**Universidad de Valladolid**

E.T.S Ingeniería Informática

**TRABAJO FIN DE GRADO**

Grado en Ingeniería Informática

# Revisión y adaptación de la herramienta para el cálculo de Posibles Conflictos


Autor:
**D. Manuel Domínguez Prieto**
Tutor:
**D. J. Belarmino Pulido Junquera**

# Agradecimientos

A mi familia, en especial a mis padres Enrique y Mª José, y a mi futura esposa Esther por sus constantes ánimos y apoyo durante la realización de este trabajo; y a mi tutor Belarmino por su completa disponibilidad y su inestimable ayuda ante todos los problemas que me han ido surgiendo.

# Resumen

Este TFG analiza y mejora una herramienta para calcular posibles conflictos, realizada en el lenguaje de programación Java aprovechando las mejoras introducidas en las nuevas versiones de dicho lenguaje.

El objetivo de esta herramienta es, a partir de la definición de un sistema, detectar dónde pueden producirse conflictos; es decir, discrepancias entre valores observados y valores predichos. Estas tareas son previas a su utilización en el diagnóstico basado en modelos.

Esta herramienta usa algoritmos que, en sistemas medianamente complejos, requieren mucho tiempo para finalizar; principalmente debido a operaciones de búsqueda, inserción y borrado de elementos en listas considerablemente grandes. Este tiempo puede reducirse usando estructuras de datos ordenadas y aprovechando algoritmos de búsqueda que hacen estas operaciones más eficientes.

Existen múltiples versiones de estos algoritmos, añadiendo cada uno de ellos características adicionales que permiten diagnosticar sistemas más complejos, por ejemplo, sistemas híbridos donde cada ecuación del modelo tiene una condición que indica cuándo se puede utilizar. En este trabajo hemos modificado las versiones existentes y aplicado las nuevas estructuras ordenadas para hacerlos más eficientes.

Por otro lado, esta herramienta ofrecía también una función para representar gráficamente los posibles conflictos en forma de hipergrafos dirigidos; ésta tenía problemas de gestión de memoria (se quedaba bloqueada en sistemas muy grandes) y la representación no era demasiado clara; por lo que se tuvo que realizar una nueva versión.

# Tabla de contenidos

# Índice de figuras

# 1

# Introducción. Contexto

## 1.1. Introducción

En el presente capítulo se dará una breve introducción teórica a este Trabajo de Fin de Grado (TFG), explicando su contexto y describiendo los conceptos básicos en los que se asienta.

Se describirán concretamente los conceptos de diagnóstico, diagnóstico basado en modelos y conflicto; así como los conjuntos de apoyo que serán usados para su obtención: cadena evaluable minimal y modelo evaluable minimal.

Se puede encontrar información más detallada a este respecto en los artículos escritos por los tutores de este proyecto [Pul97] y [Pul01].

Posteriormente, se hará una breve descripción de la aplicación existente para calcular el conjunto de Posibles Conflictos y que será el punto de partida sobre el que se asienta el trabajo de este TFG.

Por último, se describirán los objetivos concretos en que se centrará este TFG.

## 1.2. Conceptos

Las tareas de diagnóstico poseen una gran importancia en la vida diaria. Podemos encontrar multitud de ejemplos de dichas tareas en nuestro día a día: los indicadores de fallos asociados a los sensores de temperatura y nivel de aceite de nuestro automóvil, el indicador de batería de un teléfono móvil, etc.; aunque también podemos pensar en otros ejemplos de tareas de diagnóstico más complejas, como las que podríamos encontrar en hospitales, cadenas de montaje automatizadas, etc.

Puesto que cada vez los sistemas que se desarrollan son más complejos, surge la necesidad de automatizar este tipo de diagnósticos. Esto dará lugar a una mejora de

dichos sistemas, al facilitarse el control de errores en el proceso de fabricación y ayudar a evitar situaciones de riesgo provocadas por fallos en los procesos.

Según Balakrishnan y Honavar [Bal98], el diagnóstico es "la tarea que dado un conjunto de síntomas observados identifica una causa probable que pueda explicarlos". Basándose en esta definición, se establece una clasificación sobre los sistemas automáticos de diagnóstico que engloba las aproximaciones más utilizadas. Esta clasificación se obtiene a partir de la respuesta a las siguientes preguntas: ¿cómo conoce el sistema la relación entre los síntomas observados y su diagnóstico asociado?, ¿cómo representa el sistema esa relación? y ¿cómo se usa esta representación para diagnosticar fallos?

Según esta clasificación, los tipos de técnicas de diagnóstico serán las siguientes:

- *Técnicas basadas en conocimiento*: utilizan la experiencia anterior sobre el dominio a diagnosticar.
- *Técnicas basadas en casos*: usan un conjunto de casos para los que se conocen las soluciones.
- *Técnicas basadas en aprendizaje automático*: con dos enfoques distintos, uno acopla información nueva al conocimiento ya existente, mientras que el otro extrae conocimiento de la regularidad de los datos.
- ***Técnicas basadas en modelos***: razonan a partir de un modelo del sistema.

Esta última técnica es en la que nos vamos a centrar en el desarrollo de este TFG. En adelante nos referiremos a ella como Diagnóstico Basado en Modelos (DBM).

El DBM se basa en comparar varias observaciones sobre el funcionamiento de un dispositivo (datos reales) y las predicciones hechas por el modelo (datos que deberían producirse si el dispositivo funciona correctamente). Cuando ambos datos difieran podremos concluir que el dispositivo no funciona correctamente.

El DBM presenta las siguientes ventajas frente a las otras técnicas de diagnóstico:

- Dificultad para la adquisición y representación del conocimiento experto.
- Independencia del dispositivo a diagnosticar.
- Evita los problemas que genera la obtención de algún tipo de modelo de conocimiento a partir de un conjunto de datos.

Existen distintas formas de DBM, pero todas ellas comparten el método de razonar en función de la información que proporcionan los modelos del sistema. Esto se traduce en las siguientes ventajas:

- Utilizan exclusivamente conocimiento sobre la estructura y el funcionamiento de los componentes del sistema.
- Solucionan el problema de las variantes. Distintos sistemas se pueden diagnosticar usando los mismos modelos; tan sólo es necesario adaptar los

parámetros de los mismos, como por ejemplo, los distintos modelos de un motor de coche

- Poseen una metodología para la derivación automática de asociaciones entre síntomas y causas. Esto hace que no sea necesario incluir dicha asociación de forma explícita para realizar el diagnóstico.

Para describir el sistema podemos basarnos en su estructura o en su comportamiento. Podremos entonces elaborar un conjunto de modelos que definen el comportamiento de sus componentes y, a su vez, las conexiones entre dichos componentes. De hecho, la información con la que contamos para realizar las predicciones es:

- Una descripción del comportamiento correcto de cada componente.
- Una descripción de la estructura interna del sistema o dispositivo, y de cómo se conectan entre sí los componentes.
- Un conjunto de observaciones.

Este tipo de técnicas de DBM son especialmente indicadas cuando los modelos de los componentes del sistema son conocidos; tales como en circuitos electrónicos, motores, plantas industriales, etc.

Usando estos modelos y un conjunto de observaciones el problema del diagnóstico se resuelve cuando se identifica uno o más componentes tales que, poniendo de manifiesto su comportamiento incorrecto, hacen que el sistema como un todo se comporte incorrectamente.

Otra característica del DBM es que permite detectar fallos que no se hayan identificado con anterioridad, ya que no necesita conocer los síntomas asociados a comportamientos de fallo; sólo necesita conocer los modelos de sus componentes.

De entre todas las aproximaciones al DBM vamos a centrarnos en el Diagnóstico Basado en Consistencia. Éste consiste en realizar la diagnosis mediante un proceso iterativo de predicción del comportamiento, detección de discrepancias, generación de un conjunto de candidatos y refinamiento de dicho conjunto.

Por tanto podemos definir un conflicto como una discrepancia entre un valor observado y un valor predicho.

Al conjunto de componentes que han intervenido en la estimación de esa discrepancia o conflicto se les llamará conjunto conflicto.

Cuando los conjuntos conflictos no se calculan on-line si no que se analiza el modelo del sistema y se determinan off-line todos los conjuntos de componentes que pueden dar lugar a un conflicto hablamos de "Posibles Conflictos" [Pul01].

# 1.3. Cálculo de posibles conflictos

Centrándonos ya en el algoritmo con el que trabaja nuestra aplicación, vamos a pasar a explicar de forma breve el proceso de cálculo de posibles conflictos. Se puede encontrar una explicación más detallada en el capítulo 1.3 de la memoria del PFC de Rubén Lajo [Laj07].

Partiremos de la definición del sistema. De forma muy simplificada, podemos ver dicha definición como un conjunto de valores (variables) y de relaciones entre ellas (ecuaciones). Para cada ecuación tendremos, además, un conjunto de interpretaciones que nos indicarán qué variables pueden ser calculadas en caso de que conozcamos el valor del resto de variables de la ecuación.

Veamos esto con un sencillo ejemplo:

Tenemos tres variables: A, B y C relacionadas por una ecuación (f A B C). Para esa ecuación tenemos, además, dos interpretaciones:

- A=f_1(B,C)
- C=f_2(A,B)

Esto quiere decir que, conocidos los valores de B y C podemos obtener el valor de A y, conocidos los de A y B podemos obtener el de C; pero usando esta ecuación, no tenemos forma de conocer el valor de B.

Las variables podrán ser de dos tipos: observadas, cuando puede medirse su valor, y no observadas cuando no.

Partiendo de esta definición del sistema debemos buscar un conjunto de subsistemas sobredeterminados. Estos subsistemas, a los que llamaremos Cadenas Evaluables Minimales (CEMs) serán un conjunto de ecuaciones que cumplirán las siguientes condiciones:

- Formar un hipergrafo conexo, entendiendo que las ecuaciones son las relaciones o hiperarcos, y las variables son los nodos del hipergrafo.
- Tener algún valor observado
- Que se puedan propagar localmente los valores para hallar los no observados
- Que haya tantas ecuaciones como variables desconocidas + 1
- Puesto que son minimales, su hipergrafo no debe estar contenido en el hipergrafo de otra CEM ya hallada.

Una vez tenemos el Conjunto de Cadenas Evaluables Minimales (CCEM), para cada CEM debemos hallar todas las formas en las que puede evaluarse.

A las distintas formas en las que puede evaluarse una CEM las denominaremos Modelos Evaluables Minimales. Dichos modelos pueden ser representados mediante un grafo y/o que deberá cumplir las siguientes condiciones:

- Ha de tomar una y solo una de las posibles interpretaciones asociadas a cada hiperarco (ecuación) de la CEM.
- Ha de contener un solo 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 obtenidas a partir de las observaciones y propagando localmente.

Cada MEM puede llevar asociado 0 o más MEM.

Una vez hemos calculado el Conjunto de Modelos Evaluables Minimales (CMEM), el cálculo del Conjunto de Posibles Conflictos (CPC) es inmediato. Un Posible Conflicto (PC) será el conjunto de relaciones en una CEM que tiene, al menos, un MEM asociado; esto es, es un subsistema en el que podemos encontrar discrepancias entre los valores observados y los valores calculados.

En sistemas híbridos las ecuaciones de un modelo pueden ser válidas sólo 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. Sin embargo, si las condiciones se pueden escribir a partir de datos observados, se pueden simplificar enormemente.

## 1.4. Objetivos del TFG

En anteriores versiones de la aplicación de CPCs se desarrollaron varios algoritmos para realizar el cálculo anteriormente explicado. Dichos algoritmos tenían el inconveniente de que en sistemas grandes el tiempo de cálculo era excesivamente alto, debido a la gran cantidad de procesos de búsqueda en profundidad y anchura que se hacen en el hipergrafo. Lo que se persigue en este TFG es conseguir una mejora notable en estos tiempos mediante el uso de estructuras de datos ordenadas que mejoren la eficiencia de las operaciones de búsqueda e inserción de los algoritmos.

También se desarrollará otra nueva versión del visor para la representación de conflictos en forma de hipergrafo dirigido.

Estableceremos, entonces, los siguientes objetivos para este proyecto:

1) Analizar las versiones previas de la aplicación y los distintos algoritmos y buscar posibles formas de mejora.
2) Realizar una nueva versión del algoritmo de cálculo de posibles conflictos básico.

3) Realizar una nueva versión del algoritmo de cálculo de posibles conflictos con condiciones.
4) Mejorar la representación gráfica actual de los posibles conflictos.

## 1.5. Contenidos de la memoria

A lo largo de este documento trataremos de describir el proceso llevado a cabo para la realización de este TFG.

Para ello, dividiremos la memoria en tres grandes apartados, que se corresponderán con las grandes etapas de desarrollo de cualquier proyecto software:

- Plan de desarrollo software
- Especificación de requisitos
- Etapa de análisis
- Etapa de diseño
- Etapa de implementación

Añadiremos además un capítulo con las conclusiones obtenidas, así como una lista de las fuentes bibliográficas utilizadas y un anexo en el que se añadirá información adicional que puede resultar de interés para el lector.

# 2

# Plan de desarrollo software

## 2.1. Introducción

### *2.1.1. Propósito*

El propósito este Plan de desarrollo Software es ofrecer toda la información necesaria para controlar el desarrollo del proyecto "Revisión y adaptación de la herramienta para el cálculo de Posibles Conflictos". Nos ofrecerá una visón de alto nivel de abstracción que facilitará al equipo de trabajo la tarea de organizar el desarrollo.

Los usuarios potenciales de este apartado de la documentación serán:

- Jefe de proyecto: lo usará para realizar la planificación temporal y de recursos; así como para efectuar el seguimiento y la evaluación del proyecto.
- Miembros del equipo de proyecto: lo utilizarán entender qué deben hacer, cuándo deben hacerlo y qué otras actividades dependen de la 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 "Revisión y adaptación de la herramienta para el cálculo de Posibles Conflictos".

Los planes que se detallan en este capítulo se basan en la especificación de requisitos, que puede encontrarse en el *capítulo 3* del presente documento.

### *2.1.3. Definiciones, Acrónimos, y Abreviaturas*

Ver el glosario general del proyecto.

### *2.1.4 Referencias*

En el presente capítulo encontramos referencias a los siguientes apartados del documento:

- Introducción teórica
- Especificación de requisitos
- Glosario
- Referencias bibliográficas

### *2.1.5. Perspectiva general*

El Plan de Desarrollo Software contendrá la siguiente información:

- Perspectiva general del proyecto: da una descripción del propósito, alcance y objetivos del proyecto. También define los productos entregables que se prevén.
- Organización del proyecto: Describe la estructura organizativa del equipo de proyecto.
- Proceso de gestión: explica la estimación de costes en términos de tiempo y recursos, define las fases principales y los hitos intermedios para el proyecto y describe cómo se monitorizará el proyecto.

## 2.2. Perspectiva general del proyecto

### *2.2.1. Propósito, alcance y objetivos*

Como explicamos con más detalle en el *capítulo 1 (Introducción teórica)*, este proyecto trata de una revisión y mejora de proyectos anteriores [Laj07] y [San07] y, al igual que éstos, se enmarca dentro del Diagnóstico Basado en Modelos y más concretamente en el Diagnóstico Basado en Consistencia.

Principalmente, podemos definir dos grandes metas de este proyecto:

- Mejorar la eficiencia del cálculo de posibles conflictos mediante el uso de estructuras de datos ordenadas: dichas mejoras se implementarán sobre el algoritmo básico de CPCs desarrollado en el proyecto [Laj07] y el algoritmo de CPCs con condiciones desarrollado en el proyecto [San07].
- Implementación de un nuevo visor para la visualización de los resultados: el objetivo es desarrollar un nuevo visor completamente nuevo, que contenga toda la funcionalidad de las anteriores versiones; pero con un aspecto y una interfaz más cuidada y agradable para el usuario.

Los nuevos algoritmos se incluirán dentro de la aplicación desarrollada en el proyecto [Laj07] y cumplirán las especificaciones marcadas por la anterior aplicación para su ampliación como plug-in.

El nuevo visor deberá estar integrado también en la anterior aplicación; pero, además, deberá poder trabajar de forma independiente a la aplicación de cálculo.

### 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 deberá estar completamente finalizado antes de junio del 2014.
- Horario de trabajo: El tiempo el equipo dedicarán a la realización del proyecto estará considerablemente limitado por motivos personales/laborales. Se calcula que la dedicación a este 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 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. Éstos serán especificados en el apartado *2.4.2.5: Recursos del proyecto*

### 2.2.3. Entregables del proyecto

A continuación se indican cada uno de los artefactos que serán generados y utilizados por el proyecto y que constituyen los documentos que se irán presentando a lo largo de todo el proceso.

Típicamente, se generarán uno o más entregables al finalizar cada iteración del proceso de desarrollo. Al tratarse de un proceso iterativo e incremental, cada uno de esos entregables será susceptible de ser modificado en posteriores etapas, teniendo sólo las versiones definitivas de cada uno de ellos una vez el desarrollo haya finalizado completamente.

Pasamos a enumerar los artefactos a generar, agrupados por flujos de trabajo:

- Requisitos:
  - Introducción teórica
  - Especificación de requisitos
- Gestión del proyecto:
  - Plan de Desarrollo de Software
- Análisis y diseño:
  - Modelo de casos de uso

    - Especificación de casos de uso
    - Modelo de análisis
    - Modelo de diseño
    - Realización de casos de uso
- Implementación:
    - Modelo de implementación
    - Versión funcional del algoritmo básico de CPCs
    - Versión funcional del algoritmo de CPCs con condiciones
    - Versión funcional del nuevo visor de resultados
    - Código fuente
- Pruebas:
    - Casos de prueba
- Lanzamiento:
    - Manual de usuario
    - Manual de instalación
    - Producto final

La presente memoria está formada por una unión entre las versiones definitivas de cada uno de estos entregables.

### *2.2.4. Evolución del Plan de Desarrollo*

El Plan de Desarrollo Software se revisará periódicamente y se refinará antes del comienzo de cada iteración.

Los sucesivos planes de iteración servirán, además de para concretar las tareas correspondientes a cada una, para corregir cualquier desviación respecto al plan inicial.

## 2.3 Organización del proyecto

### *2.3.1. Estructura organizativa*

Se trata de un proyecto individual, de forma que los roles de Jefe de Proyecto, Analista, diseñador, Programador y Personal de pruebas recaerán sobre una sola persona.

### *2.3.2. Interfaces externas*

Se mantendrá un contacto periódico con los tutores del proyecto, que proporcionarán asesoramiento y guía en cada una de las etapas del desarrollo.

Además, se contará con la colaboración del alumno David Rubio, que se encuentra realizando paralelamente otro proyecto basado en la misma herramienta.

### 2.3.3. Roles y responsabilidades

Identifica las unidades de organización del proyecto, que se responsabilizarán de todas las obligaciones, flujo de trabajo, detalles y procesos de soporte que conlleva el proyecto.

| Rol | Responsabilidad |
|---|---|
| **Jefe de proyecto** | Se encargará de la gestión de los recursos y 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.<br>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** | Su función será generar 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

Dada la naturaleza académica de este proyecto y su limitación en cuanto a recursos humanos y materiales, este apartado nos centraremos en las estimaciones temporales, y los puntos en los que se volverá a revisar dichas estimaciones.

### 2.4.2. Plan de proyecto

#### 2.4.2.1. Plan de fases

Para el desarrollo del proyecto se empleará la metodología UPEDU [Upe14]. En la siguiente tabla se muestra la distribución de tiempos que se seguirá, así como el número de iteraciones de cada fase.

| Fase | Nº de iteraciones | Duración (semanas) |
|---|---|---|
| **Concepción** | 1 | 7 |
| **Elaboración** | 2 | 23 |
| **Construcción** | 2 | 25 |
| **Transición** | 1 | 2 |

#### *2.4.2.2. Objetivos de las iteraciones*

Los hitos a alcanzar a la finalización de cada fase o iteración quedan reflejados en la siguiente tabla:

| Iteración | Descripción | Hitos |
|---|---|---|
| **Concepción** | Definición de los objetivos y requisitos del proyecto y elaboración del plan de desarrollo. | • Especificación de requisitos<br>• Plan de Desarrollo |
| **Elaboración I** | Análisis de la aplicación. Elección de casos de uso y primera aproximación al diagrama de clases. | • Modelo y especificación de casos de uso<br>• Modelo de análisis |
| **Elaboración II** | Diseño de la aplicación. Creación del diagrama de clases definitivo, diagramas de secuencia, etc. | • Modelo de diseño<br>• Realización de casos de uso |
| **Construcción I** | Implementación de la aplicación. Versión funcional de algoritmos y visor. | • Modelo de implementación<br>• Versiones beta de la aplicación |
| **Construcción II** | Refinamiento del visor y corrección de errores. | • Versiones alfa de la aplicación |
| **Transición** | Desarrollo de la versión definitiva de la aplicación y construcción de manuales. | • Versión definitiva de la aplicación<br>• Manual de usuario<br>• Manual de instalación |

#### *2.4.2.3. Versiones*

Para los dos algoritmos que se van a revisar no tiene sentido hablar de versiones; puesto que su funcionalidad y aspecto ya están definidos; el cambio que se va a realizar consiste en modificar la forma en la que tratan los datos internamente.

Respecto al visor, podemos distinguir las siguientes versiones:

- Versión alfa: Esta versión será capaz de representar los resultados de forma correcta; pero sin hacer mucho hincapié en el aspecto de la representación.

- Versión beta: Partiendo de la versión anterior, se realizarán cambios en la interfaz para hacerla más atractiva y facilitar la visualización.
- Versión definitiva: se añaden las funcionalidades secundarias: representación de condiciones, opción de imprimir el resultado…

#### *2.4.2.4. Planificación temporal*

En la *figura 2.1* se muestra el diagrama de Gantt que manifiesta objetivos de fechas para la conclusión de iteraciones y fases e hitos.

#### *2.4.2.5. Recursos del proyecto*

Para la realización de este proyecto se cuenta con los siguientes recursos:

- Recursos humanos:
  - Un alumno de último curso de Grado de Ingeniería Informática
  - Dos tutores para realizar el seguimiento del proyecto
- Recursos hardware:
  - Un ordenador personal
  - Un ordenador portátil
- Recursos software:
  - Sistema operativo Windows 7 Professional
  - Microsoft Office 2010
  - StarUML
  - REM (Requeriments Manager)
  - GanttProject (Free project sheduling and management)
  - Dropbox
  - Java Runtime Environment
  - Eclipse

Todos estos programas son gratuitos, excepto el sistema operativo y la suite ofimática Office, de los cuales ya se poseía una licencia.

