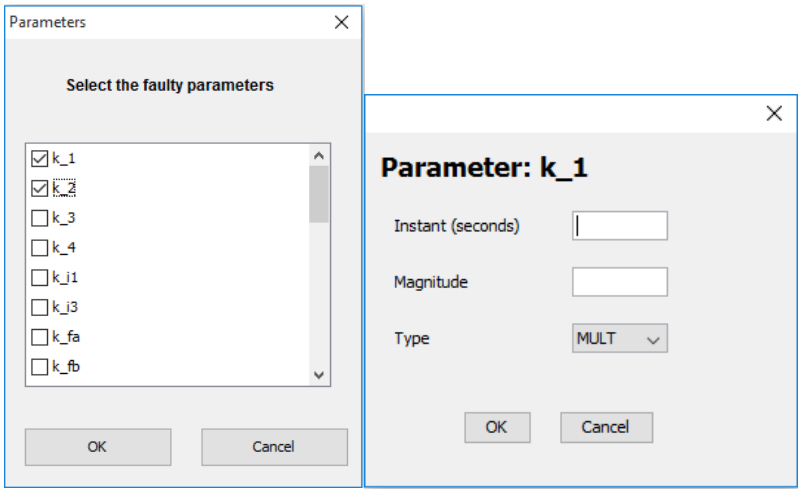


Figura 9.4 : Pasos de creación del fichero de fallos automático

Primero elegiríamos de la lista aquellos candidatos a fallo y después introduciríamos los datos necesarios para identificar el fallo. Tras eso elegiríamos dónde guardar el nuevo fichero y el nombre del mismo.

Tras esto tenemos un segundo menú que agrupa 4 pestañas cada una de cuales agrupa una funcionalidad.

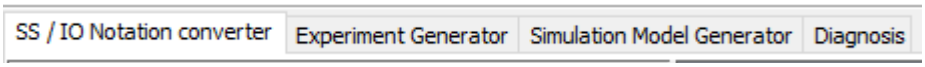


Figura 9.5 : Menú

Todas estas pestañas tienen dos botones en común, “Load Model” para introducir los datos y “Save” para guardar los resultados de la ejecución. El guardado de los resultados es simple, al pulsar en el botón “Save” (“Save Results” en el caso de Diagnosis) eliges la ruta del archivo y el nombre y guardará fichero de texto plano ASCII con los resultados. La introducción de los datos comenzará tras pulsar el botón “Load Model” de cualquiera de las pestañas que nos indicará si queremos hacer la carga manual o automática (“Path File”) como vemos en la imagen.

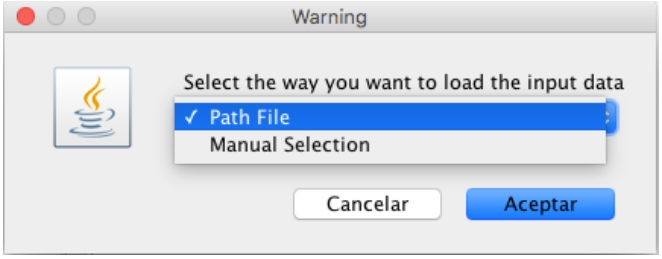


Figura 9.6 : Carga manual o automática

Si elegimos la carga automática simplemente nos pedirá el fichero de rutas que hemos descrito en el apartado anterior. Si elegimos la carga manual dependiendo de la pestaña que estemos necesitará unos archivos u otros que iremos eligiendo uno a uno. En esta carga manual habrá dos excepciones en la carga, las que hemos explicado en la opción "File", el fichero de configuración y el de fallos, que nos ofrecerá la opción de cargar desde un fichero o crearlo tal y como hemos explicado anteriormente eligiendo "Introduce interactively".

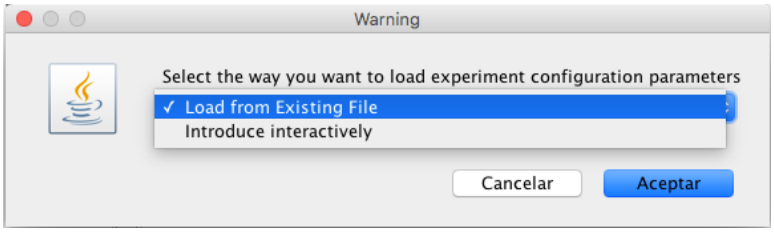
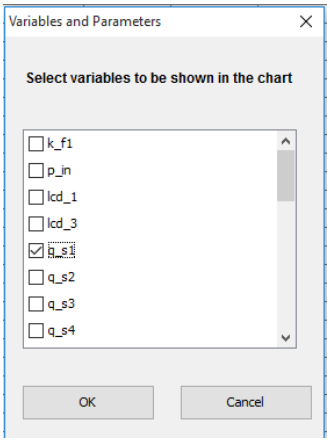


Figura 9.7 : Carga desde fichero existente o crear nuevo

Otra funcionalidad que es común en dos de las pestañas es la muestra de un gráfico de los resultados, que se encuentra presente en "Experiment Generator" y "Diagnosis". En "Experiment Generator" el botón aparece junto al resto de botones sin embargo en "Diagnosis" este botón aparece después de hacer el cálculo de residuos encima de los resultados. Tras pulsar el botón "Display Graph", tras generar un experimento o realizar el cálculo de residuos nos aparece un menú en el cual elegiremos cuáles son las variables que queremos ver representadas en la gráfica.