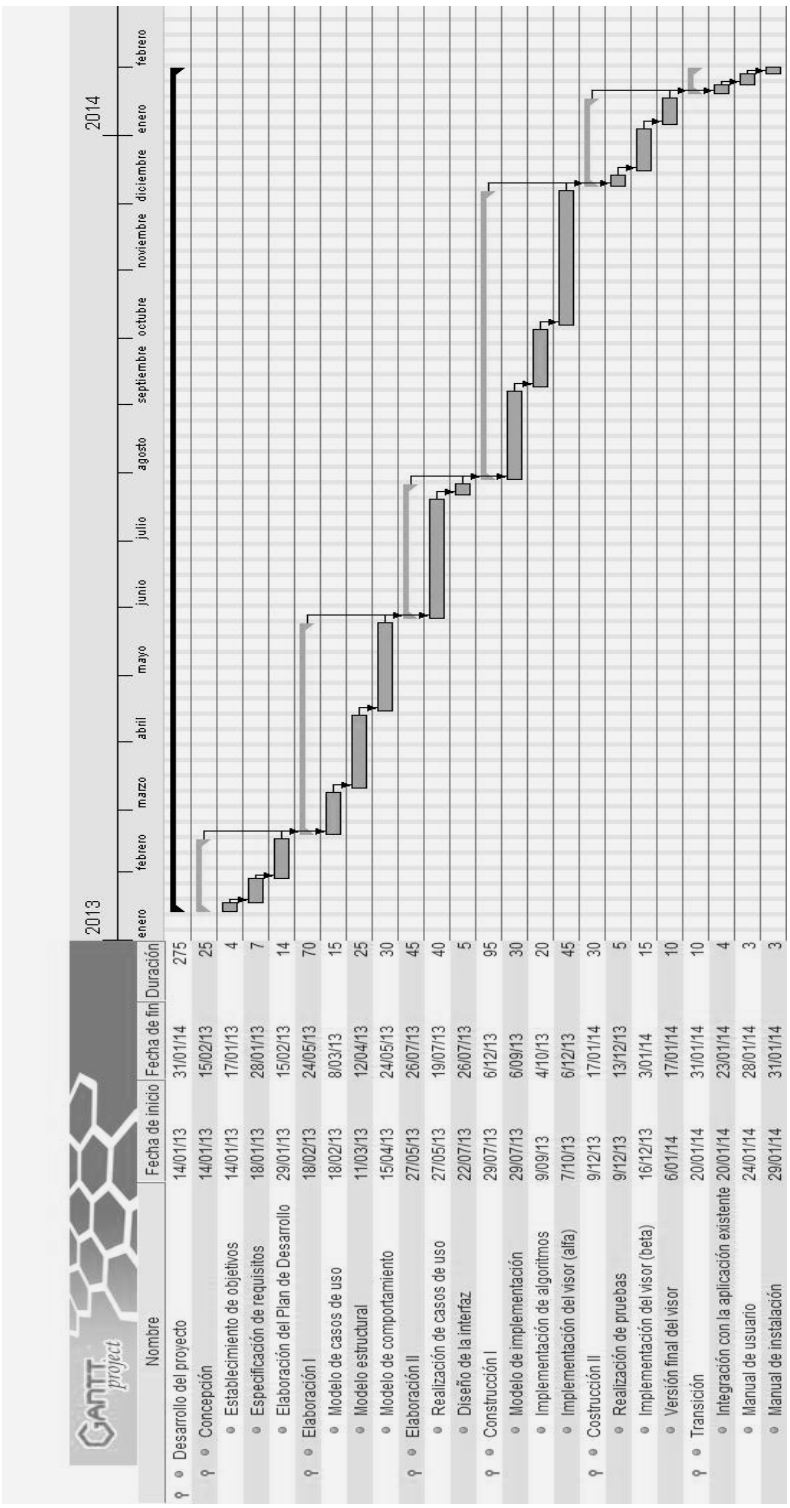
### *2.4.3. Monitorización y Control del proyecto*

#### *2.4.3.1. Gestión de requisitos*

Se puede encontrar toda la información relativa a la captura de requisitos en el *capítulo 3* del presente documento.

#### *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.




| Nombre | Fecha de inicio | Fecha de fin | Duración |
|---|---|---|---|
| Desarrollo del proyecto | 14/01/13 | 31/01/14 | 275 |
| Concepción | 14/01/13 | 15/02/13 | 25 |
| Establecimiento de objetivos | 14/01/13 | 17/01/13 | 4 |
| Especificación de requisitos | 18/01/13 | 28/01/13 | 7 |
| Elaboración del Plan de Desarrollo | 29/01/13 | 15/02/13 | 14 |
| Elaboración I | 18/02/13 | 24/05/13 | 70 |
| Modelo de casos de uso | 18/02/13 | 8/03/13 | 15 |
| Modelo estructural | 11/03/13 | 12/04/13 | 25 |
| Modelo de comportamiento | 15/04/13 | 24/05/13 | 30 |
| Elaboración II | 27/05/13 | 26/07/13 | 45 |
| Realización de casos de uso | 27/05/13 | 19/07/13 | 40 |
| Diseño de la interfaz | 22/07/13 | 26/07/13 | 5 |
| Construcción I | 29/07/13 | 6/12/13 | 95 |
| Modelo de implementación | 29/07/13 | 6/09/13 | 30 |
| Implementación de algoritmos | 9/09/13 | 4/10/13 | 20 |
| Implementación del visor (alfa) | 7/10/13 | 6/12/13 | 45 |
| Costrucción II | 9/12/13 | 17/01/14 | 30 |
| Realización de pruebas | 9/12/13 | 13/12/13 | 5 |
| Implementación del visor (beta) | 16/12/13 | 3/01/14 | 15 |
| Versión final del visor | 6/01/14 | 17/01/14 | 10 |
| Transición | 20/01/14 | 31/01/14 | 10 |
| Integración con la aplicación existente | 20/01/14 | 23/01/14 | 4 |
| Manual de usuario | 24/01/14 | 28/01/14 | 3 |
| Manual de instalación | 29/01/14 | 31/01/14 | 3 |

***Figura 2.1: Diagrama de Gantt***

#### *2.4.3.3. Gestión de riesgos*

En esta sección se analizarán los posibles riesgos que pueden poner en peligro la correcta consecución del proyecto. Se identifican en orden decreciente de prioridad los eventos que pueden ocasionar problemas durante el desarrollo, lo que servirá de referencia durante todo el proceso para detectar fuentes de problemas antes de que estos sucedan.

##### *2.4.3.3.1. Lista de riesgos*

Utilizaremos las siguientes definiciones y convenios para describir cada riesgo:

Magnitud: califica el nivel de riesgo de 1 a 10 (1: riesgo bajo; 10: riesgo alto) basándonos en la probabilidad de ocurrencia y en la criticidad del mismo. Ordena la lista de riesgos.

Descripción: breve frase que describe el posible problema.

Probabilidad: probabilidad aproximada de que ocurra el riesgo.

Impacto:

- C (Crítico): afecta a toda la funcionalidad del proyecto y la línea base.
- A (Alto): afecta a las necesidades del usuario y a la funcionalidad.
- M (Medio): se puede contener pero la mayoría de las veces es evitable.
- B (Bajo): se puede aceptar y convivir con él pero también mitigar rápidamente.

Indicador: métrica o detector del riesgo a monitorizar.

Plan de contingencia: estrategia para mitigar o evitar las consecuencias del riesgo

A continuación se mostrará la lista de posibles riesgos:

| **RSK-01** | **Retraso en la entrega** |
|---|---|
| **Magnitud** | 10 |
| **Descripción** | La entrega del proyecto será posterior a la fecha planificada. |
| **Probabilidad** | 50% |
| **Impacto** | C |
| **Indicador** | 10% de las actividades pendientes 1 mes antes de la fecha límite de entrega. |
| **Plan de contingencia** | Aumentar el número de horas que se dedican al proyecto. |

| RSK-02 | Falta de experiencia con la tecnología utilizada |
|---|---|
| **Magnitud** | 7 |
| **Descripción** | El equipo de programadores carece de la experiencia necesaria para llevar a cabo la implementación del proyecto con la tecnología elegida. |
| **Probabilidad** | 65% |
| **Impacto** | A |
| **Indicador** | Retrasos importantes en el proceso de implementación. Excesivo número de fallos detectados en la fase de pruebas. |
| **Plan de contingencia** | Formación previa del equipo de programadores en la tecnología elegida. |

| RSK-03 | Falta de experiencia en el proceso de planificación |
|---|---|
| **Magnitud** | 6 |
| **Descripción** | Debido a que el jefe de proyecto posee escasa experiencia en la planificación es posible que ésta no se realice de forma correcta, ocasionando los consiguientes problemas (retrasos, mala organización de las tareas…). |
| **Probabilidad** | 60% |
| **Impacto** | A |
| **Indicador** | Errores continuos en la planificación, mala estimación de los costes de cada iteración. |
| **Plan de contingencia** | Búsqueda exhaustiva de información. Comparación con otros proyectos de tamaño similar, consulta a expertos… |

| RSK-04 | Disminución del tiempo de dedicación del equipo de desarrollo |
|---|---|
| **Magnitud** | 6 |
| **Descripción** | Los miembros del equipo, por circunstancias ajenas a su control, ven limitado el tiempo que pueden dedicar al desarrollo del proyecto. |
| **Probabilidad** | 50% |
| **Impacto** | A |
| **Indicador** | Ralentización notable del avance en el desarrollo del proyecto. |
| **Plan de contingencia** | Disminución de los requisitos, retraso en la entrega del proyecto. |

| RSK-05 | Mala calidad de la interfaz gráfica de la aplicación |
|---|---|
| **Magnitud** | 4 |
| **Descripción** | Dificultad para acceder a las funciones de la aplicación más frecuentes; visualización pobre y dificultosa de los resultados obtenidos. |
| **Probabilidad** | 50% |
| **Impacto** | M |
| **Indicador** | Descontento del cliente con la interfaz mostrada. Peticiones constantes de cambios en la interfaz. |
| **Plan de contingencia** | Desarrollo de prototipos no funcionales de la interfaz previa construcción de ésta. |

| RSK-06 | Mala comprensión de las necesidades del cliente |
|---|---|
| **Magnitud** | 4 |
| **Descripción** | Requisitos mal planteados. Trabajo mal enfocado a satisfacer las necesidades del cliente. |
| **Probabilidad** | 15% |
| **Impacto** | C |
| **Indicador** | Cambios constantes en los requisitos. |
| **Plan de contingencia** | Realizar cuantas entrevistas con el cliente sean necesarias para tener totalmente claras y definidas las necesidades de éste. |

| RSK-07 | Cambio en los requisitos del sistema |
|---|---|
| **Magnitud** | 3 |
| **Descripción** | Aparición de nuevas necesidades del cliente en un estado avanzado del proceso de desarrollo. |
| **Probabilidad** | 5% |
| **Impacto** | C |
| **Indicador** | Inclusión de nuevos requisitos en fases avanzadas del desarrollo. Pérdida de trabajo ya realizado y necesidad de realizar más trabajo del inicialmente planificado. |
| **Plan de contingencia** | Hacer ver al cliente las consecuencias de modificar los requisitos una vez ha comenzado el proceso de desarrollo. |

# 3

# Captura de requisitos

## 3.1. Introducción

En este capítulo se pretenden explicar 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 y la satisfacción del cliente.

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*

El presente capítulo da una descripción de todos los requisitos de software para el proyecto "Revisión y adaptación de la herramienta para el cálculo de Posibles Conflictos".

### *3.1.3. Definiciones, Acrónimos, y Abreviaturas*

Ver el glosario general del proyecto.

### *3.1.4 Referencias*

- Glosario
- Bibliografía

### *3.1.5. Perspectiva general*

El capítulo de especificación de requisitos contendrá la siguiente información:

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

## 3.2. Objetivos

A continuación se presentará una lista con los objetivos principales que se persigue lograr en este proyecto:

| **OBJ-01** | **Análisis del sistema existente** |
|---|---|
| **Descripción** | Se realizará un análisis profundo de las anteriores versiones de la aplicación y los algoritmos, buscando una forma de mejorar la eficiencia de estos. |
| **Importancia** | Alta |
| **Comentarios** | - |

| **OBJ -02** | **Nueva versión del algoritmo básico de CPCs** |
|---|---|
| **Descripción** | Se implementará una nueva versión del algoritmo de CPCs. Esta versión deberá permitir una mejora significativa en el tiempo de cálculo respecto a las versiones anteriores. |
| **Importancia** | Alta |
| **Comentarios** | |

| OBJ -03 | Nueva versión del algoritmo de CPCs con condiciones |
|---|---|
| **Descripción** | Se implementará una nueva versión del algoritmo de CPCs con condiciones. Esta versión deberá permitir una mejora significativa en el tiempo de cálculo respecto a las versiones anteriores. Deberá permitir, además, leer las condiciones desde un archivo de entrada e incluirlas en el archivo de salida de resultados. |
| **Importancia** | Alta |
| **Comentarios** | |

| OBJ -04 | Nueva aplicación Visor |
|---|---|
| **Descripción** | Se implementará una nueva versión de la aplicación de visualización de resultados. Ésta deberá ser más atractiva visualmente que sus predecesoras y deberá ser capaz de funcionar de forma independiente a la aplicación de cálculo. |
| **Importancia** | Alta |
| **Comentarios** | |

## 3.3. Requisitos funcionales

Esta es la lista de requisitos funcionales de la aplicación. Definen cómo debe comportarse el sistema; qué debe ser capaz de hacer.

| RF-01 | Cálculo de Posibles Conflictos |
|---|---|
| **Descripción** | El sistema deberá ser capaz de calcular de forma correcta los posibles conflictos usando el nuevo algoritmo mejorado. |
| **Importancia** | Vital |
| **Comentarios** | |

| RF-02 | Cálculo de posibles conflictos con condiciones |
|---|---|
| **Descripción** | El sistema deberá ser capaz de calcular de forma correcta los posibles conflictos teniendo en cuenta condiciones usando el nuevo algoritmo mejorado. |
| **Importancia** | Vital |
| **Comentarios** | |

| RF-03 | Visor de resultados |
|---|---|
| **Descripción** | El sistema deberá ofrecer una representación gráfica de los resultados obtenidos, representando los modelos evaluables minimales como hipergrafos o árboles y/o dirigidos. |
| **Importancia** | Vital |
| **Comentarios** | |

| RF-04 | Archivos de Entrada/Salida |
|---|---|
| **Descripción** | El sistema deberá ser capaz de leer los sistemas de entrada de un archivo de entrada, en formato XML ó texto plano. Así mismo, también deberá poder producir como salida otro archivo en formato XML ó texto plano (a elección del usuario) que contenga los resultados del cálculo y que sea compatible con otras aplicaciones de diagnóstico del grupo GSI. |
| **Importancia** | Vital |
| **Comentarios** | |

| RF-05 | Condiciones en los archivos de Entrada/Salida |
|---|---|
| **Descripción** | El sistema deberá ser capaz de leer las condiciones del fichero de descripción de sistema, ya esté en formato XML ó texto plano; así como escribir éstas en el fichero de resultados. |
| **Importancia** | Alta |
| **Comentarios** | |

| RF-06 | Impresión de resultados |
|---|---|
| **Descripción** | El sistema deberá ofrecer una opción que permita al usuario imprimir los resultados tal y como se muestran con el visor. |
| **Importancia** | Baja |
| **Comentarios** | |

| RF-07 | Multilenguaje |
|---|---|
| **Descripción** | El sistema deberá poseer una interfaz completamente traducida a español e inglés; pudiendo modificarse el idioma en cualquier momento. Además, se debe permitir al usuario agregar nuevos idiomas. |
| **Importancia** | Media |
| **Comentarios** | |

| RF-08 | Autonomía del visor |
|---|---|
| **Descripción** | El sistema deberá ser capaz de mostrar la representación gráfica de un sistema de forma completamente independiente a su cálculo. La herramienta del visor debe ser capaz de funcionar de forma completamente independiente de la herramienta de cálculo. |
| **Importancia** | Alta |
| **Comentarios** | |

| RF-09 | Representación de condiciones en el visor |
|---|---|
| **Descripción** | El sistema deberá representar gráficamente las condiciones en el visor. |
| **Importancia** | Alta |
| **Comentarios** | |

# 3.4. Requisitos de interfaz externo

Estos requisitos se refieren a la relación de la aplicación a desarrollar con su entorno.

## *3.4.1 Interfaces de usuario*

| IU-01 | Interfaz gráfica del visor |
|---|---|
| **Descripción** | El sistema deberá mostrar el resultado del cálculo de posibles conflictos mediante una representación gráfica consistente en un hipergrafo conexo. Esta representación deberá, además, incluir información sobre las condiciones y a qué interpretaciones afecta cada una de ellas. |
| **Importancia** | Vital |
| **Comentarios** | |

## *3.4.2 Interfaces software*

| IS-01 | Formato de las descripciones de sistemas |
|---|---|
| **Descripción** | El sistema ha de ser capaz de cargar descripciones de sistemas en dos formatos: texto plano y descripciones estructuradas en formato XML. |
| **Importancia** | Vital |
| **Comentarios** | |

| IS-02 | Formato del fichero de resultados |
|---|---|
| **Descripción** | El sistema deberá ser capaz de producir ficheros de resultados en formato texto plano y XML. El fichero de resultado en formato XML deberá ser compatible con el visor. |
| **Importancia** | Vital |
| **Comentarios** | |

## 3.5. Requisitos de información

| RI-01 | Información de condiciones |
|---|---|
| **Descripción** | El sistema deberá ser capaz de leer información sobre las condiciones de los ficheros de entrada en sus dos formatos posibles; así como escribir información sobre éstas en el fichero de resultados. |
| **Importancia** | Alta |
| **Comentarios** | |

## 3.6. Restricciones de diseño

Ahora pasamos a listar las restricciones impuestas en el diseño de la aplicación.

| RCE-01 | Estándar XML |
|---|---|
| **Descripción** | El sistema deberá manejar archivos de entrada/salida acordes con el estándar XML. |
| **Importancia** | Vital |
| **Comentarios** | |

## 3.7. Requisitos no funcionales

Por último, enumeraremos los requisitos no funcionales. Éstos se refieren a atributos de calidad, no a comportamientos específicos; por ejemplo: robustez, eficiencia, fiabilidad…

| RNF-01 | Lenguaje de programación |
|---|---|
| **Descripción** | El sistema deberá implementarse en lenguaje Java. El principal motivo es que las anteriores plataformas fueron implementadas en este lenguaje. Además, esto permitirá a la herramienta funcionar en varios sistemas operativos sin un esfuerzo de programación adicional. |
| **Importancia** | Alta |
| **Comentarios** | |

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

| RNF-03 | Eficiencia de los algoritmos de cálculo |
|---|---|
| **Descripción** | El sistema deberá realizar los cálculos de una forma significativamente más eficiente que con los algoritmos de las versiones anteriores, tanto en el factor tiempo de cálculo como en los recursos (memoria) necesarios. |
| **Importancia** | Alta |
| **Comentarios** | |

| RNF-04 | Fiabilidad de los algoritmos de cálculo |
|---|---|
| **Descripción** | El sistema deberá realizar los cálculos de forma correcta. Deberá responder de forma robusta ante situaciones de error inesperadas. |
| **Importancia** | Alta |
| **Comentarios** | |

# 4

# Modelo de análisis

## 4.1. Introducción

### *4.1.1. Propósito*

El propósito del modelo de análisis es dar una visión de la arquitectura del sistema a muy alto nivel.

Será la base en la que se apoyará el diseño y la posterior implementación del sistema,

Los usuarios potenciales de este apartado de documentación serán los analistas, ya que éstos serán los encargados de construirlo; y los diseñadores, pues será el medio que les permitirá realizar el diseño de la aplicación.

### *4.1.2. Alcance*

Este modelo de análisis proporcionará las pautas a más alto nivel para la realización del proyecto "Revisión y adaptación de la herramienta para el cálculo de Posibles Conflictos".

### *4.1.3. Definiciones, Acrónimos, y Abreviaturas*

Ver el glosario general del proyecto.

### *4.1.4. Referencias*

En el presente capítulo encontramos referencias a los siguientes apartados del documento:

- Especificación de requisitos
- Glosario
- Referencias bibliográficas

### *4.1.5. Perspectiva general*

Dentro de este capítulo podremos encontrar lo siguiente:

- Modelo de casos de uso: nos darán información sobre qué operaciones puede llevar a cabo el sistema y el conjunto de pasos de las que se compone cada una de ellas.
- Modelo estructural: consiste en una primera aproximación a la estructura de clases, el almacenamiento de la información y métodos que va a tener nuestro sistema.
- Modelo de comportamiento: nos mostrará, en forma de diagrama, el ciclo básico de funcionamiento de la aplicación.
- Modelo de interacción: describirá la interacción entre los elementos del sistema en forma de diagramas de secuencia.

## 4.2. Modelo de casos de uso

En este apartado explicaremos las operaciones que es capaz de realizar nuestro sistema y de qué forma las llevará a cabo.

### *4.2.1. Casos de uso de la aplicación de CPCs*

En la *Figura 4.1* podemos ver un diagrama de los casos de uso de la aplicación de CPCs. Esta aplicación fue desarrollada en el proyecto de Rubén Lajo [Laj07] y posteriormente modificada por David Rubio en el transcurso de sus prácticas en empresa para el Grupo GSI durante el curso 2012-2013. En el desarrollo del presente proyecto, sólo se verán afectados los casos que vemos sombreados. Para una descripción detallada del resto de casos remitimos al lector a la memoria del mencionado proyecto.

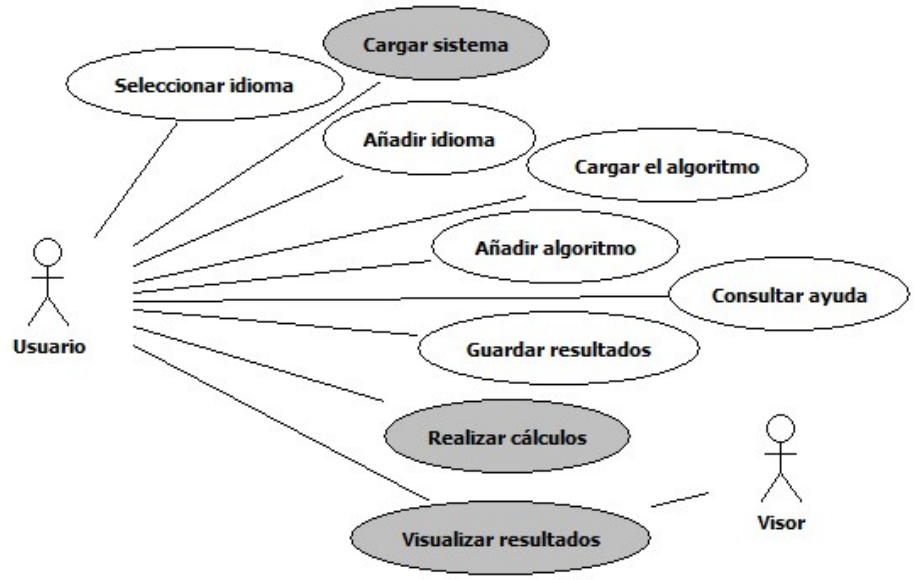



***Figura 4.1: Casos de uso de la aplicación de CPCs***

A continuación daremos una breve descripción de los casos de uso que no se ven afectados en el desarrollo de este proyecto. Posteriormente, veremos en más detalle los que sí se ven afectados:

- ***Seleccionar idioma***: permite al usuario modificar el idioma de la interfaz.
- ***Añadir idioma***: permite al usuario incluir un nuevo idioma para la presentación de la interfaz.
- ***Cargar algoritmo***: permite al usuario seleccionar el algoritmo que se utilizará para realizar los cálculos.
- ***Añadir algoritmo***: permite al usuario añadir un nuevo algoritmo de cálculo a la aplicación.
- ***Consultar ayuda***: permite al usuario consultar los documentos de ayuda de la aplicación.
- ***Guardar resultados***: permite al usuario almacenar en un fichero los resultados de los cálculos realizados.

Estos son los casos de uso que sí se ven afectados en este proyecto:

| **CU-01** | **Cargar sistema** | |
|---|---|---|
| **Descripción** | El sistema deberá comportarse de la siguiente forma cuando el usuario desee cargar la descripción de un sistema mediante un fichero de entrada. | |
| **Precondiciones** | • El fichero de entrada debe ser un fichero de texto plano o en formato XML y debe contener una descripción correcta de un sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario selecciona la opción de "Cargar sistema". |
| | **2** | El sistema solicita al usuario que escoja el fichero de entrada. |
| | **3** | El usuario selecciona el fichero de entrada en el que se encuentra la descripción del sistema que desea cargar. |
| | **4** | El sistema lee el fichero, carga los datos (incluyendo las condiciones, en caso de que éstas existan) y presenta al usuario una representación esquemática del sistema cargado. |
| **Postcondiciones** | El sistema estará cargado y el usuario podrá ver el contenido de éste de una forma cómoda. | |
| **Excepciones** | **Paso** | **Acción** |
| | **4** | En caso de que el archivo elegido no pueda ser abierto para lectura (ya sea por problemas de permisos, inexistencia de éste, etc.) deberá notificárselo al usuario y devolver a éste el control de la aplicación. |
| | **4** | En caso de que el archivo elegido no contenga una descripción de sistema correcta el sistema deberá notificárselo al usuario y devolver a éste el control de la aplicación. |
| **Rendimiento** | La carga del sistema deberá realizarse de forma rápida. Para sistemas de tamaño pequeño/medio el tiempo de carga debería ser inapreciable. | |
| **Importancia** | Alta | |
| **Comentarios** | - | |

| CU-02 | Realizar cálculos | |
|---|---|---|
| **Descripción** | El sistema deberá comportarse de la siguiente forma cuando el usuario desee realizar el cálculo de posibles conflictos. | |
| **Precondiciones** | • Debe haberse cargado un sistema de forma correcta.<br>• Debe haberse cargado un algoritmo de forma correcta. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario solicitará al sistema realizar los cálculos pertinentes. |
| | **2** | El sistema realizará los cálculos de acuerdo al algoritmo que se encuentre cargado y presentará los resultados al usuario. |
| **Postcondiciones** | Los cálculos se habrán realizado. Los resultados obtenidos podrán verse en pantalla y estarán listos para ser almacenados en un fichero de salida. | |
| **Excepciones** | **Paso** | **Acción** |
| | **-** | - |
| **Rendimiento** | Las operaciones a realizar, por su propia naturaleza, consumirán bastante tiempo; sobre todo con sistemas grandes. Es, por tanto, de vital importancia reducir el tiempo de dichas operaciones en la medida de lo posible. Se debe vigilar también el consumo de memoria; puesto que este también tiende a ser elevado. | |
| **Importancia** | Alta | |
| **Comentarios** | Este caso de uso incluye otros casos que dependerán del algoritmo cargado y serán explicados en los siguientes apartados. | |

| **CU-03** | **Visualizar resultados** | |
|---|---|---|
| **Descripción** | El sistema deberá comportarse de la siguiente forma cuando el usuario desee obtener una representación gráfica de los posibles conflictos. | |
| **Precondiciones** | • Deben haberse guardado previamente los resultados en formato XML. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario solicitará al sistema la visualización gráfica de los resultados. Podrá elegir si desea visualizarlos usando el antiguo visor o el nuevo. |
| | **2** | El sistema mostrará los resultados en forma de hipergrafo dirigido y conexo usando el visor seleccionado. |
| **Postcondiciones** | Podrá observarse en pantalla una representación gráfica de los resultados. | |
| **Excepciones** | **Paso** | **Acción** |
| | **2** | En caso de que el visor seleccionado sea el antiguo, si no se han almacenado los resultados en formato XML, se mostrará un mensaje de error y se devolverá el control al usuario. |
| | **2** | En caso de que el visor seleccionado sea el nuevo, si no se han almacenado los resultados en formato XML, se abrirá la herramienta del visor sin ningún fichero de resultados cargado. |
| | **2** | En ambos casos, si el fichero de resultados está mal formado, se notificará del error y se devolverá el control al usuario. |
| **Rendimiento** | La operación de visualizar resultados deberá ser prácticamente inmediata. | |
| **Importancia** | Alta | |
| **Comentarios** | Una vez abierto el visor, se podrán realizar diversos tipos de operaciones. En siguientes apartados veremos los casos de uso asociados a la herramienta del visor. | |

### 4.2.2. Casos de uso del algoritmo básico de CPCs

Ahora pasaremos a describir los casos de uso del algoritmo básico de CPCs que se desarrollará en este proyecto.

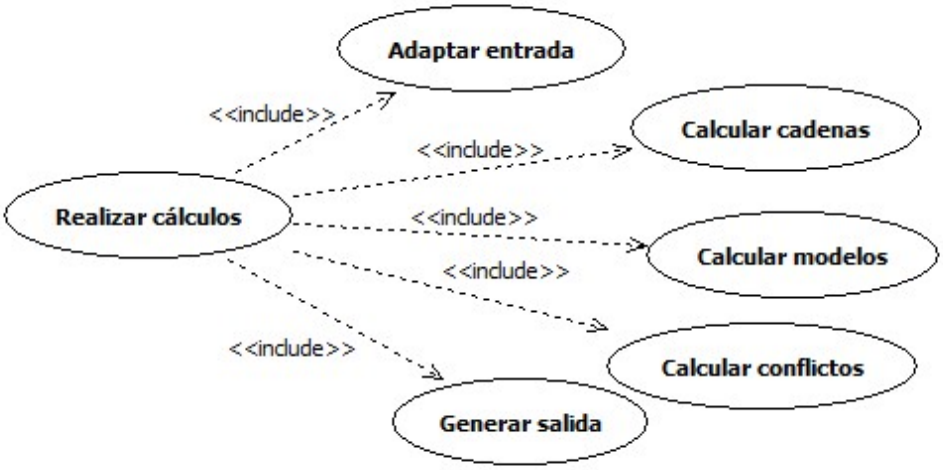



***Figura 4.2: Casos de uso del algoritmo básico de CPCs***

| **CU-04** | **Adaptar entrada** | |
|---|---|---|
| **Descripción** | Partiendo de la descripción del sistema, la aplicación realizará lo siguiente para adaptar los datos de forma que éstos puedan usarse para realizar futuros cálculos de acuerdo con el algoritmo seleccionado.<br>Se intentará, siempre que sea posible, organizar los datos en estructuras ordenadas que hagan más eficientes las operaciones de búsqueda, inserción y borrado. | |
| **Precondiciones** | • El sistema ha de haber sido cargado correctamente con anterioridad. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario selecciona cargar el algoritmo de cálculo de posibles conflictos básico. |
| | **2** | El sistema realiza las acciones pertinentes para posibilitar futuros cálculos sobre el sistema cargado; poniendo especial énfasis en la eficiencia de las estructuras utilizadas. |
| **Postcondiciones** | Los datos estarán listos para realizar posteriores cálculos de la forma más eficiente posible. | |

| **Excepciones** | **Paso** | **Acción** |
|---|---|---|
| | **2** | En caso de que no haya ningún sistema cargado, se solicitará al usuario que elija el fichero en el que se encuentra la descripción del sistema que se desea utilizar. |
| | **2** | En caso de que haya algún error en la estructura del sistema que impida adaptar sus datos se informará al usuario y se devolverá a éste el control de la aplicación. |
| **Rendimiento** | La operación de adaptación de la entrada deberá ser prácticamente inmediata. | |
| **Importancia** | Alta | |
| **Comentarios** | - | |

| **CU-05** | **Calcular cadenas** | |
|---|---|---|
| **Descripción** | El sistema debe comportarse de la siguiente forma cuando el usuario desee calcular el conjunto de cadenas evaluables minimales usando el algoritmo básico de cálculo de posibles conflictos. | |
| **Precondiciones** | • Debe haberse cargado un sistema de forma correcta.<br>• Los datos de dicho sistema deben haber sido adecuadamente adaptados para realizar con ellos los cálculos pertinentes. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario solicita al sistema el cálculo de las cadenas evaluables minimales. |
| | **2** | El sistema realiza los cálculos necesarios mostrando al usuario un listado de las cadenas evaluables minimales obtenidas para el sistema actual. |
| **Postcondiciones** | Realizado este caso, tendremos disponible información sobre las cadenas evaluables minimales del sistema. Esto nos permitirá continuar con el siguiente paso del cálculo de posibles conflictos. | |
| **Excepciones** | **Paso** | **Acción** |
| | **-** | - |
| **Rendimiento** | De todos los pasos del cálculo de posibles conflictos, éste es el que más tiempo consume. Se deberá de poner, por tanto, una atención especial en controlar su rendimiento en cuanto al tiempo de cómputo. | |
| **Importancia** | Alta | |
| **Comentarios** | - | |

| CU-06 | Calcular modelos | |
|---|---|---|
| **Descripción** | El sistema debe comportarse de la siguiente forma cuando el usuario desee calcular el conjunto de modelos evaluables minimales usando el algoritmo básico de cálculo de posibles conflictos. | |
| **Precondiciones** | • Debe haberse completado de forma satisfactoria el caso de uso de calcular cadenas. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario solicita al sistema el cálculo de los modelos evaluables minimales. |
| | **2** | El sistema realiza los cálculos necesarios mostrando al usuario un listado de los modelos evaluables minimales obtenidos para el sistema actual. |
| **Postcondiciones** | Realizado este caso, tendremos disponible información sobre los modelos evaluables minimales del sistema. Esto nos permitirá realizar el paso final para obtener los posibles conflictos | |
| **Excepciones** | **Paso** | **Acción** |
| | **-** | - |
| **Rendimiento** | El conjunto de operaciones para obtener los modelos evaluables minimales es considerablemente inferior al necesario para calcular las cadenas. Aun así, las operaciones llevarán también una cantidad de tiempo significativo; por lo que el control del rendimiento en cuanto al tiempo de cómputo será también de gran importancia. | |
| **Importancia** | Alta | |
| **Comentarios** | - | |

| CU-07 | Calcular conflictos | |
|---|---|---|
| **Descripción** | El sistema debe comportarse de la siguiente forma cuando el usuario desee calcular el conjunto de posibles conflictos usando el algoritmo básico de cálculo de posibles conflictos. | |
| **Precondiciones** | • Debe haberse completado de forma satisfactoria el caso de uso de calcular modelos. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario solicita al sistema el cálculo de los posibles conflictos. |
| | **2** | El sistema realiza los cálculos necesarios mostrando al usuario un listado de los posibles conflictos obtenidos para el sistema actual. |
| **Postcondiciones** | | |

| | | |
|---|---|---|
| **Excepciones** | **Paso** | **Acción** |
| | **2** | En caso de que no se hayan realizado los dos pasos del cálculo anteriores (cálculo de cadenas y cálculo de modelos), éstos se realizarán de forma consecutiva, mostrando al usuario el resultado final del cálculo. |
| **Rendimiento** | Realizado este caso, tendremos disponible información sobre los posibles conflictos del sistema. Esta información estará disponible para ser guardada en un fichero de salida. | |
| **Importancia** | | |
| **Comentarios** | | |

| | | |
|---|---|---|
| **CU-08** | **Generar salida** | |
| **Descripción** | El sistema debe comportarse de la siguiente forma cuando el usuario desee almacenar los resultados de los cálculos realizados en un fichero de salida. | |
| **Precondiciones** | • Debe haberse completado de forma satisfactoria el caso de uso de calcular conflictos. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario solicita al sistema almacenar los resultados obtenidos. |
| | **2** | El sistema pregunta al usuario qué formato desea usar para almacenar los resultados (XML o texto plano). |
| | **3** | El usuario confirma el tipo de fichero que desea generar. |
| | **4** | El sistema solicita al usuario que elija dónde se almacenarán los resultados. |
| | **5** | El usuario especifica la ubicación y el nombre del archivo en el que se almacenarán los resultados. |
| | **6** | El sistema almacena los resultados en la ubicación y con el formato especificados. |
| **Postcondiciones** | Al finalizar este caso de uso habremos generado un fichero con los resultados de los cálculos realizados. | |
| **Excepciones** | **Paso** | **Acción** |
| | **6** | En caso de que el destino elegido ya exista se preguntará al usuario si desea o no sobreescribirlo. |
| | **6** | En caso de que la ubicación no pueda ser accedida (por ejemplo, por un problema de permisos) se notificará del error y se devolverá el control al usuario. |
| **Rendimiento** | Esta operación deberá realizarse de forma prácticamente inmediata. | |
| **Importancia** | Alta | |
| **Comentarios** | - | |

### 4.2.3. Casos de uso del algoritmo CPCs con condiciones

A continuación, describiremos los casos de uso del algoritmo de CPCs con condiciones que se desarrollará en este proyecto.

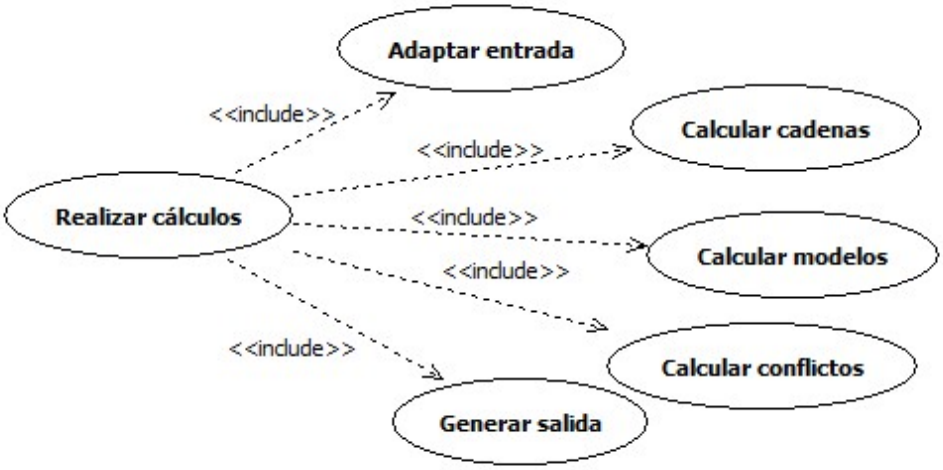



***Figura 4.3: Casos de uso del algoritmo de CPCs con condiciones***

| **CU-09** | **Adaptar entrada** | |
|---|---|---|
| **Descripción** | Partiendo de la descripción del sistema, la aplicación realizará lo siguiente para adaptar los datos de forma que éstos puedan usarse para realizar futuros cálculos de acuerdo con el algoritmo seleccionado.<br>Se intentará, siempre que sea posible, organizar los datos en estructuras ordenadas que hagan más eficientes las operaciones de búsqueda, inserción y borrado.<br>La descripción del sistema podrá contener condiciones, que deberán ser procesadas al igual que el resto de los datos. | |
| **Precondiciones** | • El sistema ha de haber sido cargado correctamente con anterioridad. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario selecciona cargar el algoritmo de cálculo de posibles conflictos con condiciones. |
| | **2** | El sistema realiza las acciones pertinentes para posibilitar futuros cálculos sobre el sistema cargado; poniendo especial énfasis en la eficiencia de las estructuras utilizadas. |

| | | |
|---|---|---|
| **Postcondiciones** | Los datos estarán listos para realizar posteriores cálculos de la forma más eficiente posible. | |
| **Excepciones** | **Paso** | **Acción** |
| | **2** | En caso de que no haya ningún sistema cargado, se solicitará al usuario que elija el fichero en el que se encuentra la descripción del sistema que se desea utilizar. |
| | **2** | En caso de que haya algún error en la estructura del sistema que impida adaptar sus datos se informará al usuario y se devolverá a éste el control de la aplicación. |
| **Rendimiento** | La operación de adaptación de la entrada deberá ser prácticamente inmediata. | |
| **Importancia** | Alta | |
| **Comentarios** | - | |

| **CU-10** | **Calcular cadenas** | |
|---|---|---|
| **Descripción** | El sistema debe comportarse de la siguiente forma cuando el usuario desee calcular el conjunto de cadenas evaluables minimales usando el algoritmo de cálculo de posibles conflictos con condiciones. | |
| **Precondiciones** | • Debe haberse cargado un sistema de forma correcta.<br>• Los datos de dicho sistema deben haber sido adecuadamente adaptados para realizar con ellos los cálculos pertinentes. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario solicita al sistema el cálculo de las cadenas evaluables minimales. |
| | **2** | El sistema realiza los cálculos necesarios mostrando al usuario un listado de las cadenas evaluables minimales obtenidas para el sistema actual, teniendo siempre en cuenta el valor de las condiciones. |
| **Postcondiciones** | Realizado este caso, tendremos disponible información sobre las cadenas evaluables minimales del sistema. Esto nos permitirá continuar con el siguiente paso del cálculo de posibles conflictos. | |
| **Excepciones** | **Paso** | **Acción** |
| | **-** | - |
| **Rendimiento** | De todos los pasos del cálculo de posibles conflictos, éste es el que más tiempo consume. Se deberá de poner, por tanto, una atención especial en controlar su rendimiento en cuanto al tiempo de cómputo. | |
| **Importancia** | Alta | |
| **Comentarios** | - | |

| CU-11 | Calcular modelos | |
|---|---|---|
| **Descripción** | El sistema debe comportarse de la siguiente forma cuando el usuario desee calcular el conjunto de modelos evaluables minimales usando el algoritmo de cálculo de posibles conflictos con condiciones. | |
| **Precondiciones** | • Debe haberse completado de forma satisfactoria el caso de uso de calcular cadenas. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario solicita al sistema el cálculo de los modelos evaluables minimales. |
| | **2** | El sistema realiza los cálculos necesarios mostrando al usuario un listado de los modelos evaluables minimales obtenidos para el sistema actual, teniendo siempre en cuenta el valor de las condiciones. |
| **Postcondiciones** | Realizado este caso, tendremos disponible información sobre los modelos evaluables minimales del sistema. Esto nos permitirá realizar el paso final para obtener los posibles conflictos | |
| **Excepciones** | **Paso** | **Acción** |
| | **-** | - |
| **Rendimiento** | El conjunto de operaciones para obtener los modelos evaluables minimales es considerablemente inferior al necesario para calcular las cadenas. Aun así, las operaciones llevarán también una cantidad de tiempo significativo; por lo que el control del rendimiento en cuanto al tiempo de cómputo será también de gran importancia. | |
| **Importancia** | Alta | |
| **Comentarios** | - | |

| CU-12 | Calcular conflictos | |
|---|---|---|
| **Descripción** | El sistema debe comportarse de la siguiente forma cuando el usuario desee calcular el conjunto de posibles conflictos usando el algoritmo de cálculo de posibles conflictos con condiciones. | |
| **Precondiciones** | • Debe haberse completado de forma satisfactoria el caso de uso de calcular modelos. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario solicita al sistema el cálculo de los posibles conflictos. |
| | **2** | El sistema realiza los cálculos necesarios mostrando al usuario un listado de los posibles conflictos obtenidos para el sistema actual. |

| | | |
|---|---|---|
| **Postcondiciones** | | |
| **Excepciones** | **Paso** | **Acción** |
| | **2** | En caso de que no se hayan realizado los dos pasos del cálculo anteriores (cálculo de cadenas y cálculo de modelos), éstos se realizarán de forma consecutiva, mostrando al usuario el resultado final del cálculo. |
| **Rendimiento** | Realizado este caso, tendremos disponible información sobre los posibles conflictos del sistema. Esta información estará disponible para ser guardada en un fichero de salida. | |
| **Importancia** | | |
| **Comentarios** | | |

| **CU-13** | **Generar salida** | |
|---|---|---|
| **Descripción** | El sistema debe comportarse de la siguiente forma cuando el usuario desee almacenar los resultados de los cálculos realizados en un fichero de salida. | |
| **Precondiciones** | • Debe haberse completado de forma satisfactoria el caso de uso de calcular conflictos. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario solicita al sistema almacenar los resultados obtenidos. |
| | **2** | El sistema pregunta al usuario qué formato desea usar para almacenar los resultados (XML o texto plano). |
| | **3** | El usuario confirma el tipo de fichero que desea generar. |
| | **4** | El sistema solicita al usuario que elija dónde se almacenarán los resultados. |
| | **5** | El usuario especifica la ubicación y el nombre del archivo en el que se almacenarán los resultados. |
| | **6** | El sistema almacena los resultados en la ubicación y con el formato especificados.<br>Dicho fichero incluirá información sobre las condiciones. |
| **Postcondiciones** | Al finalizar este caso de uso habremos generado un fichero con los resultados de los cálculos realizados. | |

| Excepciones | Paso | Acción |
|---|---|---|
| | 6 | En caso de que el destino elegido ya exista se preguntará al usuario si desea o no sobreescribirlo. En caso de que decida sobreescribirlo el caso de uso seguirá su secuencia normal; si no, se cancelará el guardado del archivo. |
| | 6 | En caso de que la ubicación no pueda ser accedida (por ejemplo, por un problema de permisos) se notificará del error y se devolverá el control al usuario. |
| **Rendimiento** | Esta operación deberá realizarse de forma prácticamente inmediata. | |
| **Importancia** | Alta | |
| **Comentarios** | - | |

### *4.2.4. Casos de uso del Visor*

Por último, describiremos los casos de uso del nuevo visor.