Figuras 9.8 : Selección de variables a mostrar en gráfica

Finalmente cada una de las pestañas tiene su funcionalidad principal en el botón o botones restantes, en cada caso son:

- **SS/IO Notation Converter**
  Una vez que el sistema esté cargado tenemos el botón "Convert Model" que al pulsarlo generará un Modelo E/S a partir de los datos cargados y lo mostrará al usuario en el panel derecho.

- **Experiment Generator**
  Una vez que el sistema esté cargado tenemos el botón "Generate Experiment" que al pulsarlo generará un experimento a partir de los datos cargados y lo mostrará al usuario en el panel derecho en forma de tabla.

- **Simulation Model Generator**
  Una vez que el sistema esté cargado tenemos el botón "Generate(Simulation)" que al pulsarlo generará los Modelos de Simulación fusionados a partir de los datos cargados y lo mostrará al usuario en el panel derecho.

- **Diagnosis**
  En este caso, una vez que el sistema esté cargado, la funcionalidad de Diagnosis tiene dos pasos. En primer lugar de calculan los residuos y en segundo lugar a partir de los residuos se calculan los candidatos. Cada uno de estos pasos es configurable:

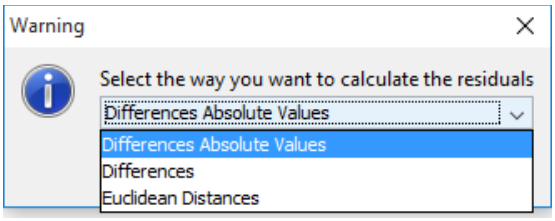


Figura 9.9 : Opciones de cálculo de residuo

Seleccionamos el modo de cálculo que deseemos y tras ello nos mostrará los resultados en el panel superior derecho en forma de tabla. Como hemos dicho antes, aparecerá el botón para la muestra de los resultados en gráfica justo encima.

Tras tener realizado el cálculo de residuos ya podemos proceder al cálculo de candidatos para ello pulsamos el botón "Fault Candidates Calculation" que nos dará dos opciones:

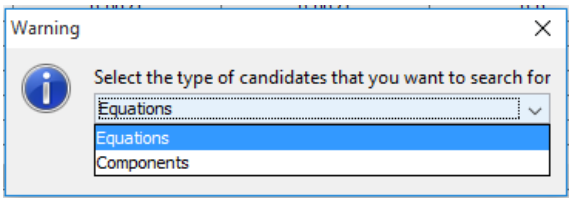


Figura 9.10 : Opciones de cálculo de candidatos

Seleccionamos una de las dos opciones, según deseemos realizar el cálculo y nos pedirá entonces el fichero en formato .xml de las definiciones de los PCs. Tras esto nos pedirá que seleccionemos el modo de umbral para los PCs, para el que tenemos 3 opciones:

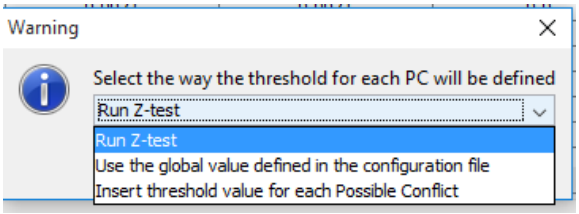


Figura 9.11 : Opciones de selección de umbral

Tras la selección, la segunda opción directamente nos ofrecerá los resultados, las otras, cada una nos llevará a una nueva ventana donde se introducirán los valores de umbral o en el caso del Z-test los valores de las ventanas y el alfa. El

Z-test es un test estadístico que sigue una distribución normal según la hipótesis nula. Al Z-test le introducimos 4 valores:

- Reference Window Size: es el número de unidades de tiempo que tomamos para ventana de referencia.
- Sample Window Size: es el número de unidades de tiempo que tomamos para la ventana de muestra .
- Window Mismatch: es el número de unidades de tiempo que tomamos para el desfase.
- Significance Level(alpha): nivel de significancia, se conoce como el error máximo al momento de rechazar la hipótesis nula cuando es verdadera. Tiene 3 grados, si vale 0.01 es muy significativo, si vale 0.05 es significativo y si vale 0.1 es poco significativo.