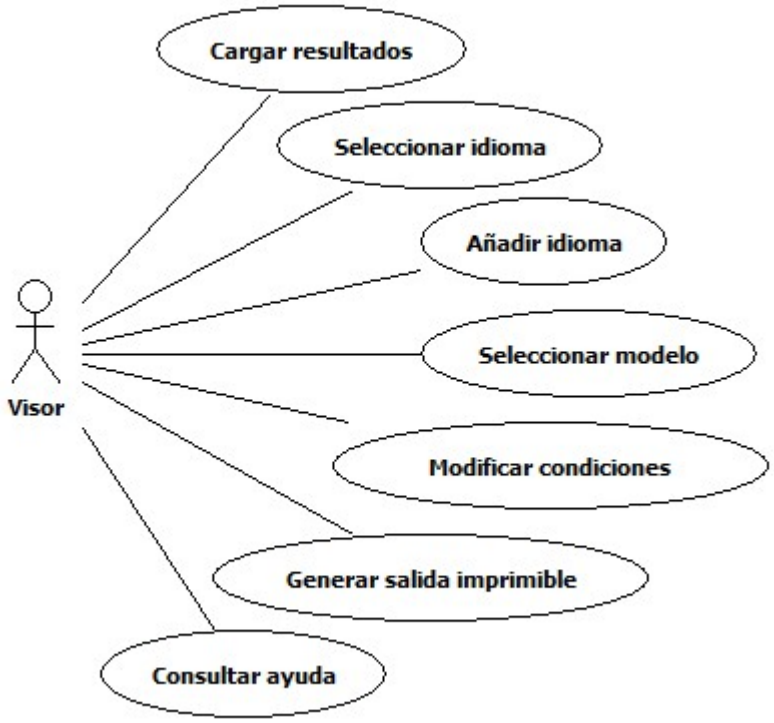



***Figura 4.4: Casos de uso del Visor***

| CU-14 | Cargar resultados | |
|---|---|---|
| **Descripción** | El sistema deberá comportarse de la siguiente forma cuando el usuario desee cargar los resultados obtenidos mediante un fichero de entrada. | |
| **Precondiciones** | • El fichero de resultados debe ser un fichero en formato XML correctamente formado. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario selecciona la opción de "Cargar resultados". |
| | **2** | El sistema solicita al usuario que escoja el fichero que contiene los resultados deseados. |
| | **3** | El usuario selecciona el fichero de resultados en el que se encuentran los resultados que se desean representar gráficamente. |
| | **4** | El sistema lee el fichero, carga los resultados y muestra al usuario una representación gráfica del primer posible conflicto encontrado. |
| **Postcondiciones** | Los resultados han sido cargados; lo que nos permitirá obtener una representación gráfica de cada uno de los posibles conflictos | |
| **Excepciones** | **Paso** | **Acción** |
| | **4** | En caso de que el archivo elegido no pueda ser abierto para lectura (ya sea por problemas de permisos, inexistencia de éste, etc.) deberá notificárselo al usuario y devolver a éste el control de la aplicación. |
| | **4** | En caso de que el archivo elegido no contenga una descripción de los resultados correcta el sistema deberá notificárselo al usuario y devolver a éste el control de la aplicación. |
| **Rendimiento** | Esta operación deberá realizarse de forma prácticamente inmediata. | |
| **Importancia** | Alta | |
| **Comentarios** | - | |

| CU-15 | Seleccionar idioma | |
|---|---|---|
| **Descripción** | El sistema deberá comportarse de la siguiente forma cuando el usuario desee modificar el idioma de la interfaz. | |
| **Precondiciones** | • El idioma deseado debe estar disponible para la aplicación. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario solicita al sistema modificar el idioma en el que se presenta la interfaz de la aplicación. |

| | | |
|---|---|---|
| | **2** | El sistema modificará todo el texto que aparezca en la interfaz de la aplicación para mostrarlo en el idioma seleccionado. |
| **Postcondiciones** | El idioma de la interfaz de la aplicación habrá sido modificado. | |
| **Excepciones** | **Paso** | **Acción** |
| | **2** | En caso de que haya algún error en la definición del idioma elegido, se mostrará un mensaje de error y se devolverá el control al usuario. |
| **Rendimiento** | Esta operación deberá realizarse de forma prácticamente inmediata. | |
| **Importancia** | Media | |
| **Comentarios** | - | |

| **CU-16** | **Añadir idioma** | |
|---|---|---|
| **Descripción** | El sistema deberá comportarse de la siguiente forma cuando el usuario desee añadir una nueva opción de idioma a la aplicación. | |
| **Precondiciones** | - | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario selecciona la opción de "Añadir idioma". |
| | **2** | El sistema solicita al usuario que escoja el fichero que contiene la descripción del idioma a añadir. |
| | **3** | El usuario selecciona el fichero que contiene la descripción del idioma que desea añadir. |
| | **4** | El sistema lee el fichero y añade la nueva opción de idioma a la aplicación basándose en los datos contenidos en éste. |
| **Postcondiciones** | Una vez completado este caso de uso tendremos disponible un nuevo idioma para la interfaz de nuestra aplicación. | |
| **Excepciones** | **Paso** | **Acción** |
| | **4** | En caso de que el archivo elegido no pueda ser abierto para lectura (ya sea por problemas de permisos, inexistencia de éste, etc.) deberá notificárselo al usuario y devolver a éste el control de la aplicación. |
| | **4** | En caso de que el archivo elegido no contenga una definición correcta de un nuevo idioma el sistema deberá notificárselo al usuario y devolver a éste el control de la aplicación. |
| **Rendimiento** | Esta operación deberá realizarse de forma prácticamente inmediata. | |
| **Importancia** | Media | |
| **Comentarios** | - | |

| CU-17 | Seleccionar modelo | |
|---|---|---|
| **Descripción** | El sistema deberá comportarse de la siguiente forma cuando el usuario desee visualizar otro MEM distinto al que actualmente se está mostrando (pero perteneciente al mismo conjunto de resultados). | |
| **Precondiciones** | • Debe haberse cargado de forma satisfactoria un fichero de resultados. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario seleccionará el modelo que desee visualizar. |
| | **2** | El sistema mostrará al usuario una representación gráfica del MEM seleccionado. |
| **Postcondiciones** | Tras la realización de este caso de uso, se mostrará la representación gráfica del MEM seleccionado. | |
| **Excepciones** | **Paso** | **Acción** |
| | **-** | - |
| **Rendimiento** | Esta operación deberá realizarse de forma prácticamente inmediata. | |
| **Importancia** | Alta | |
| **Comentarios** | - | |

| CU-18 | Modificar condiciones | |
|---|---|---|
| **Descripción** | El sistema deberá comportarse de la siguiente forma cuando el usuario desee modificar el valor de las condiciones para comprobar a qué ecuaciones/interpretaciones afecta. | |
| **Precondiciones** | • Debe haberse cargado de forma satisfactoria un fichero de resultados.<br>• El modelo actualmente seleccionado debe estar afectado por alguna condición. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario solicita al sistema la modificación de una condición. |
| | **2** | El sistema actualiza la representación gráfica del actual modelo indicando (mediante colores, por ejemplo) qué ecuaciones/interpretaciones dejarán de ser válidas con el nuevo valor de la condición modificada. |
| **Postcondiciones** | Tras completar este caso de uso podremos ver gráficamente a qué ecuaciones/interpretaciones afectan las condiciones modificadas. | |
| **Excepciones** | **Paso** | **Acción** |
| | **-** | - |

| | |
|---|---|
| **Rendimiento** | Esta operación deberá realizarse de forma prácticamente inmediata. |
| **Importancia** | Media |
| **Comentarios** | - |

| **CU-19** | **Generar salida imprimible** | |
|---|---|---|
| **Descripción** | El sistema deberá comportarse de la siguiente forma cuando el usuario desee obtener una versión imprimible del modelo que se está visualizando actualmente. | |
| **Precondiciones** | • Debe haberse cargado de forma satisfactoria un fichero de resultados.<br>• Se debe estar visualizando actualmente el modelo del que se desee obtener una versión imprimible. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario solicita al sistema obtener una versión imprimible (ps, pdf...) del modelo que se está visualizando actualmente. |
| | **2** | El sistema solicita al usuario que elija dónde se almacenará el fichero resultante. |
| | **3** | El usuario especifica la ubicación y el nombre del archivo en el que se almacenará la representación del modelo. |
| | **4** | El sistema almacena la representación del modelo en la ubicación especificada. |
| **Postcondiciones** | Tras completar correctamente este caso de uso se deberá haber generado un fichero imprimible con la representación gráfica del modelo actual. | |
| **Excepciones** | **Paso** | **Acción** |
| | **4** | En caso de que el destino elegido ya exista se preguntará al usuario si desea o no sobreescribirlo. En caso de que decida sobreescribirlo el caso de uso seguirá su secuencia normal; si no, se cancelará el guardado del archivo. |
| | **4** | En caso de que la ubicación no pueda ser accedida (por ejemplo, por un problema de permisos) se notificará del error y se devolverá el control al usuario. |
| **Rendimiento** | Esta operación deberá realizarse de forma prácticamente inmediata. | |
| **Importancia** | Baja | |
| **Comentarios** | - | |

| CU-20 | Consultar ayuda | |
|---|---|---|
| **Descripción** | El sistema deberá comportarse de la siguiente forma cuando el usuario desee obtener ayuda sobre el funcionamiento de la aplicación. | |
| **Precondiciones** | - | |
| **Secuencia normal** | **Paso** | **Acción** |
| | **1** | El usuario solicita al sistema acceder a una ayuda sobre el funcionamiento de la aplicación. |
| | **2** | El sistema muestra al usuario una ayuda sobre el funcionamiento de la aplicación. |
| **Postcondiciones** | Tras la finalización del caso de uso, será visible en la pantalla una ayuda sobre el funcionamiento de la aplicación. | |
| **Rendimiento** | Esta operación deberá realizarse de forma prácticamente inmediata. | |
| **Importancia** | Baja | |
| **Comentarios** | - | |

## 4.3. 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.

Como hicimos con los casos de uso, empezaremos por analizar los cambios que se han realizado en la aplicación principal, luego continuaremos describiendo la estructura de los algoritmos que se han desarrollado en este proyecto y, para finalizar analizaremos la nueva herramienta de visualización.

### *4.3.1. Diagrama de clases de la aplicación de CPCs*

Como ya hemos comentado antes, partimos de la aplicación desarrollada en el proyecto de Rubén Lajo [Laj07], al que se han realizado algunas modificaciones. En el siguiente diagrama se marcarán con un sombreado las clases que se han modificado. Para obtener más información sobre el resto de clases y sus métodos se puede consultar la memoria de dicho proyecto.

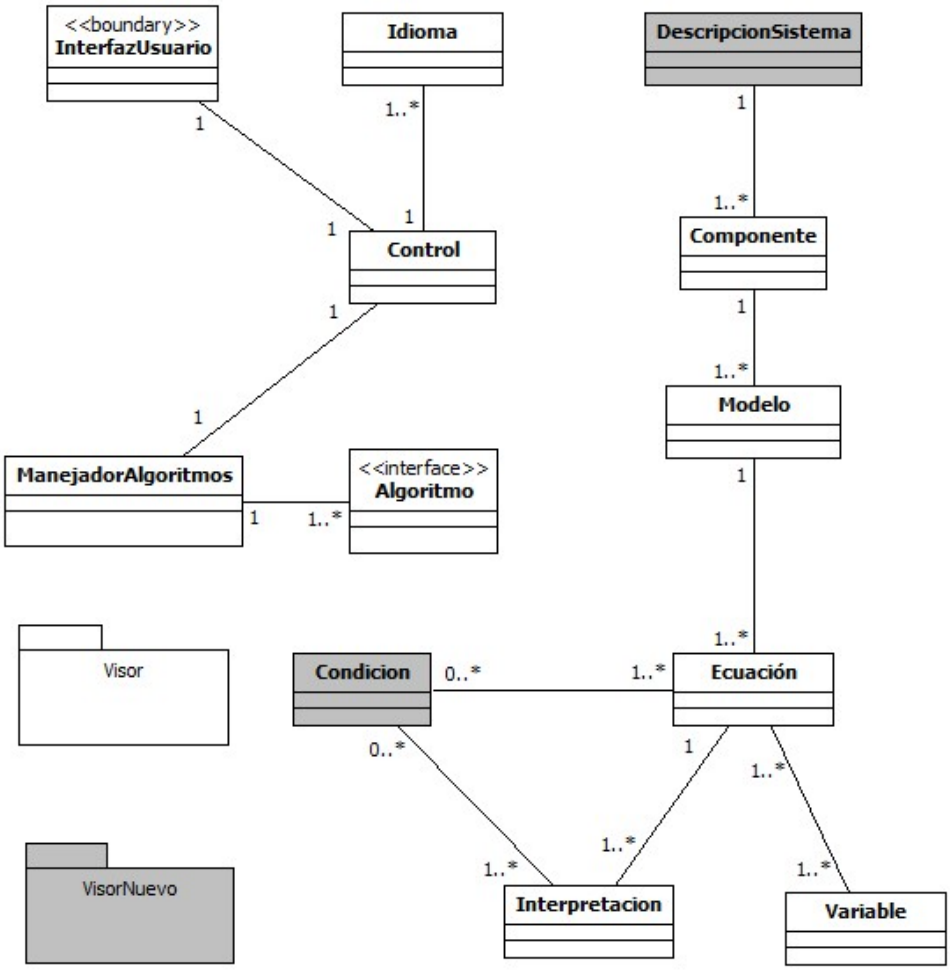



***Figura 4.5: Clases de la aplicación de CPCs***

| DescripcionSistema | |
|---|---|
| **Responsabilidades** | |
| Clase que representa el sistema sobre el que vamos a realizar los cálculos. | |
| **Atributos** | |
| ***Nombre(Tipo)*** | **Descripción** |
| *nombre(String)* | Nombre que identifica al sistema |
| **Operaciones** | |
| *crear(String ficheroEntrada)* | |
| *Objetivos* | Lee y almacena un sistema partiendo de un fichero de entrada. Esta lectura incluirá información de las condiciones, a diferencia de las versiones anteriores de la aplicación. |
| *Entrada* | El fichero que contiene la descripción del sistema. |
| *Salida* | - |

| Condición | |
|---|---|
| **Responsabilidades** | |
| Clase que representa una condición del sistema. | |
| **Atributos** | |
| ***Nombre(Tipo)*** | **Descripción** |
| *nombre(String)* | Nombre que identifica unívocamente a la condición. |
| *valor(String)* | Valor de la condición (verdadero, falso ó indiferente) |

## 4.3.2. Diagrama de clases del algoritmo básico

A continuación mostraremos el diagrama de clases del algoritmo básico de CPCs. Aunque se han producido variaciones en dicho algoritmo, desde el punto de vista estructural dichas variaciones no tienen ningún efecto; por lo que remitiremos también al lector a la memoria del proyecto de Rubén Lajo [Laj07] para obtener una descripción de las clases y sus métodos.

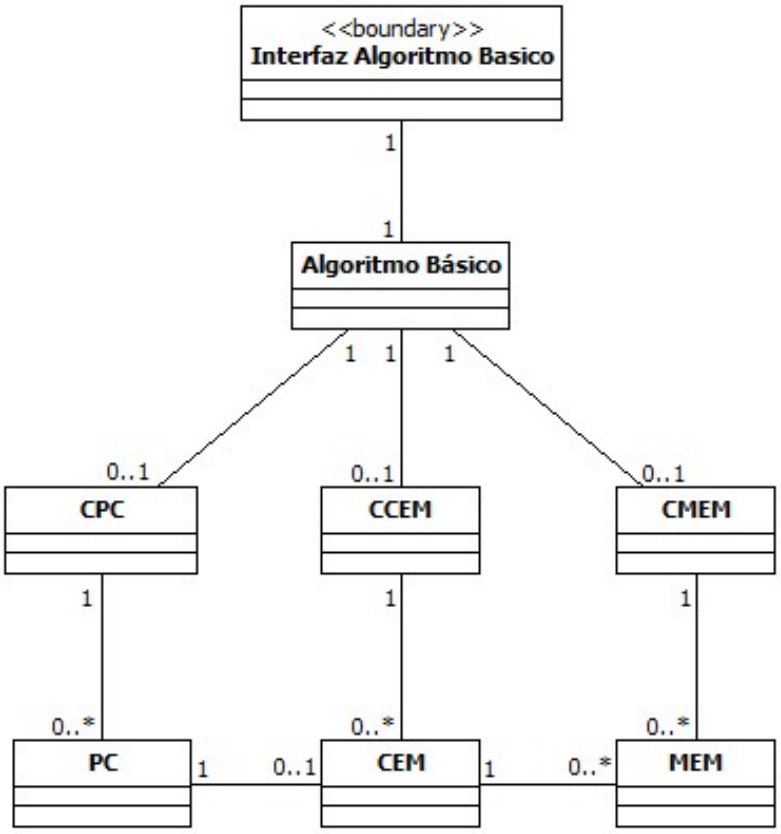



***Figura 4.6:Clases del algoritmo de CPCs básico***

### *4.3.3. Diagrama de clases del algoritmo con condiciones*

Algo similar podemos decir sobre el diagrama de clases del algoritmo con condiciones; sólo que esta vez remitiremos a la memoria de Raúl Sánchez [San07].

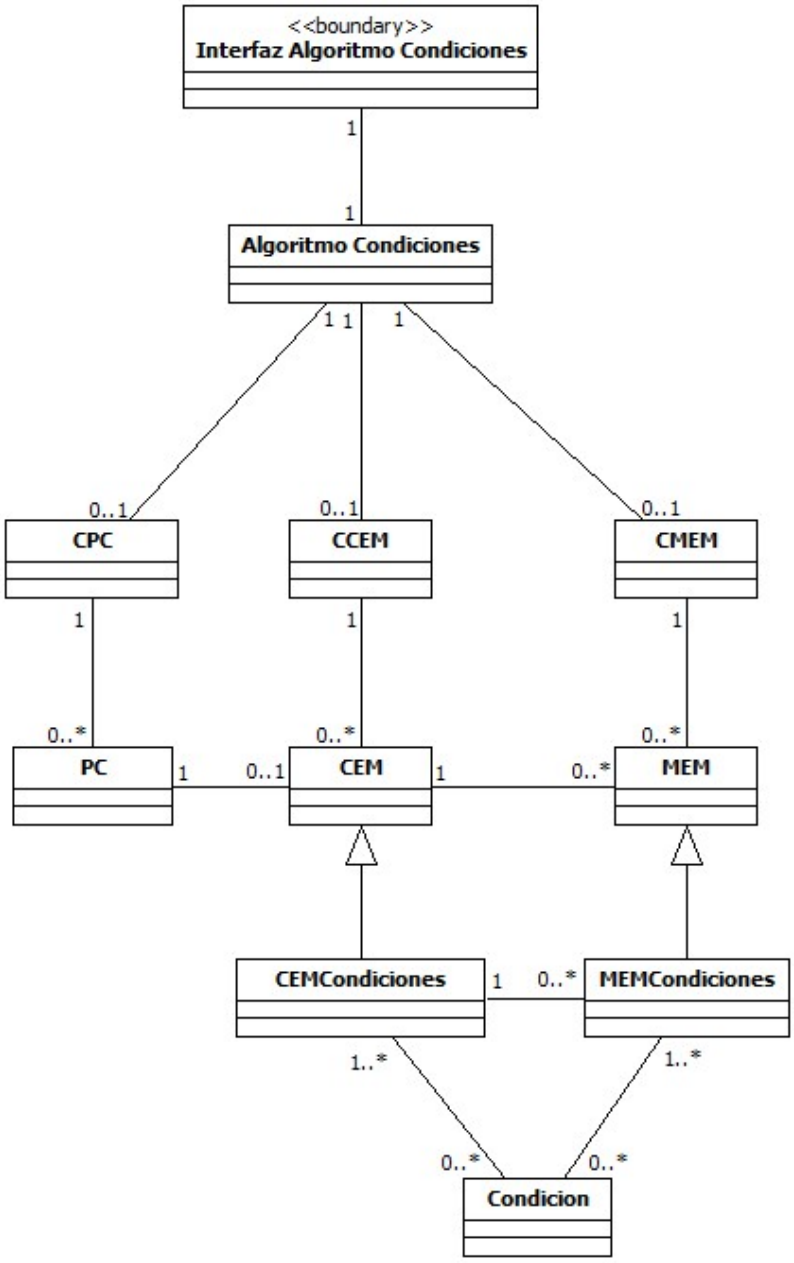



***Figura 4.7: Clases del algoritmo de CPCs con condiciones***

## 4.3.4. Diagrama de clases del Visor

Para finalizar mostraremos el diagrama de clases de la nueva herramienta de visualización desarrollada.

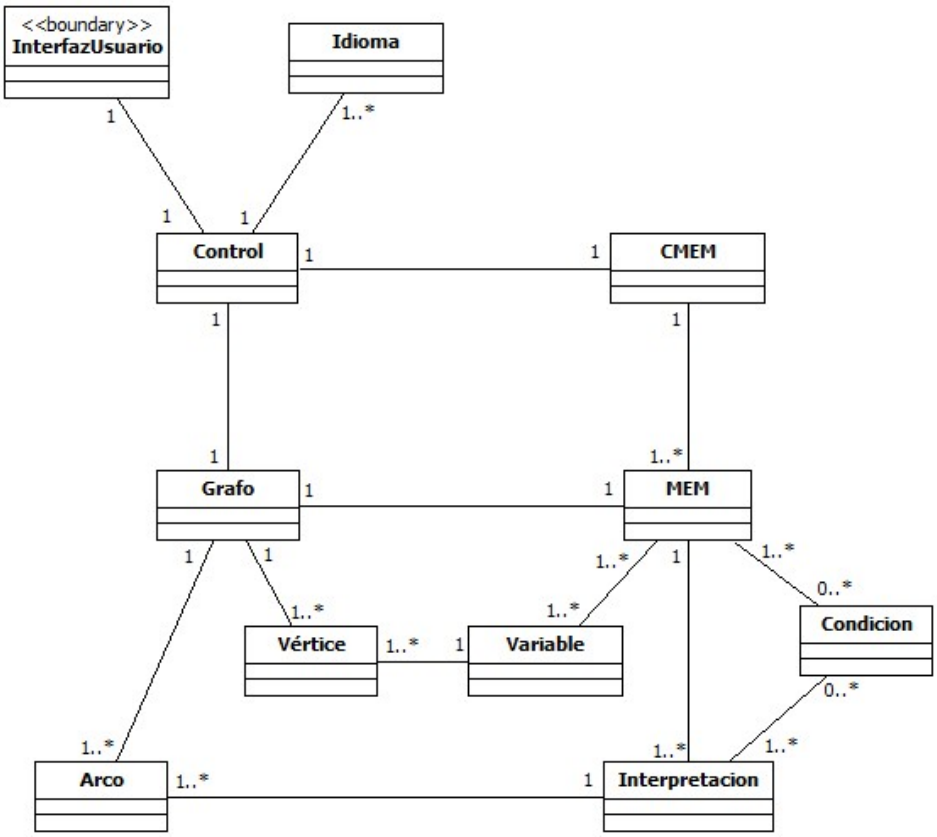



***Figura 4.8:Clases del Visor***

| **<<boundary>>**<br>**InterfazUsuario** | |
|---|---|
| **Responsabilidades** | |
| Clase encargada de realizar la interacción con el usuario del sistema. | |
| **Atributos** | |
| ***Nombre(Tipo)*** | **Descripción** |
| *idioma(String)* | Almacena el nombre del idioma actualmente seleccionado. |
| *cmem(CMEM)* | Almacena el CMEM que se encuentra cargado actualmente. |
| *selectedMEM(MEM)* | Almacena el MEM que se está representando actualmente. |
| *condiciones(condición[])* | Almacena el conjunto de condiciones que afectan al MEM actualmente seleccionado con sus actuales valores (recordemos que el usuario puede modificar libremente dichos valores). |
| **Operaciones** | |
| *actualizaIdioma(String idioma)* | |
| *Objetivos* | Modificará todo el texto que aparezca en la interfaz de la aplicación traduciéndolo al idioma que se proporciona como entrada. |
| *Entrada* | El idioma al que deseemos cambiar la interfaz de la aplicación. |
| *Salida* | - |
| *cargaCMEM(CMEM cmem)* | |
| *Objetivos* | Modificará el CMEM que se encuentra cargado actualmente. |
| *Entrada* | El CMEM que deseamos cargar. |
| *Salida* | - |
| *cargaMEM(MEM selectedMEM)* | |
| *Objetivos* | Modificará el MEM que estamos visualizando actualmente, mostrando el nuevo MEM que deseamos visualizar y actualizando la interfaz de la forma que sea necesaria. |
| *Entrada* | El MEM que se desee visualizar. |
| *Salida* | - |

| *versionImprimible(String nomFichero)* | |
|---|---|
| *Objetivos* | Almacena la representación gráfica del MEM actualmente seleccionado en un archivo imprimible (pdf, ps…). |
| *Entrada* | La ruta completa del archivo en el que se almacenará la representación gráfica del MEM. |
| *Salida* | - |
| *actualizaCond()* | |
| *Objetivos* | Modificará la representación del MEM que se está visualizando de acuerdo con los cambios que el usuario haya introducido en las condiciones. |
| *Entrada* | - |
| *Salida* | - |
| *acerca()* | |
| *Objetivos* | Acercará la vista de la representación gráfica del MEM que se está visualizando. |
| *Entrada* | - |
| *Salida* | - |
| *aleja()* | |
| *Objetivos* | Alejará la vista de la representación gráfica del MEM que se está visualizando. |
| *Entrada* | - |
| *Salida* | - |

| **Idioma** | |
|---|---|
| **Responsabilidades** | |
| Clase encargada de almacenar la información acerca de un idioma. | |
| **Atributos** | |
| ***Nombre(Tipo)*** | **Descripción** |
| *nombre(String)* | Almacena el nombre del idioma. |
| **Operaciones** | |
| *obtenerCadena(String etiqueta):String* | |
| *Objetivos* | Devolverá el texto correspondiente a la etiqueta indicada en el idioma actual. |
| *Entrada* | La etiqueta de la cual necesitamos obtener su texto. |
| *Salida* | El texto correspondiente a la etiqueta en el idioma actual. |

| **Control** | |
|---|---|
| **Responsabilidades** | |
| Clase encargada del control de la funcionalidad de toda la aplicación. | |
| **Atributos** | |
| ***Nombre(Tipo)*** | **Descripción** |
| *idioma(String)* | Almacena el nombre del idioma actualmente seleccionado. |
| *cmem(CMEM)* | Almacena el CMEM que se encuentra cargado actualmente. |
| *selectedMEM(MEM)* | Almacena el MEM que se está representando actualmente. |
| *condiciones(condición[])* | Almacena el conjunto de condiciones que afectan al MEM actualmente seleccionado con sus actuales valores (recordemos que el usuario puede modificar libremente dichos valores). |
| **Operaciones** | |
| *actualizaIdioma(String idioma)* | |
| *Objetivos* | Indica a la interfaz que actualice la interfaz de acuerdo al idioma indicado. |
| *Entrada* | El idioma al que deseemos cambiar la interfaz de la aplicación. |
| *Salida* | - |
| *cargaCMEM(CMEM cmem)* | |
| *Objetivos* | Modifica el CMEM que se encuentra cargado actualmente. |
| *Entrada* | El CMEM que deseamos cargar. |
| *Salida* | - |
| *cargaMEM(MEM selectedMEM)* | |
| *Objetivos* | Indica a la interfaz que muestre el MEM indicado. |
| *Entrada* | El MEM que se desee visualizar. |
| *Salida* | - |
| *versionImprimible(String nomFichero)* | |
| *Objetivos* | Almacena la representación gráfica del MEM actualmente seleccionado en un archivo imprimible (pdf, ps…). |
| *Entrada* | La ruta completa del archivo en el que se almacenará la representación gráfica del MEM. |
| *Salida* | - |

| *actualizaCond()* | |
|---|---|
| *Objetivos* | Indica a la interfaz que modifique la representación del MEM que se está visualizando de acuerdo con los cambios que el usuario haya introducido en las condiciones. |
| *Entrada* | - |
| *Salida* | - |
| *acerca()* | |
| *Objetivos* | Indica a la interfaz que acerque la vista de la representación gráfica del MEM que se está visualizando. |
| *Entrada* | - |
| *Salida* | - |
| *aleja()* | |
| *Objetivos* | Indica a la interfaz que aleje la vista de la representación gráfica del MEM que se está visualizando. |
| *Entrada* | - |
| *Salida* | - |

| **CMEM** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa los posibles conflictos que queremos visualizar en forma de un CMEM. | |
| **Operaciones** | |
| *cargaResultados(String nomFichero)* | |
| *Objetivos* | Carga un CMEM partiendo del fichero de resultados indicado. |
| *Entrada* | La ruta completa al fichero donde se encuentran los resultados que se desean visualizar. |
| *Salida* | - |

| **MEM** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa un MEM concreto. | |
| **Atributos** | |
| ***Nombre(Tipo)*** | **Descripción** |
| *id(Integer)* | Entero que identifica unívocamente al MEM. |
| *ecCadena(String)* | Ecuaciones que componen la CEM de la que se obtuvo este MEM. |
| *nodoDiscrepancia(Variable)* | Variable cabeza de la primera interpretación del MEM. Dicho vértice será la raíz del hipergrafo que vamos a dibujar. |

| **Interpretación** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa una interpretación. | |
| **Atributos** | |
| ***Nombre(Tipo)*** | **Descripción** |
| *nombre(String)* | Nombre que identifica unívocamente a la interpretación. |
| *tipo(String)* | Tipo de la interpretación (integral ó derivada). |
| *cabeza(Variable)* | Variable cabeza de la interpretación. |
| *cola(Variable[])* | Conjunto de variables que forman la cola de la interpretación. |

| **Variable** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa a una variable del sistema. | |
| **Atributos** | |
| ***Nombre(Tipo)*** | **Descripción** |
| *nombre(String)* | Nombre que identifica unívocamente a la variable. |
| *observada(Boolean)* | Indica si la variable es o no observada. |

| **Condición** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa una condición del sistema. | |
| **Atributos** | |
| ***Nombre(Tipo)*** | **Descripción** |
| *nombre(String)* | Nombre que identifica unívocamente a la condición. |
| *valor(String)* | Valor de la condición (verdadero, falso ó indiferente) |

| **Grafo** | |
|---|---|
| **Responsabilidades** | |
| Clase que contiene el grafo que representa un MEM. | |
| **Operaciones** | |
| *crearGrafo(MEM mem)* | |
| *Objetivos* | Crea el grafo que representa el MEM indicado. |
| *Entrada* | El MEM del cual queremos obtener el grafo. |
| *Salida* | - |

| **Vértice** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa un vértice del grafo. Cada vértice se corresponderá con una variable. | |
| **Atributos** | |
| ***Nombre(Tipo)*** | **Descripción** |
| *id(String)* | Nombre que identifica unívocamente al vértice. |

| **Arco** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa un arco del grafo. Un conjunto de arcos que llegan a un mismo vértice representarán una interpretación. | |
| **Atributos** | |
| ***Nombre(Tipo)*** | **Descripción** |
| *id(String)* | Nombre que identifica unívocamente al arco. |
| *origen(Vértice)* | Vértice origen del arco. |
| *destino(Vértice)* | Vértice destino del arco. |

# 4.4. Modelo de comportamiento

En este apartado se dará, mediante el diagrama de actividad que vemos en *la figura 4.9*, una idea de la secuencia de pasos que sigue el sistema en un ciclo de funcionamiento básico.

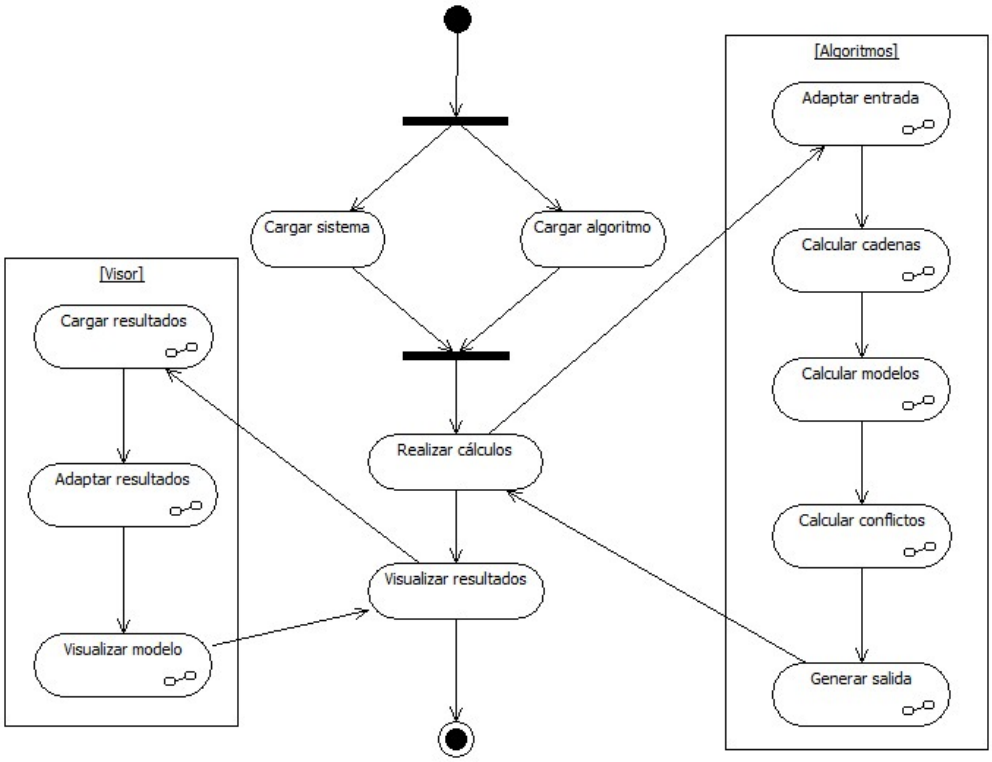



***Figura 4.9: Diagrama de actividad***

Vemos que tenemos dos partes diferenciadas del resto del diagrama. La de la izquierda corresponde con las actividades que realizará la aplicación de visualización de resultados. La de la derecha, se corresponde con las actividades que dependerán del algoritmo que haya sido cargado.

# 4.5. Modelo de interacción

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 4.2 (excepto para los que se corresponden a los algoritmos; pues, como ya indicamos en el apartado anterior, al nivel del análisis no presentan ninguna diferencia con los descritos en las memorias de los proyectos de Rubén Lajo [Laj07] y Raúl Sánchez [San07]).

## *4.5.1. Escenarios en la aplicación de CPCs*

### *4.5.1.1. Escenario de Cargar sistema (Aplicación de CPCs)*

(*Ver figura 4.10*). Este escenario se iniciará con el usuario solicitando a la interfaz la carga de un sistema. La interfaz requerirá que el usuario le proporcione un archivo de entrada para obtener los datos. Una vez se le ha proporcionado, comenzará la carga del sistema.

En primer lugar, se enviará la orden al objeto de la clase Control. Éste creará un objeto de tipo DescripcionSistema que irá añadiendo todos sus elementos.

El objeto de tipo DescripcionSistema contendrá un conjunto de elementos de tipo Componente que deberemos de crear uno por uno.

A su vez, para cada objeto de tipo Componente tendremos varios objetos de tipo Modelo; para cada objeto Modelo, varias ecuaciones; para cada Ecuación, un conjunto de interpretaciones, variables y condiciones; y, por último, para cada interpretación, un conjunto de condiciones.

## *4.5.2. Escenarios en la aplicación Visor*

### *4.5.2.1. Escenario de Cargar resultados*

(*Ver figura 4.11*). Este escenario es muy similar al anterior. En este caso los resultados se almacenarán como un objeto de la clase CMEM. Éste contendrá varios objetos de la clase MEM, que contendrán a su vez un conjunto de interpretaciones y variables.

Los objetos de tipo Interpretación podrán contener también un conjunto de condiciones.

#### *4.5.2.2. Escenario de Seleccionar idioma*

(*Ver figura 4.12*). En este escenario el usuario solicitará a la interfaz modificar el idioma. La interfaz notificará al objeto de la clase Control, que creará un nuevo objeto de tipo Idioma y lo enviará de vuelta a la interfaz para que se actualice.

#### *4.5.2.3. Escenario de Añadir idioma*

(*Ver figura 4.13*). En este escenario, el usuario solicitará a la interfaz añadir un nuevo idioma. Para ello, la interfaz le solicitará que seleccione el archivo en el que se encuentra la descripción del nuevo idioma. Con este archivo, la interfaz notificará al objeto Control para que añada el nuevo idioma. Éste creará un nuevo objeto Idioma y lo añadirá a la lista de idiomas disponibles.

#### *4.5.2.4. Escenario de Seleccionar modelo*

(*Ver figura 4.14*). En este escenario el usuario indicará a la interfaz el MEM que desea visualizar; la cual pasará la orden al objeto Control.

El objeto de la clase Control extraerá la información del MEM a visualizar del objeto de tipo CMEM (que contiene todos los MEMs cargados). Con esa información creará un objeto de tipo Grafo. El grafo se construirá añadiendo un conjunto de vértices y arcos. Una vez construido, se enviará a la interfaz para que ésta lo muestre al usuario.

#### *4.5.2.5. Escenario de Modificar condiciones*

(*Ver figura 4.15*). En este escenario el usuario realizará una modificación en las condiciones que desea visualizar en este momento y que afectan al MEM que se está mostrando actualmente. La interfaz notificará al objeto Control de dichos cambios; éste comprobará qué condiciones del MEM ya no son compatibles y proporcionará esa información a la interfaz para que se la muestre al usuario.

#### *4.5.2.6. Escenario de Generar salida imprimible*

(*Ver figura 4.16*). En este escenario, el usuario solicitará a la interfaz almacenar en un fichero la representación gráfica del MEM que se está visualizando actualmente. La interfaz le solicitará que proporcione el lugar en el que se almacenará el fichero y enviará esta información al objeto Control.

El objeto Control obtendrá de la interfaz la representación del MEM y la almacenará en el archivo especificado.

#### *4.5.2.7. Escenario de Consultar ayuda*

(*Ver figura 4.17*). Este escenario es bastante simple. El usuario solicita a la interfaz obtener ayuda sobre la aplicación. La interfaz notificará esta petición al objeto Control, el cual indicará a la interfaz que muestre la ayuda al usuario.