Tras introducir los valores nos mostrará los resultados en el panel inferior derecho.

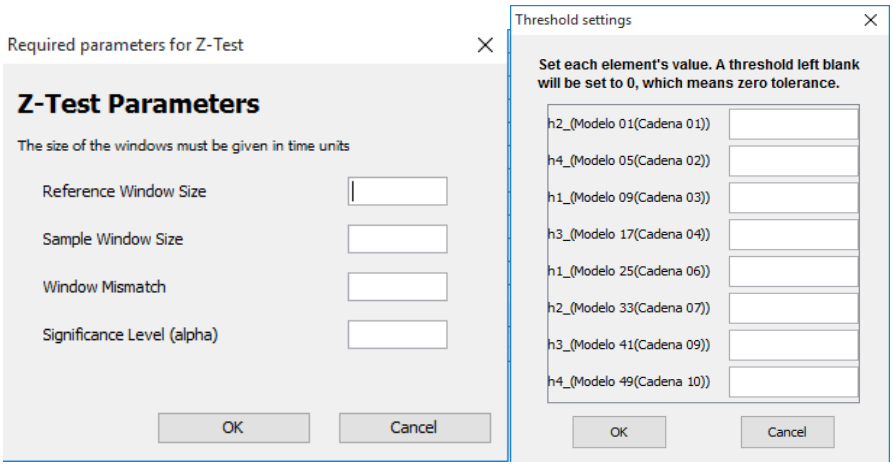


Figura 9.12 : Z-test / Introducción de valores de umbral

10. CONCLUSIONES

Tras la finalización de este TFG, podemos concluir que se han cumplido los objetivos inicialmente fijados, y en base a ellos se puede afirmar que:

- Se ha analizado la versión previa de la aplicación.
- Se han introducido los sistemas híbridos como nuevo modo de funcionamiento de la herramienta.
- Se han introducido nuevos algoritmos añadiendo nueva funcionalidad a la herramienta.

Mediante la consecución de dichos objetivos, el alumno ha adquirido cierta soltura con un lenguaje en el que tenía escasa experiencia previa (Java).

Se han obtenido conocimientos sobre la diagnosis basada en modelos con Posibles Conflictos, técnica sobre la cual antes de comenzar no se tenía ninguna noción de su funcionamiento.

Por último, pero no menos importante, al haber trabajado sobre un sistema ya construido previamente, se ha tomado conciencia de la enorme importancia de contar con un código debidamente comentado y una documentación sobre el desarrollo del software correcta, completa y consistente con el producto implementado, facilitando así la tarea de mantenimiento y la realización de futuras ampliaciones.

Como ampliación futura se podría implementar una base de datos que guardase los resultados de los experimentos facilitando el trabajo y evitando el molesto uso de los ficheros.

11. BIBLIOGRAFÍA

- (Balakrishnan y Honavar,1998) Balakrishnan, K., Honavar, V. (1998). Intelligent diagnosis systems. Journal of Intelligent Systems, 8(3-4), 239-290.

- (Domínguez, 2013) Domínguez, M. (2013). Revisión y adaptación de la herramienta para el cálculo de Posibles Conflictos. Trabajo de Fin de Grado. Universidad de Valladolid.

- (Gilat, 2006) Gilat, A. (2006). Matlab: Una introducción con ejemplos prácticos. Reverté.

- (Hernández, 2012) Alberto Hernández Cerezo. "Generación automática de modelos para diagnosis usando PCs". Proyecto Fin de Carrera. Sept. 2012. ETS Ingeniería Informática de Valladolid. Dirección: Belarmino Pulido y Noemí Moya.

- (Moya y cols, 2012) Moya, N., Pulido, B., Alonso-González, C., Bregon, A., & Rubio, D. (2012, July). Automatic generation of Dynamic Bayesian Networks for hybrid systems fault diagnosis. In Proceeding of Intl. Workshop on principles of Diagnosis, DX.

- (Pulido, 2016) B. Pulido, C.J. Alonso-González, A. Bregon, A. Hernández, D. Rubio, L.M. Villarroel. "DxPCs: A toolbox for model-based diagnosis of dynamic systems using Possible Conflicts". En Progress in Artificial Intelligence. Vol. 5, Issue, 2. Pgs. 111-120. Springer. Mayo, 2016. DOI: 10.1007/s13748-015-0078-5. ISSN: 2192-6352 ISSN: 2192-6360 (electronic version).

- (Pulido y Gelso, 2005) Pulido, B., Gelso, E. (2005, Noviembre). Viabilidad de una técnica de compilación de dependencias para diagnosis basada en modelos en tareas de supervisión. Artículo en las actas y Ponencia, 21-30, Vol. II . XI Conferencia de la Asociación Española para la Inteligencia Artificial, CAEPIA-2005. Santiago de Compostela, España.