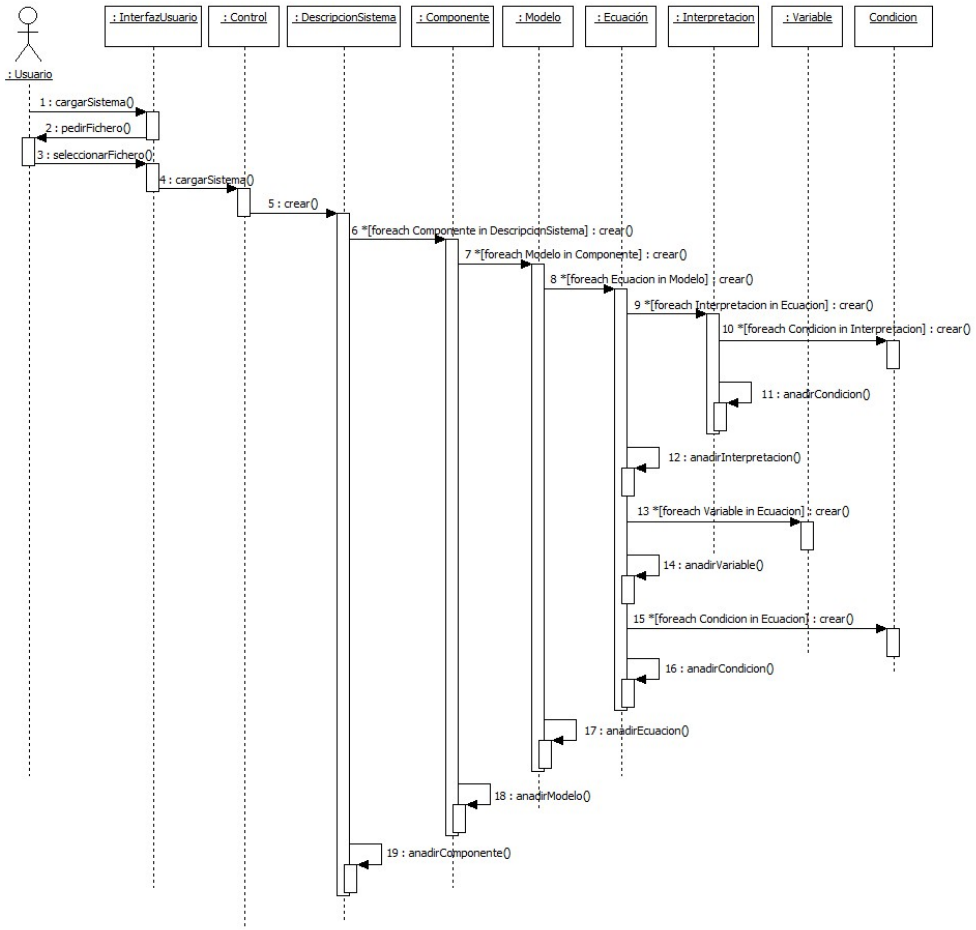



***Figura 4.10: Diagrama de Secuencia de Cargar Sistema***

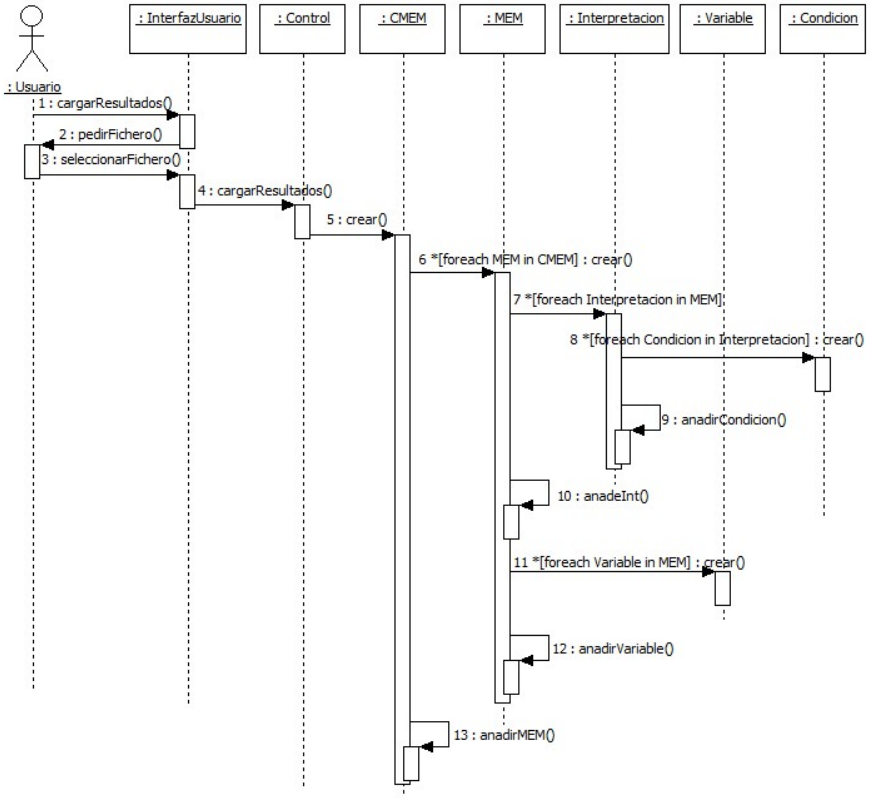



***Figura 4.11: Diagrama de Secuencia de Cargar Sistema***

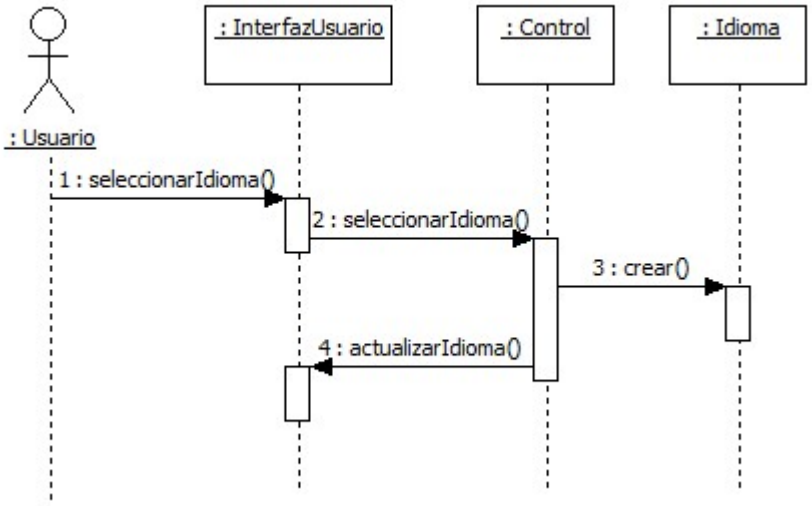



***Figura 4.12: Diagrama de Secuencia de Seleccionar Idioma***

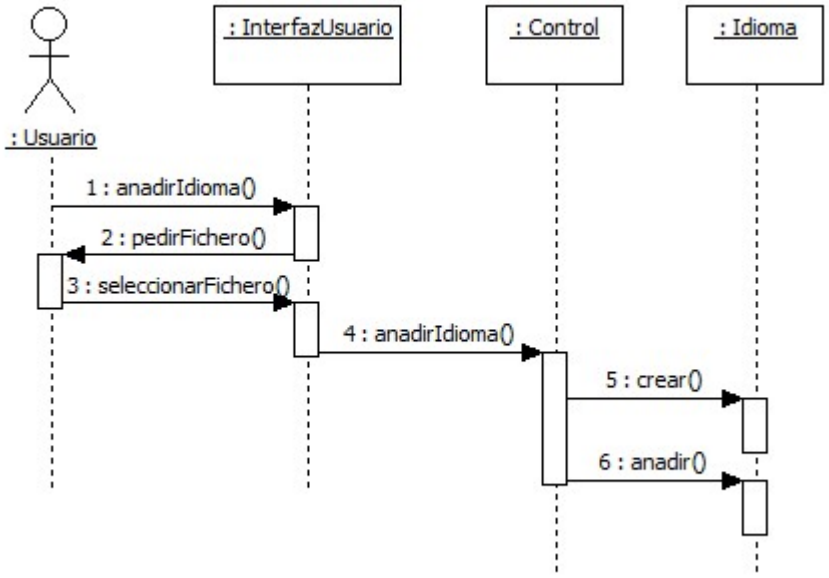



***Figura 4.13: Diagrama de Secuencia de Añadir Idioma***

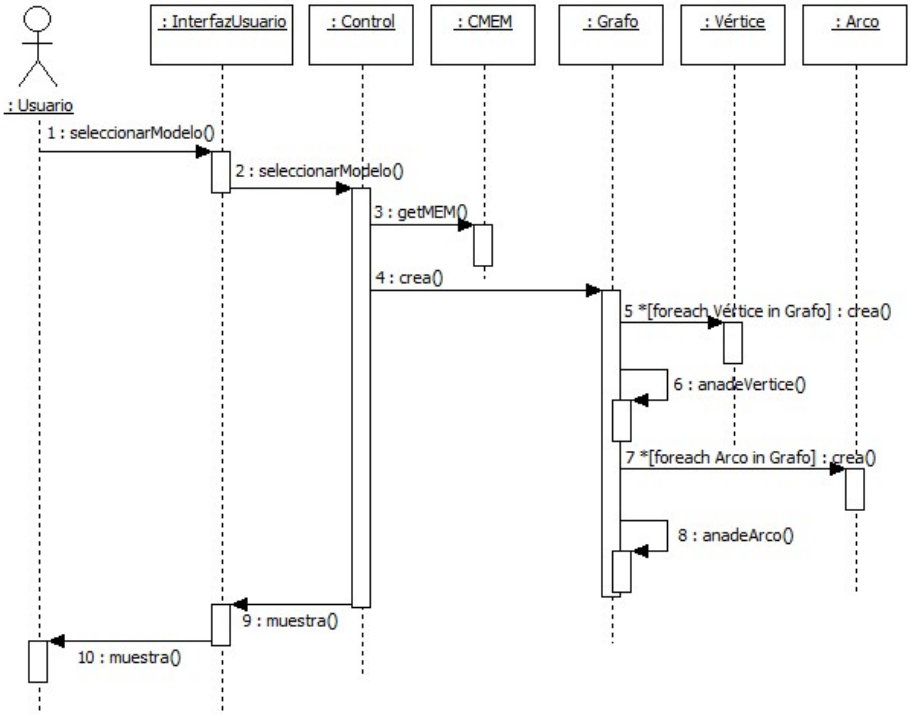



***Figura 4.14: Diagrama de Secuencia de Seleccionar modelo***

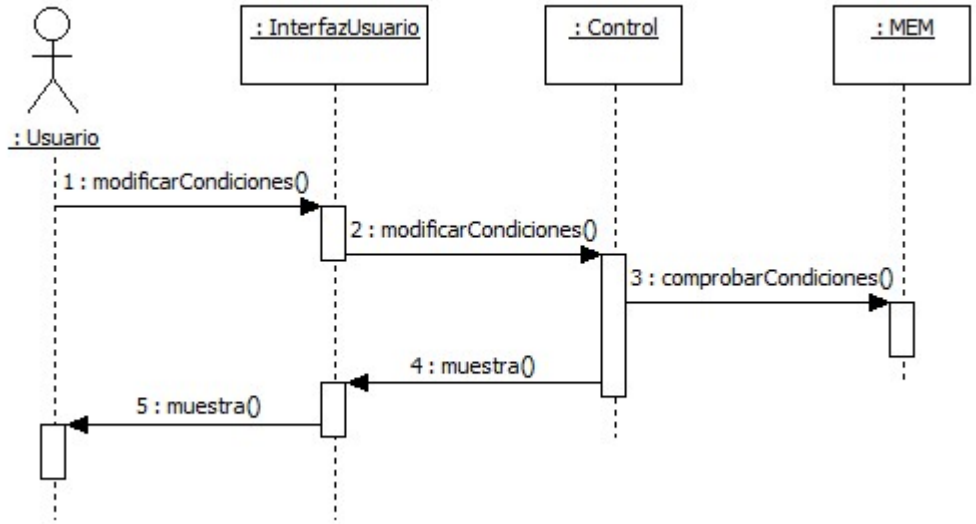



***Figura 4.15: Diagrama de Secuencia de Modificar Condiciones***

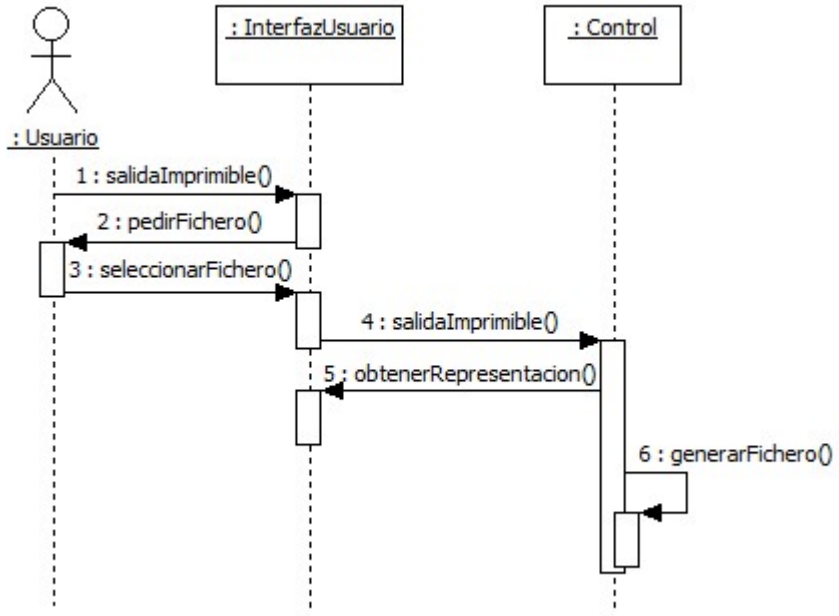



***Figura 4.16: Diagrama de Secuencia de Generar Salida Imprimible***

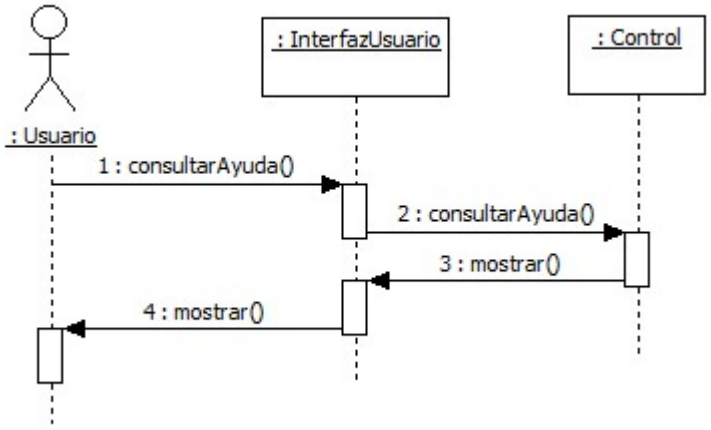



***Figura 4.17: Diagrama de Secuencia de Consultar Ayuda***

# 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 modelo de diseño especificará la forma en que debe llevarse a cabo la implementación del proyecto "Revisión y adaptación de la herramienta para el cálculo de Posibles Conflictos".

### *5.1.3. Definiciones, Acrónimos, y Abreviaturas*

Ver el glosario general del proyecto.

### *5.1.4 Referencias*

En el presente capítulo encontramos referencias a los siguientes apartados del documento:

- Modelo de implementación
- Glosario

- Referencias bibliográficas

### 5.1.5. 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.
- Diseño de la interfaz: bocetos sobre el aspecto que presentará la interfaz de la herramienta de visualización de resultados.

## 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).

Esto es aplicable tanto para la herramienta de CPCs, como para los algoritmos de cálculo y para el Visor.

Pasemos a continuación a explicar en qué consiste el patrón de diseño que vamos a utilizar.

### 5.2.1. El patrón MVC

Este patrón, como ya hemos mencionado antes, será típicamente utilizado por cualquier aplicación que posea una interfaz gráfica. En él, las clases de las que se compondrá la aplicación estarán agrupadas en tres grandes paquetes, que son los que dan nombre al patrón:

- **Modelo**: contendrá toda la información relativa a los datos con los que trabajará la aplicación.

- **Vista**: será la encargada de la comunicación directa con el usuario. Su misión será reconocer las órdenes que éste le envía y mandárselas al Controlador.
- **Controlador**: será el encargado de llevar a cabo las órdenes que le llegan de la Vista, realizando los cambios correspondientes sobre el Modelo (o posiblemente también la propia Vista).

Podemos verlo gráficamente con este sencillo esquema:


***Figura 5.1: Esquema del patrón MVC***

Así, podemos ver cualquier interacción con el sistema como el siguiente conjunto de pasos:

1. El Usuario envía un estímulo al paquete Vista.
2. El paquete Vista reconoce ese estímulo y envía la orden correspondiente al paquete Controlador.
3. El paquete Controlador, valiéndose de los datos que se encuentran en el paquete Modelo, realiza las acciones correspondientes a la orden que ha recibido.
4. Una vez finalizadas, enviará el resultado de éstas al paquete Vista, el cual mostrará los resultados al Usuario.

En los siguientes apartados veremos cómo el diseño estará siempre enfocado a cumplir con el patrón que acabamos de describir.

# 5.3. Realización de casos de uso

En este apartado mostraremos los diagramas de casos de uso de la aplicación. En su mayoría, estos coincidirán con los mostrados en el modelo de diseño, aunque pueden añadirse/modificarse para adaptarnos a las exigencias de la codificación.

## *5.3.1. Casos de uso de la aplicación de CPCs*

Al igual que en el análisis, los casos de uso referentes a la aplicación de CPCs que sufrirán modificaciones durante el desarrollo de este proyecto aparecerán sombreados.

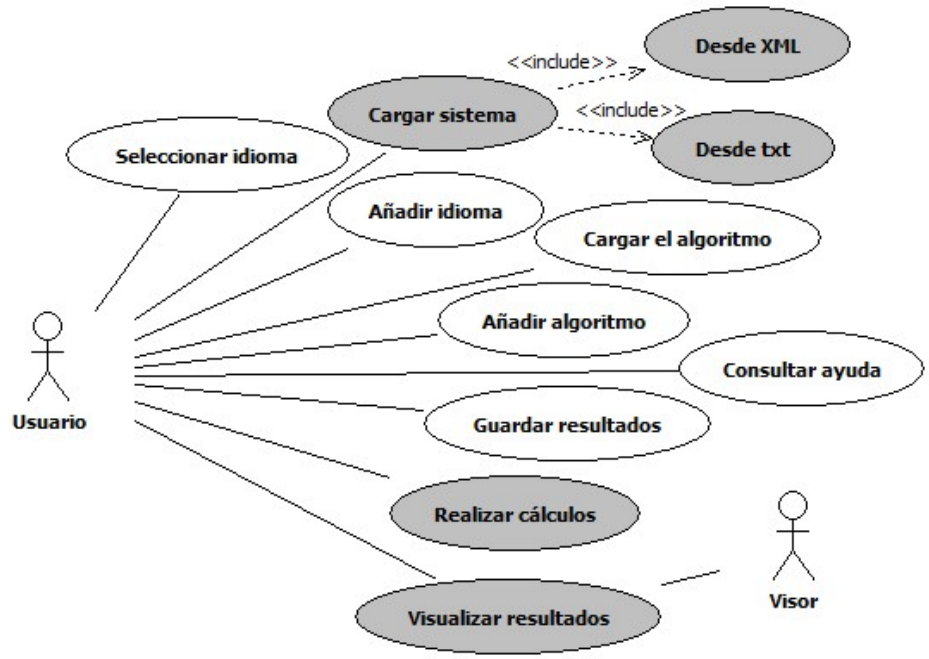



***Figura 5.2: Casos de uso de la aplicación de CPCs***

A partir de este punto se considera que todos los casos de uso serán originales o sufrirán alguna modificación durante el desarrollo del proyecto

## *5.3.2. Casos de uso del algoritmo básico de CPCs*

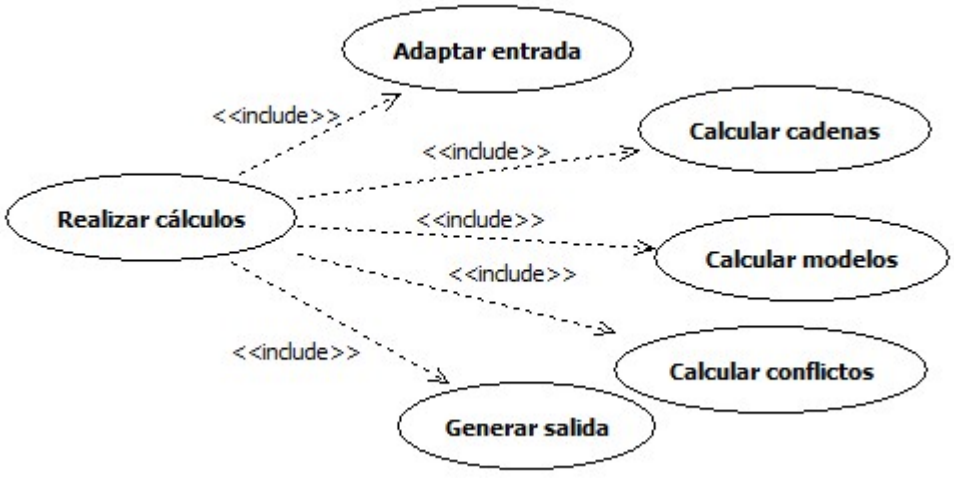



***Figura 5.3: Casos de uso del algoritmo básico de CPCs***

### 5.3.3. Casos de uso del algoritmo CPCs con condiciones

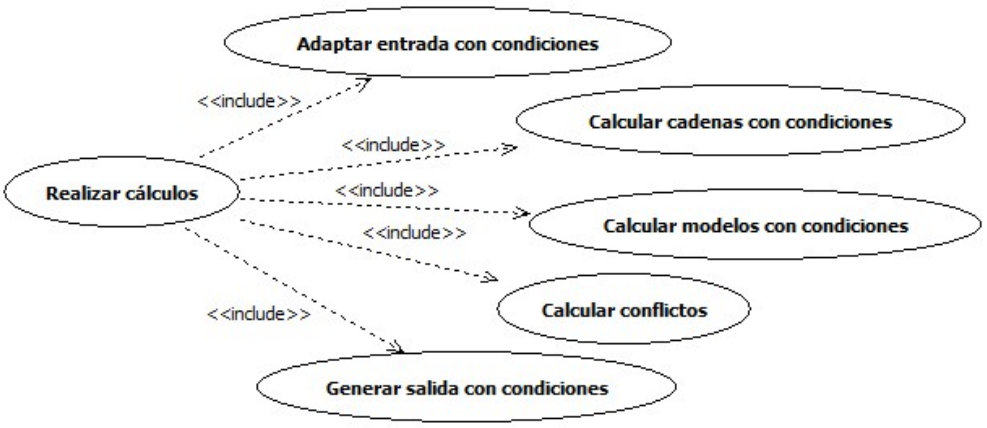



*Figura 5.4: Casos de uso del algoritmo de CPCs con condiciones*

### 5.3.4. Casos de uso del Visor

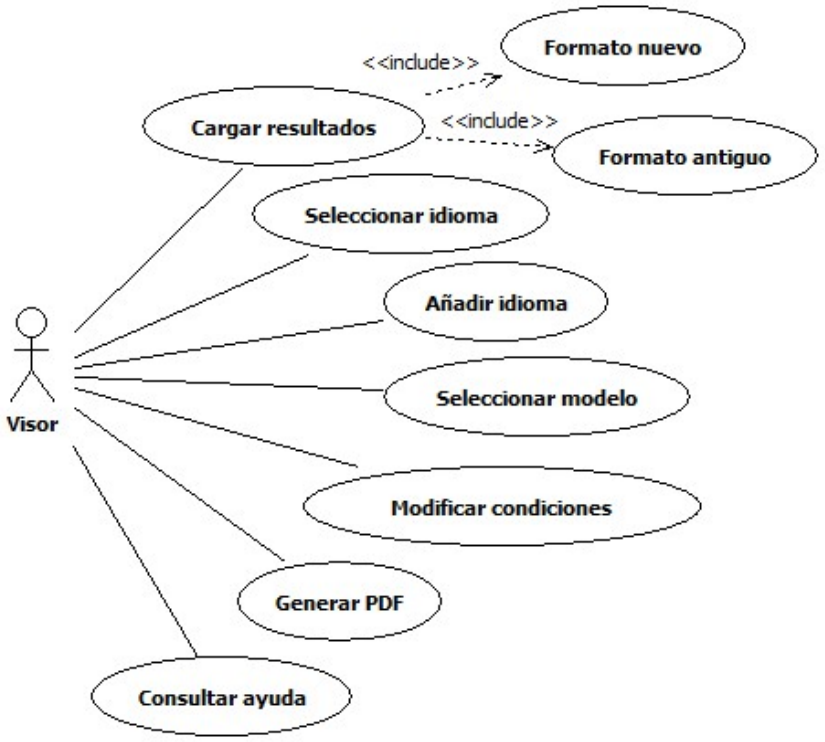



*Figura 5.5: Casos de uso del Visor*

# 5.4. Diseño de las clases

En este apartado haremos un refinamiento de las clases vistas en el modelo de análisis.

Las clases que definamos en este momento, junto con sus atributos y operaciones, serán las que posteriormente van a ser programadas.

Veremos además cómo cada una de las clases puede incluirse en uno de los tres paquetes que hemos mencionado en el apartado de Arquitectura del sistema (Modelo, Vista y Controlador).

## *5.4.1. Diagrama de clases de la aplicación de CPCs*

Al igual que vimos en el modelo de análisis, mostraremos un diagrama completo de las clases de la aplicación de CPCs desarrollada por Rubén Lajo [Laj07] y modificada por David Rubio. Todas las modificaciones realizadas están enfocadas a la adición de las condiciones a la descripción del sistema; por lo que se mostrarán en detalle las clases que son afectadas por éstas. Como siempre, remitimos al lector a la memoria de Rubén Lajo si necesita más información sobre el resto de clases.

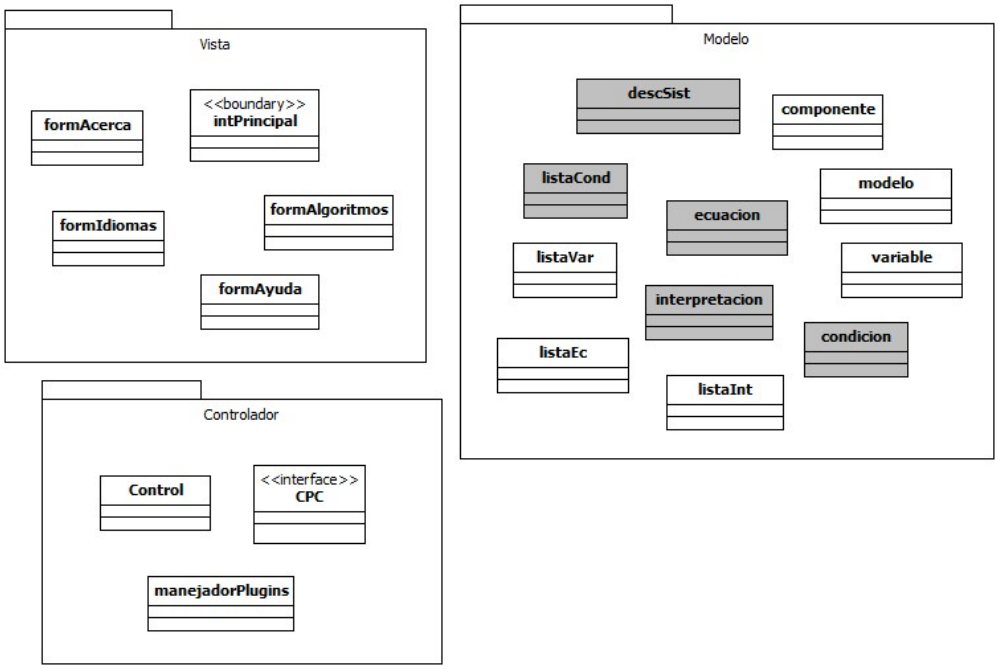



***Figura 5.6: Diagrama de clases de la aplicación de CPCs***

| descSist | |
|---|---|
| **Responsabilidades** | |
| Clase que representa el sistema sobre el que vamos a realizar los cálculos. | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *nombreOnto(String)* | Nombre de la ontología (sin relevancia en el actual proyecto). |
| *nombre(String)* | Nombre que identifica al sistema. |
| *obs(listaVar)* | Lista de variables observables del sistema. |
| *noobs(listaVar)* | Lista de variables no observables del sistema. |
| *raiz(componente)* | Primer componente del sistema. |
| *numrestricciones(int)* | Número de restricciones del sistema. |
| *condiciones(listaCond)* | Lista de condiciones del sistema. |
| *numcondiciones(int)* | Número de condiciones del sistema. |
| **Operaciones** | |
| *<<creador>>descSist():descSist* | |
| *Objetivos* | Crea un objeto de la clase descSist. |
| *Entrada* | - |
| *Salida* | El objeto creado. |
| *contieneComp(componente comp):boolean* | |
| *Objetivos* | Comprueba si el sistema contiene o no un componente. |
| *Entrada* | El componente del cual necesitamos conocer si está o no incluido en el sistema. |
| *Salida* | True (cierto) si el sistema contiene el componente y False (falso) en caso contrario. |
| *anadeComp(componente comp)* | |
| *Objetivos* | Añade al sistema un componente. |
| *Entrada* | El componente que deseamos añadir. |
| *Salida* | - |
| *desdeTxt(String fich, String nom)* | |
| *Objetivos* | Lee y almacena un sistema partiendo de un fichero de entrada en formato de texto plano (.txt). Esta lectura incluirá información de las condiciones, a diferencia de las versiones anteriores de la aplicación. |
| *Entrada* | El fichero que contiene la descripción del sistema y el nombre con el que se identificará al sistema. |
| *Salida* | - |

| *desdeXml(String fich)* | |
|---|---|
| *Objetivos* | Lee y almacena un sistema partiendo de un fichero de entrada en formato XML (.xml). Esta lectura incluirá información de las condiciones, a diferencia de las versiones anteriores de la aplicación. |
| *Entrada* | El fichero que contiene la descripción del sistema. |
| *Salida* | - |

| **condicion** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa una condición en el sistema. | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *nombre(String)* | Nombre que identifica a la condición |
| *valor(String)* | Valor de la condición ("T": verdadero, "F": falso, "X": indiferente) |
| *siguiente(condicion)* | Siguiente condición en una lista de condiciones. |
| **Operaciones** | |
| *<<creador>>condicion(String n, String v):condicion* | |
| *Objetivos* | Crear un nuevo objeto de la clase condicion. |
| *Entrada* | El nombre y el valor de la condición a crear. |
| *Salida* | El objeto creado. |
| *getValor():String* | |
| *Objetivos* | Devuelve el valor de la condición en un formato que facilita futuros cálculos. |
| *Entrada* | - |
| *Salida* | "0" si el valor de la condición es falso, "1" si es verdadero y "2" si es indiferente. |

| **listaCond** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa una lista enlazada de condiciones. | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *raiz(condicion)* | Primera condición de la lista. |
| **Operaciones** | |
| *<<creador>> listaCond():listaCond* | |
| *Objetivos* | Crear un nuevo objeto de la clase listaCond. |
| *Entrada* | - |
| *Salida* | El objeto creado. |

| *anadeCond(condicion)* | |
|---|---|
| *Objetivos* | Añade una nueva condición a la lista. |
| *Entrada* | La condición a añadir. |
| *Salida* | - |
| *contieneCond(condicion):boolean* | |
| *Objetivos* | Comprueba si la lista contiene una condición del mismo nombre que la proporcionada como argumento. |
| *Entrada* | La condición de la cual queremos comprobar su existencia. |
| *Salida* | True si la lista contiene una condición del mismo nombre que la proporcionada como argumento y false en caso contrario. |
| *contieneContraria(condicion):boolean* | |
| *Objetivos* | Comprueba si la lista contiene una condición del mismo nombre y valor incompatible que la proporcionada como argumento. |
| *Entrada* | La condición de la cual queremos comprobar su compatibilidad con la lista. |
| *Salida* | True en caso de que exista una condición con el mismo nombre y valor incompatible ("V" + "F") y false en caso contrario. |

A las otras dos clases que sufren modificaciones (*ecuacion* e *interpretacion*) simplemente se les añadirá el siguiente atributo:

- condiciones(listaCond): Lista de condiciones de la ecuación/interpretación.

## 5.4.2. Diagrama de clases del algoritmo básico

A continuación mostraremos el diagrama de clases del algoritmo básico de CPCs.

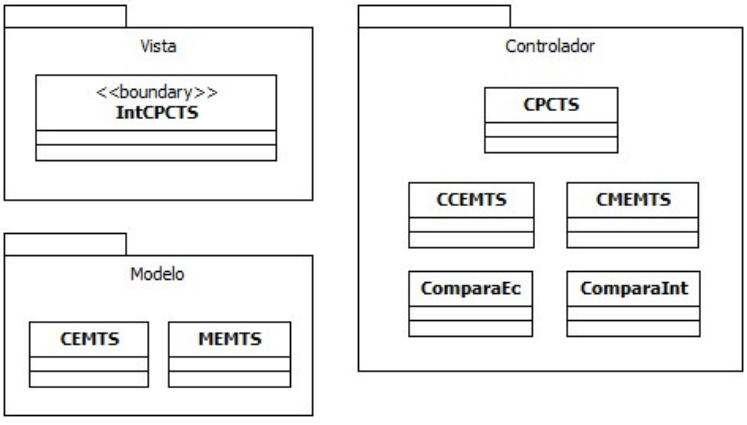



***Figura 5.7: Diagrama de clases del algoritmo básico***

*(Nota: la terminación TS que aparece en el nombre de las clases hace referencia a la estructura de datos utilizada. Se puede encontrar más información en el capítulo 6).*

| **<<boundary>> IntCPCTS** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa la interfaz principal del algoritmo con el usuario. | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *padre(CPCTS)* | Objeto encargado de ejecutar las acciones enviadas a la interfaz. |
| *CCEMcalc(boolean)* | Indica si el CCEM ha sido calculado. |
| *CMEMcalc(boolean)* | Indica si el CMEM ha sido calculado. |
| *CPCcalc(boolean)* | Indica si el CPCs ha sido calculado. |
| *mensajes(ResourceBundle)* | Objeto capaz de proporcionar un texto determinado en diferentes idiomas. |
| **Operaciones** | |
| *<<creador>> IntCPCTS(JPanel destino, CPCTS desde, ResourceBundle m):IntCPCTS* | |
| *Objetivos* | Crear un nuevo objeto de la clase IntCPCTS. |
| *Entrada* | La ventana en la que se mostrarán los resultados, el objeto que se encagará de ejecutar las acciones y el objeto encargado de proporcionar el texto en el idioma seleccionado. |
| *Salida* | El objeto creado. |
| *jbInit()* | |
| *Objetivos* | Moldear y colocar en su correspondiente lugar todos los elementos de la interfaz (menús, botones, texto, pestañas…). |
| *Entrada* | - |
| *Salida* | - |
| *imprimeCadenas(CCEMTS cad, double tiempo)* | |
| *Objetivos* | Muestra información sobre las CEM calculadas y el tiempo que se ha tardado en calcularlas. |
| *Entrada* | La estructura de datos que contiene las CEM y el tiempo en que se han calculado. |
| *Salida* | - |
| *imprimeModelos(CMEMTS mod, double tiempo)* | |
| *Objetivos* | Muestra información sobre los MEM calculadas y el tiempo que se ha tardado en calcularlas. |
| *Entrada* | La estructura de datos que contiene los MEM y el tiempo en que se han calculado. |
| *Salida* | - |

| *imprimeCPC(CCEMTS cad, CMEMTS mod)* | |
|---|---|
| *Objetivos* | Muestra información sobre los PCs hallados. |
| *Entrada* | Las estructuras de datos que contienen las cadenas y los modelos calculados. |
| *Salida* | - |
| *abreModelos()* | |
| *Objetivos* | Abre una pestaña en la que se incluirá la información sobre los MEM. |
| *Entrada* | - |
| *Salida* | - |
| *abreConflictos()* | |
| *Objetivos* | Abre una pestaña en la que se incluirá la información sobre los posibles conflictos. |
| *Entrada* | - |
| *Salida* | - |
| *cierra()* | |
| *Objetivos* | Elimina toda información sobre los cálculos realizados y cierra todas las pestañas abiertas. |
| *Entrada* | - |
| *Salida* | - |
| *actualizaIdioma(ResourceBundle m)* | |
| *Objetivos* | Modifica el idioma de la interfaz. |
| *Entrada* | El objeto encargado de proporcionar el texto en el idioma seleccionado. |
| *Salida* | - |
| *limpiaCCEM()* | |
| *Objetivos* | Elimina todo el texto que se encuentra en la pestaña de las CEM. |
| *Entrada* | - |
| *Salida* | - |
| *limpiaCMEM()* | |
| *Objetivos* | Elimina todo el texto que se encuentra en la pestaña de los MEM. |
| *Entrada* | - |
| *Salida* | - |
| *limpiaCPC()* | |
| *Objetivos* | Elimina todo el texto que se encuentra en la pestaña de los PCs. |
| *Entrada* | - |
| *Salida* | - |

| CPCTS | |
|---|---|
| **Responsabilidades** | |
| Clase control del algoritmo. | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *interfaz(IntCPCTS)* | Objeto encargado de la comunicación con el usuario. |
| *sistema(descSist)* | Descripción del sistema cargado. |
| *cadenas(CCEMTS)* | Estructura que contiene las CEM calculadas. |
| *modelos(CMEMTS)* | Estructura que contiene los MEM calculados. |
| *ficheroGuardar(String)* | Ruta completa al archivo en el que se guardarán los resultados. |
| *mensajes(ResourceBundle)* | Objeto capaz de proporcionar un texto determinado en diferentes idiomas. |
| **Operaciones** | |
| *asignaSistema(descSist s)* | |
| *Objetivos* | Asocia a este objeto la descripción de sistema indicada. |
| *Entrada* | La descripción de sistema con la que se va a trabajar. |
| *Salida* | - |
| *calcularCPC()* | |
| *Objetivos* | Realiza todos los cálculos necesarios para obtener los PCs. |
| *Entrada* | - |
| *Salida* | - |
| *aXml(String fichero)* | |
| *Objetivos* | Almacena los resultados obtenidos en un fichero formato XML. |
| *Entrada* | La ruta completa al fichero en el que se guardarán los datos. |
| *Salida* | - |
| *aTxt(String fichero)* | |
| *Objetivos* | Almacena los resultados obtenidos en un fichero de texto plano. |
| *Entrada* | La ruta completa al fichero en el que se guardarán los datos. |
| *Salida* | - |

| *asignaPanel(JPanel p, ResourceBundle m)* | |
|---|---|
| *Objetivos* | Crea la interfaz del plugin y la integra en la interfaz del programa principal. |
| *Entrada* | El elemento en el que integraremos la interfaz del plugin y el objeto encargado de proporcionar el texto en el idioma seleccionado. |
| *Salida* | - |
| *actualizaIdioma(ResourceBundle m)* | |
| *Objetivos* | Notifica a la interfaz para que actualice el idioma. |
| *Entrada* | El objeto encargado de proporcionar el texto en el idioma seleccionado. |
| *Salida* | - |
| *calculosRealizados():boolean* | |
| *Objetivos* | Indica si se ha realizado o no el cálculo de PCs. |
| *Entrada* | - |
| *Salida* | True si los PCs han sido calculados y false en caso contrario. |
| *actionPerformed(ActionEvent evento)* | |
| *Objetivos* | Captura un evento que se ha producido en la interfaz y ejecuta la orden correspondiente. |
| *Entrada* | El evento reibido. |
| *Salida* | |
| *calcularCadenas()* | |
| *Objetivos* | Realiza los cálculos necesarios para obtener las CEM. |
| *Entrada* | - |
| *Salida* | - |
| *calcularModelos()* | |
| *Objetivos* | Realiza los cálculos necesarios para obtener los MEM. |
| *Entrada* | - |
| *Salida* | - |
| *muestraExcepcion(String excep)* | |
| *Objetivos* | Muestra una excepción en una ventana emergente. |
| *Entrada* | El mensaje a mostrar tal y como se mostrará. |
| *Salida* | |
| *muestraError(String err)* | |
| *Objetivos* | Muestra un mensaje de error. Dicho mensaje dependerá del idioma seleccionado. |
| *Entrada* | La cadena que nos proporcionará el mensaje a mostrar. |
| *Salida* | - |

| *muestraConfirm(String confirmacion):int* | |
|---|---|
| *Objetivos* | Muestra un mensaje de confirmación. Dicho mensaje dependerá del idioma seleccionado. |
| *Entrada* | La cadena que nos proporcionará el mensaje a mostrar. |
| *Salida* | 0 si el usuario selecciona "Sí"; 1 si selecciona "No". |

| **CCEMTS** | |
|---|---|
| **Responsabilidades** | |
| Clase encargada de calcular y almacenar el conjunto de CEMs. | |
| **Operaciones** | |
| *<<creador>> CCEMTS(descSist sistema):CCEMTS* | |
| *Objetivos* | Al invocar la creación de esta clase se llevará a cabo el cálculo de las CEMs. |
| *Entrada* | La descripción del sistema sobre el que se realizarán los cálculos. |
| *Salida* | El conjuto de CEM calculado. |
| *calcularCadenas(descSist HSD)* | |
| *Objetivos* | Extrae de la descripción del sistema una lista de las ecuaciones que éste contiene. Para cada una de ellas, llamará a la función "construirCadena". |
| *Entrada* | La descripción del sistema sobre el que se realizan los cálculos. |
| *Salida* | - |
| *construirCadena(CEMTS cadena, ecuacion R, listaEc disponibles)* | |
| *Objetivos* | Primera fase de la construcción de una cadena |
| *Entrada* | La cadena que estamos construyendo, la ecuación de la que partimos y la lista de ecuaciones que todavía no hemos utilizado. |
| *Salida* | - |
| *justificar(CEMTS cadena, listaVar porJustificar, listaVar justificadas, listaEc disponibles)* | |
| *Objetivos* | Segunda fase de la construcción de la cadena. Es en la que se llevan a cabo la mayor parte de los cálculos.<br>Se trata de una función recursiva. |
| *Entrada* | La cadena que estamos construyendo, la lista de variables que no han sido justificadas, la lista de variables que sí han sido justificadas y la lista de ecuaciones que todavía no hemos utilizado. |
| *Salida* | - |

| *nombrarCadenas()* | |
|---|---|
| *Objetivos* | Recorre la lista de CEMs que se han obtenido asignándoles números enteros de forma consecutiva, con el fin de identificarlas. En versiones anteriores dichos identificadores se asignaban a las CEMs en el momento de añadirlas al CCEM; pero como ahora trabajamos con conjuntos ordenados es necesario realizar esto una vez se han añadido todos los elementos al conjunto. |
| *Entrada* | - |
| *Salida* | - |

| **CMEMTS** | |
|---|---|
| **Responsabilidades** | |
| Clase encargada de calcular y almacenar el conjunto de MEMs. | |
| **Operaciones** | |
| *<<creador>> CMEMTS(CCEMTS cadenas):CMEMTS* | |
| *Objetivos* | Al invocar la creación de esta clase se llevará a cabo el cálculo de los MEMs. |
| *Entrada* | El conjunto de CEMs obtenido en el paso anterior del cálculo de PCs. |
| *Salida* | El conjuto de MEM calculado. |
| *calcularModelos(CCEMTS cadenas)* | |
| *Objetivos* | Llama a la función "construirModelos" para cada una de las CEMs del CCEM proporcionado. |
| *Entrada* | El conjunto de CEMs sobre el que calcularemos las MEMs. |
| *Salida* | - |
| *construirModelos(CEMTS cadena)* | |
| *Objetivos* | Primera fase de la construcción de los modelos. |
| *Entrada* | La cadena a partir de la que estamos buscando sus modelos asociados. |
| *Salida* | - |

| *construirModelo(MEMTS mm, CEMTS SinUsar, listaVar porJustificar, listaVar justificadas)* | |
|---|---|
| *Objetivos* | Segunda fase de la construcción de los modelos. Es en la que se llevan a cabo la mayor parte de los cálculos.<br>Se trata de una función recursiva. |
| *Entrada* | El modelo que tratamos de añadir al CMEM, la cadena a partir de la cual estamos buscando sus modelos asociados (a la que se han eliminado las ecuaciones que ya hemos comprobado), la lista de variables que no han sido justificadas y la lista de variables que sí han sido justificadas. |
| *Salida* | - |
| *contieneCadena(CEMTS cad):boolean* | |
| *Objetivos* | Indica si existe algún MEM en el conjunto que ha sido calculado a partir de la cadena especificada. Ésta será la función utilizada para obtener el conjunto de PCs. |
| *Entrada* | La cadena que queremos buscar. |
| *Salida* | True si hay algún MEM en el conjunto calculado a partir de la CEM especificada y false en caso contrario. |
| *nombrarModelos()* | |
| *Objetivos* | Recorre la lista de MEMs que se han obtenido asignándoles números enteros de forma consecutiva, con el fin de identificarlas. En versiones anteriores dichos identificadores se asignaban a los MEMs en el momento de añadirlas al CMEM; pero como ahora trabajamos con conjuntos ordenados es necesario realizar esto una vez se han añadido todos los elementos al conjunto. |
| *Entrada* | - |
| *Salida* | - |

| **ComparaEc** | |
|---|---|
| **Responsabilidades** | |
| En principio, la clase ecuacion no tiene un método que permita comparar dos instancias suyas. Esta clase proporciona una forma de hacerlo, lo que nos permitirá introducir una ecuación en una estructura ordenada. | |
| **Operaciones** | |
| *compare(ecuacion ec1, ecuacion ec2):int* | |
| *Objetivos* | Comparar dos ecuaciones. |
| *Entrada* | Las dos ecuaciones que se desean comparar. |
| *Salida* | 0 si las ecuaciones son iguales, un valor positivo si la primera es mayor que la segunda y un valor negativo si ocurre lo contrario. |

| ComparaInt | |
|---|---|
| **Responsabilidades** | |
| Esta clase cumple la misma función que la anterior sólo que para las interpretaciones. | |
| **Operaciones** | |
| *compare(interpretacion int1, interpretacion int2):int* | |
| *Objetivos* | Comparar dos interpretaciones. |
| *Entrada* | Las dos interpretaciones que se desean comparar. |
| *Salida* | 0 si las interpretaciones son iguales, un valor positivo si la primera es mayor que la segunda y un valor negativo si ocurre lo contrario. |

| CEMTS | |
|---|---|
| **Responsabilidades** | |
| Clase que representa una CEM. | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *idcadena(int)* | Entero que identifica unívocamente a la CEM. |
| **Operaciones** | |
| *<<creador>> CEMTS():CEMTS* | |
| *Objetivos* | Crear un nuevo objeto de la clase CEMTS. |
| *Entrada* | - |
| *Salida* | El objeto creado. |

| MEMTS | |
|---|---|
| **Responsabilidades** | |
| Clase que representa un MEM. | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *idmodelo(int)* | Entero que identifica unívocamente al MEM. |
| *idcadena(int)* | Identificador de la cadena a partir de la cual se calculó este MEM. |
| *primeraInterpretacion (interpretacion)* | Primera interpretación del MEM. Necesitamos almacenar este valor puesto que, al ordenar los MEMs perderemos esta información que, posteriormente, será necesaria para la representación gráfica. |

| **Operaciones** | |
|---|---|
| *<<creador>> MEMTS(interpretacion primeraInt):MEMTS* | |
| *Objetivos* | Crear un nuevo objeto de la clase MEMTS e inserta en éste la primera interpretación. |
| *Entrada* | La primera interpretación del MEM. |
| *Salida* | El objeto creado. |

## *5.4.3. Diagrama de clases del algoritmo con condiciones*

Continuaremos con el diagrama de clases del algoritmo de CPCs con condiciones.

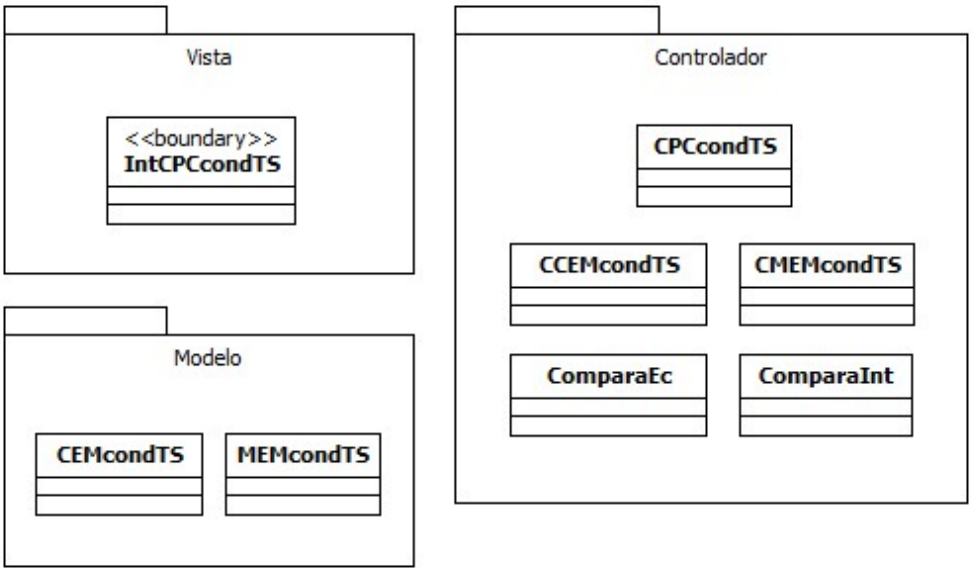



***Figura 5.8: Diagrama de clases del algoritmo con condiciones***

Las clases de este algoritmo se corresponden con las del anterior con algunas diferencias, que expondremos a continuación:

- IntCPCcondTS: contiene los mismos atributos y operaciones que la clase IntCPCTS.
- CPCcondTS: contiene los mismos atributos y operaciones que la clase CPCTS.
- CCEMcondTS: contiene los mismos atributos y operaciones que la clase CCEMTS.
- CMEMcondTS: contiene los mismos atributos y operaciones que la clase CMEMTS.
- ComparaEc y ComparaInt son copias exactas de las clases que se encuentran en el anterior algoritmo.

- CEMcondTS: contiene los mismos atributos y operaciones que la clase CEMTS y añade las siguientes operaciones:

| *getCond():listaCond* | |
|---|---|
| *Objetivos* | Proporcionar una lista con todas las condiciones que afectan a la CEM. |
| *Entrada* | - |
| *Salida* | La lista de condiciones que afectan a la CEM. |
| *esCompatible(listaCond l):boolean* | |
| *Objetivos* | Indica si la condición proporcionada es compatible todas las condiciones de las ecuaciones que forman la CEM. |
| *Entrada* | La condición que queremos comprobar. |
| *Salida* | True si no hay ninguna condición incompatible y false en caso contrario. |

- MEMcondTS: contiene los mismos atributos y operaciones que la clase MEMTS y añade las siguientes operaciones:

| *getCond():listaCond* | |
|---|---|
| *Objetivos* | Proporcionar una lista con todas las condiciones que afectan al MEM. |
| *Entrada* | - |
| *Salida* | La lista de condiciones que afectan al MEM. |
| *esCompatible(listaCond l):boolean* | |
| *Objetivos* | Indica si la condición proporcionada es compatible todas las condiciones de las interpretaciones que forman el MEM. |
| *Entrada* | La condición que queremos comprobar. |
| *Salida* | True si no hay ninguna condición incompatible y false en caso contrario. |

### 5.4.4. Diagrama de clases del visor

Para finalizar mostraremos el diagrama de clases de la nueva herramienta de visualización desarrollada.

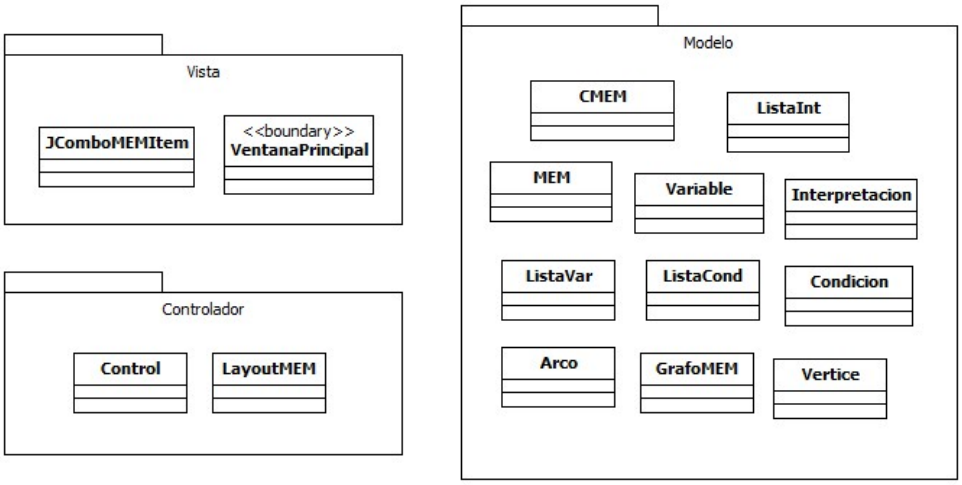



***Figura 5.9: Diagrama de clases del visor***

| **<<boundary>> VentanaPrincipal** | |
|---|---|
| **Responsabilidades** | |
| Clase encargada de realizar la interacción con el usuario del sistema. | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *padre(Control)* | Objeto encargado de ejecutar las acciones enviadas a la interfaz. |
| *loadedCMEM(CMEM)* | Almacena el CMEM que se encuentra cargado actualmente. |
| *selectedMEM(MEM)* | Almacena el MEM que se está representando actualmente. |
| *condiciones(ListaCond)* | Almacena el conjunto de condiciones que afectan al MEM actualmente seleccionado con sus actuales valores (recordemos que el usuario puede modificar libremente dichos valores). |
| *mensajes(ResourceBundle)* | Objeto capaz de proporcionar un texto determinado en diferentes idiomas. |

| **Operaciones** | |
|---|---|
| *<<creador>> VentanaPrincipal(Control v, ResourceBundle m):VentanaPrincipal* | |
| *Objetivos* | Crear un nuevo objeto de la clase VentanaPrincipal. |
| *Entrada* | El objeto que se encagará de ejecutar las acciones y el objeto encargado de proporcionar el texto en el idioma seleccionado. |
| *Salida* | El objeto creado. |
| *jbInit()* | |
| *Objetivos* | Moldear y colocar en su correspondiente lugar todos los elementos de la interfaz (menús, botones, texto, pestañas…). |
| *Entrada* | - |
| *Salida* | - |
| *actualizaIdioma(ResourceBundle m, int langid)* | |
| *Objetivos* | Modifica el idioma de la interfaz. |
| *Entrada* | El objeto encargado de proporcionar el texto en el idioma seleccionado y un entero que identifica a dicho idioma. |
| *Salida* | - |
| *cargaCMEM(CMEM cmem)* | |
| *Objetivos* | Modificará el CMEM que se encuentra cargado actualmente. |
| *Entrada* | El CMEM que deseamos cargar. |
| *Salida* | - |
| *addCPCTree()* | |
| *Objetivos* | Añade a la interfaz información sobre el CMEM cargado en forma de lista desplegable. |
| *Entrada* | - |
| *Salida* | - |
| *cambiaMEM(MEM mem)* | |
| *Objetivos* | Modificará el MEM que estamos visualizando actualmente, mostrando el nuevo MEM que deseamos visualizar y actualizando la interfaz de la forma que sea necesaria. |
| *Entrada* | El MEM que se desee visualizar. |
| *Salida* | - |

| *dibujaMEM(boolean toPDF)* | |
|---|---|
| *Objetivos* | Esta operación es la encargada de dibujar el hipergrafo dirigido que representa el MEM seleccionado. Podrá dibujarlo de dos formas: una preparada para mostrarla en la interfaz y otra para guardarla en un archivo PDF. |
| *Entrada* | Si la entrada tiene valor true el hipergrafo se dibujará adaptado para guardarse en un archivo PDF; si es false, se adaptará para presentarlo en la interfaz. |
| *Salida* | - |
| *addCondToolbar()* | |
| *Objetivos* | Añade los controles necesarios para manipular las condiciones. |
| *Entrada* | - |
| *Salida* | - |
| *guardaPDF(String nomFichero)* | |
| *Objetivos* | Almacena la representación gráfica del MEM actualmente seleccionado en un archivo PDF. |
| *Entrada* | La ruta completa del archivo en el que se almacenará la representación gráfica del MEM. |
| *Salida* | - |
| *actualizaCond()* | |
| *Objetivos* | Modificará la representación del MEM que se está visualizando de acuerdo con los cambios que el usuario haya introducido en las condiciones. |
| *Entrada* | - |
| *Salida* | - |
| *acerca()* | |
| *Objetivos* | Acercará la vista de la representación gráfica del MEM que se está visualizando. |
| *Entrada* | - |
| *Salida* | - |
| *aleja()* | |
| *Objetivos* | Alejará la vista de la representación gráfica del MEM que se está visualizando. |
| *Entrada* | - |
| *Salida* | - |

| Control | |
|---|---|
| **Responsabilidades** | |
| Clase encargada del control de la funcionalidad de toda la aplicación. | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *frame(VentanaPrincipal)* | Almacena el nombre del idioma actualmente seleccionado. |
| *mensajes(ResourceBundle)* | Objeto capaz de proporcionar un texto determinado en diferentes idiomas. |
| *currentLocale(Locale)* | Identifica el lenguaje actualmente seleccionado. |
| *langdir(String)* | Almacena la ruta al directorio en el que se encuentran los archivos de lenguaje. |
| **Operaciones** | |
| *<<creador>> Control (): Control* | |
| *Objetivos* | Crear un nuevo objeto de la clase Control. |
| *Entrada* | - |
| *Salida* | El objeto creado. |
| *<<creador>> Control (String resultados): Control* | |
| *Objetivos* | Crear un nuevo objeto de la clase Control y carga un fichero de resultados. |
| *Entrada* | La ruta completa a los resultados que se van a cargar. |
| *Salida* | El objeto creado. |
| *main(String[] args)* | |
| *Objetivos* | Método que se ejecuta al iniciar la aplicación. |
| *Entrada* | Aunque acepta varios argumentos, sólo se tendrá en cuenta el primero, que corresponderá a la ruta completa a un fichero de resultados que se cargaría automáticamente al iniciar la aplicación. |
| *Salida* | - |
| *eventoIdioma(ActionEvent evento)* | |
| *Objetivos* | Indica a la interfaz que actualice la interfaz de acuerdo al idioma indicado. |
| *Entrada* | El idioma al que deseemos cambiar la interfaz de la aplicación. |
| *Salida* | - |
| *eventoAniadirIdioma()* | |
| *Objetivos* | Indica a la interfaz que actualice la interfaz de acuerdo al idioma indicado. |
| *Entrada* | El idioma al que deseemos cambiar la interfaz de la aplicación. |
| *Salida* | - |

| *eventoCargarResultados()* | |
|---|---|
| *Objetivos* | Modifica el CMEM que se encuentra cargado actualmente. |
| *Entrada* | El CMEM que deseamos cargar. |
| *Salida* | - |
| *cargaControles(String nomFichero)* | |
| *Objetivos* | Modifica el CMEM que se encuentra cargado actualmente. |
| *Entrada* | El CMEM que deseamos cargar. |
| *Salida* | - |
| *eventoCambiarMEM()* | |
| *Objetivos* | Indica a la interfaz que muestre el MEM indicado. |
| *Entrada* | El MEM que se desee visualizar. |
| *Salida* | - |
| *eventoGuardarPDF()* | |
| *Objetivos* | Almacena la representación gráfica del MEM actualmente seleccionado en un archivo imprimible (pdf, ps…). |
| *Entrada* | La ruta completa del archivo en el que se almacenará la representación gráfica del MEM. |
| *Salida* | - |
| *eventoActualizarCond()* | |
| *Objetivos* | Indica a la interfaz que modifique la representación del MEM que se está visualizando de acuerdo con los cambios que el usuario haya introducido en las condiciones. |
| *Entrada* | - |
| *Salida* | - |
| *eventoAcercar()* | |
| *Objetivos* | Indica a la interfaz que acerque la vista de la representación gráfica del MEM que se está visualizando. |
| *Entrada* | - |
| *Salida* | - |
| *eventoAlejar()* | |
| *Objetivos* | Indica a la interfaz que aleje la vista de la representación gráfica del MEM que se está visualizando. |
| *Entrada* | - |
| *Salida* | - |

| *eventoAyuda()* | |
|---|---|
| *Objetivos* | Muestra la ayuda de la aplicación. |
| *Entrada* | - |
| *Salida* | - |
| *eventoInfo()* | |
| *Objetivos* | Muestra información sobre la aplicación. |
| *Entrada* | - |
| *Salida* | - |
| *muestraInfo(String info)* | |
| *Objetivos* | Muestra un mensaje de información. Dicho mensaje dependerá del idioma seleccionado. |
| *Entrada* | La cadena que nos proporcionará el mensaje a mostrar. |
| *Salida* | - |
| *muestraExcepcion(String excep)* | |
| *Objetivos* | Muestra una excepción en una ventana emergente. |
| *Entrada* | El mensaje a mostrar tal y como se mostrará. |
| *Salida* | |
| *muestraError(String err)* | |
| *Objetivos* | Muestra un mensaje de error. Dicho mensaje dependerá del idioma seleccionado. |
| *Entrada* | La cadena que nos proporcionará el mensaje a mostrar. |
| *Salida* | - |
| *muestraConfirm(String confirmacion):int* | |
| *Objetivos* | Muestra un mensaje de confirmación. Dicho mensaje dependerá del idioma seleccionado. |
| *Entrada* | La cadena que nos proporcionará el mensaje a mostrar. |
| *Salida* | 0 si el usuario selecciona "Sí"; 1 si selecciona "No". |

| **CMEM** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa los posibles conflictos que queremos visualizar en forma de un CMEM. | |
| **Operaciones** | |
| *desdeXml(String nomFichero)* | |
| *Objetivos* | Carga un CMEM partiendo del fichero de resultados indicado. |
| *Entrada* | La ruta completa al fichero donde se encuentran los resultados que se desean visualizar. |
| *Salida* | - |

| *desdeXmlAnt(String nomFichero)* | |
|---|---|
| *Objetivos* | Carga un CMEM partiendo del fichero de resultados indicado en el formato antiguo. |
| *Entrada* | La ruta completa al fichero donde se encuentran los resultados que se desean visualizar. |
| *Salida* | - |

| **MEM** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa un MEM concreto. | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *id(int)* | Entero que identifica unívocamente al MEM. |
| *ecCadena(String)* | Ecuaciones que componen la CEM de la que se obtuvo este MEM separadas por espacios. |
| *variables(ListaVar)* | Conjunto de variables que aparecen en el MEM. |
| *primeraInt(Interpretacion)* | Primera interpretación del MEM. Necesaria para conocer el nodo discrepancia. |
| *condiciones(ListaCond)* | Lista de condiciones que afectan al MEM. |
| **Operaciones** | |
| *<<creador>> MEM(Interpretacion interpretacion):MEM* | |
| *Objetivos* | Crear un nuevo objeto de la clase MEM y añade la primera interpretación |
| *Entrada* | La primera interpretación del MEM. |
| *Salida* | El objeto creado. |
| *interpEsCabeza(Variable var):ListaInt* | |
| *Objetivos* | Obtener una lista de interpretaciones de este MEM de las que la variable pasada como argumento es cabeza. |
| *Entrada* | La variable de la cual queremos conocer las interpretaciones en las que es cabeza. |
| *Salida* | La lista de interpretaciones de este MEM en las que la variable es cabeza. |
| *anadeCond(Condicion condicion)* | |
| *Objetivos* | Añade una condición a la lista de condiciones del MEM. |
| *Entrada* | La condición a añadir. |
| *Salida* | - |

| Interpretacion | |
|---|---|
| **Responsabilidades** | |
| Clase que representa una interpretación. | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *nombre(String)* | Nombre que identifica unívocamente a la interpretación. |
| *tipo(String)* | Tipo de la interpretación (integral ó derivada). |
| *cabeza(Variable)* | Variable cabeza de la interpretación. |
| *cola(ListaVar)* | Conjunto de variables que forman la cola de la interpretación. |
| *condiciones(ListaCond)* | Conjunto de condiciones que afectan a la interpretación. |
| **Operaciones** | |
| *<<creador>>Interpretacion(String n):Interpretacion* | |
| *Objetivos* | Crear un nuevo objeto de la clase Interpretacion. |
| *Entrada* | El nombre de la interpretación. |
| *Salida* | El objeto creado. |
| *anadeVarCola(Variable variable)* | |
| *Objetivos* | Añade una variable a la cola de la interpretación. |
| *Entrada* | La variable a añadir. |
| *Salida* | - |
| *anadeCond(Condicion condicion)* | |
| *Objetivos* | Añade una condición a la lista de condiciones de la interpretación. |
| *Entrada* | La condición a añadir. |
| *Salida* | - |

| Variable | |
|---|---|
| **Responsabilidades** | |
| Clase que representa a una variable del sistema. | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *nombre(String)* | Nombre que identifica unívocamente a la variable. |
| *observada(boolean)* | Indica si la variable es o no observada. |
| **Operaciones** | |
| *<<creador>>Variable(String nombre, boolean observada):Variable* | |
| *Objetivos* | Crear un nuevo objeto de la clase Variable. |
| *Entrada* | El nombre de la variable y si es o no observada. |
| *Salida* | El objeto creado. |

| **Condicion** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa una condición del sistema. | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *nombre(String)* | Nombre que identifica unívocamente a la condición. |
| *valor(String)* | Valor de la condición (verdadero, falso ó indiferente) |
| **Operaciones** | |
| *<<creador>>Condicion(String n, String v):Condicion* | |
| *Objetivos* | Crear un nuevo objeto de la clase Variable. |
| *Entrada* | El nombre y el valor de la condición. |
| *Salida* | El objeto creado. |

| **GrafoMEM** | |
|---|---|
| **Responsabilidades** | |
| Clase que contiene el grafo que representa un MEM. | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *root(Vertice)* | Vértice raíz del grafo. |
| **Operaciones** | |
| *crearGrafo(MEM mem)* | |
| *Objetivos* | Crea el grafo que representa el MEM indicado. |
| *Entrada* | El MEM del cual queremos obtener el grafo. |
| *Salida* | - |
| *addVertexForce(Vertice v):Vertice* | |
| *Objetivos* | Añade un nuevo vértice al grafo aunque éste ya exista. Para ello, si el vértice está repetido, se le añadirá el símbolo "#" al final de su identificador y se volverá a intentar la inserción. Esto se repetirá hasta que la inserción tenga éxito. |
| *Entrada* | El vértice a añadir. |
| *Salida* | El vértice insertado con su ID modificada en caso de que ya exista otro vértice asociado a la misma variable en el grafo. |
| *getLevel(int lvl):Vector<Vertice>* | |
| *Objetivos* | Obtiene el conjunto de vértices que se encuentran en un nivel del grafo. |
| *Entrada* | El nivel del que queremos obtener el conjunto de vértices. |
| *Salida* | El conjunto de vértices que se encuentran en el nivel indicado. |

| *getIntEsCabeza(Vertice v):ListaInt* | |
|---|---|
| *Objetivos* | Obtiene una lista de interpretaciones que se corresponden con los arcos que llegan a un vértice determinado. |
| *Entrada* | El vértice del que queremos obtener la lista de interpretaciones. |
| *Salida* | La lista de interpretaciones que apuntan al vértice indicado. |
| *getEdge(Arco arco):Arco* | |
| *Objetivos* | Obtiene un arco del grafo que se corresponda con el pasado como argumento (o null en caso de que el grafo no contenga el arco). |
| *Entrada* | Un arco igual al que queremos obtener (se consideran iguales dos arcos que tengan el mismo valor en sus atributos "interpretacion", "count" y "virtual"). |
| *Salida* | El arco que nos interesa (o null en caso de que el grafo no contenga el arco). |

| **Vertice** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa un vértice del grafo. Tendremos dos tipos de vértices: los vértices “estándar”, que son los usados para representar variables, y los vértices “virtuales” que serán usados para representar arcos “virtuales” (llamaremos así a los que se usan para unir varios arcos de una misma interpretación) | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *id(String)* | Nombre que identifica unívocamente al vértice. Inicialmente coincidirá con el nombre de la variable a la que representa; aunque puede ser modificado en caso de que una misma variable aparezca en el grafo más de una vez. |
| *variable(Variable)* | Variable a la que representa el vértice. |
| *nombreInterpretacion(String)* | Sólo usada para los vértices “virtuales”: Nombre de la interpretación a la que pertenece. |
| *nivel(int)* | Nivel que ocupará el vértice en el grafo. En caso de los nodos virtuales, este valor será -1 y -2 para cada par de vértices que forman la línea usada para unir los arcos de una misma interpretación. |
| *virtual(Boolean)* | Valor lógico que informa de si el vértice es “virtual” (true) o “estándar” (false). |

| **Operaciones** | |
|---|---|
| *<<creador>>Vertice(Variable variable, int lvl):Vertice* | |
| *Objetivos* | Crea un nuevo objeto de la clase Vertice. Con este método de creación obtendremos un vértice "estándar". |
| *Entrada* | La variable a la que representa el vértice y el nivel que ocupa en el hipergrafo. |
| *Salida* | El objeto creado. |
| *<<creador>>Vertice(String interpretacion, Boolean inicio):Vertice* | |
| *Objetivos* | Crea un nuevo objeto de la clase Vertice. Con este método de creación obtendremos un vértice "virtual". |
| *Entrada* | El nombre de la interpretación a la que pertenece el vértice y un valor lógico que indica si el vértice es el origen o el destino del arco "virtual". |
| *Salida* | El objeto creado. |

| **Arco** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa un arco del grafo. Al igual que con los vértices tendremos dos tipos de arcos: "estándar" que son los que unen dos vértices que representan variables; y "virtuales" que son los que se usan para unir varios arcos de una misma interpretación. | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *interpretación (Interpretacion)* | Interpretación a la que pertenece el arco. |
| *count(int)* | Entero usado para identificar cada arco, diferenciándolo de otros arcos que representan una misma interpretación. Es decir: si una interpretación tiene tres variables en la cola, tendrá tres arcos asociados a ella (aparte de los virtuales). Asignaremos a estos arcos los números 0, 1 y 2 para diferenciarlos entre ellos<br>En caso de los arcos "virtuales", este parámetro tendrá sólo 2 valores posibles: (-1) y (-2). En las interpretaciones que se encuentran en el mismo nivel, en el grafo, asignaremos estos dos valores de forma alternativa a los arcos virtuales. Esto se usará en el dibujado del grafo para que no se solapen las etiquetas de las interpretaciones. |
| *virtual(Boolean)* | Si este parámetro es true significará que se trata de un arco "virtual". Llamaremos arcos "virtuales" a las líneas que unen varios arcos que representan una misma interpretación. |

| **Operaciones** | |
|---|---|
| *<<creador>>Arco(Interpretacion interp, boolean virt, int cont):Arco* | |
| *Objetivos* | Crea un nuevo objeto de la clase Arco. |
| *Entrada* | La interpretación a la que pertenece; un valor lógico que indica si es o no virtual y el entero que sirve para identificar el arco diferenciándolo de otros que pertenecen a la misma interpretación. |
| *Salida* | El objeto creado. |
| *uniqID():String* | |
| *Objetivos* | Crea una cadena que identifica unívocamente al arco. |
| *Entrada* | - |
| *Salida* | Una cadena que identifica unívocamente al arco. |

| **ListaInt** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa una lista de interpretaciones. | |
| **Operaciones** | |
| *get(String nombreInt):Interpretacion* | |
| *Objetivos* | Obtiene una interpretación cuyo nombre coincida con la cadena pasada como argumento o null si la lista no contiene ninguna interpretación coincidente. |
| *Entrada* | El nombre de la interpretación que buscamos. |
| *Salida* | La interpretación buscada (o null en caso de que no se encuentre). |

| **ListaVar** | |
|---|---|
| **Responsabilidades** | |
| Clase que representa una lista de variables. | |
| **Operaciones** | |
| *get(String nombreVar):Variable* | |
| *Objetivos* | Obtiene una variable cuyo nombre coincida con la cadena pasada como argumento o null si la lista no contiene ninguna variable coincidente. |
| *Entrada* | El nombre de la variable que buscamos. |
| *Salida* | La variable buscada (o null en caso de que no se encuentre). |

| ListaCond | |
|---|---|
| **Responsabilidades** | |
| Clase que representa una lista de condiciones. | |
| **Operaciones** | |
| *compatibles(ListaCond cond):boolean* | |
| *Objetivos* | Comprueba si una segunda lista de condiciones es compatible con la lista actual. |
| *Entrada* | La lista de condiciones cuya compatibilidad queremos comprobar. |
| *Salida* | True si las listas de condiciones son compatibles y false en caso contrario. |
| *contieneContraria(Condicion c):boolean* | |
| *Objetivos* | Comprueba existe alguna condición en la lista que no es compatible con la condición pasada como argumento. |
| *Entrada* | La condición cuya compatibilidad queremos comprobar. |
| *Salida* | True si encontramos alguna condición incompatible y false en caso contrario. |

| LayoutMEM | |
|---|---|
| **Responsabilidades** | |
| Clase encargada de calcular la posición de cada elemento de un grafo de la clase GrafoMEM en el espacio para poder ser dibujado. | |
| **Atributos** | |
| **Nombre(Tipo)** | **Descripción** |
| *size(Dimension)* | Tamaño que ocupará la representación del grafo. |
| *graph(GrafoMEM)* | Grafo que se va a representar. |
| *basePositions(Map<Vertice, Integer>)* | Este atributo contiene una asociación entre cada vértice y el espacio horizontal que ocupa el subárbol que cuelga de él. |
| *locations(Map<Vertice, Point2D>)* | Este atributo contiene una asociación entre cada vértice y su posición. |
| *alreadyDone(Set<Vertice>)* | Lista de vértices de los cuales ya se ha calculado su posición. |
| *distX(int)* | Distancia horizontal entre dos vértices contiguos. |
| *distY(int)* | Altura de cada nivel del grafo. |
| *m_currentPoint(Point)* | Punto actual (en general, corresponderá con el último punto en el que hemos situado un vértice). |

| **Operaciones** | |
|---|---|
| *<<creador>> LayoutMEM(GrafoMEM g, int distx, int disty):LayoutMEM* | |
| *Objetivos* | Crea un objeto de la clase LayoutMEM. |
| *Entrada* | El grafo que se va a dibujar, la distancia base entre dos vértices en el eje horizontal y la altura de cada nivel del grafo. |
| *Salida* | El objeto creado. |
| *buildMEM()* | |
| *Objetivos* | Esta función lleva a cabo la colocación de los vértices "estándar" del grafo. |
| *Entrada* | - |
| *Salida* | - |
| *buildMEM(Vertice v, int x)* | |
| *Objetivos* | Fija la posición del vértice indicado, calculando de forma recursiva la posición de sus vértices sucesores. |
| *Entrada* | El vértice del cual vamos a calcular su posición y la posición actual en la que nos encontramos en el eje de las X (horizontal). |
| *Salida* | - |
| *calculateDimensionX(Vertice v):int* | |
| *Objetivos* | Esta función calcula la anchura del subárbol que cuelga de un vértice. |
| *Entrada* | El vértice del cual nos interesa calcular la anchura del subárbol. |
| *Salida* | La anchura del subárbol. |
| *setCurrentPositionFor(Vertice vertex)* | |
| *Objetivos* | Fija la posición de un vértice al punto actual. |
| *Entrada* | El vértice cuya posición se va a fijar. |
| *Salida* | - |
| *drawAndOrArcs()* | |
| *Objetivos* | Esta función lleva a cabo la colocación de los vértices "virtuales" del grafo. |
| *Entrada* | - |
| *Salida* | - |
| *getRootLocation():Point2D* | |
| *Objetivos* | Obtiene las coordenadas del vértice raíz. |
| *Entrada* | |
| *Salida* | Las coordenadas del vértice raíz. |

| *traslate(Double x, Double y)* | |
|---|---|
| *Objetivos* | Desplaza todos los vértices del grafo. |
| *Entrada* | Desplazamiento horizontal. |
| *Salida* | Desplazamiento vertical. |
| *setLocation(Vertice v, Point2D location)* | |
| *Objetivos* | Fija la localización de un vértice. |
| *Entrada* | El vértice del que vamos a fijar la localización. |
| *Salida* | La localización que queremos dar al vértice. |

# 5.5. Modelo de interacción

En este apartado detallaremos las interacciones que ocurren entre las clases que hemos visto en el apartado anterior. Al igual que en el modelo de análisis, veremos cómo interaccionan las clases al llevarse a cabo cada uno de los casos de uso.

## *5.5.1. Escenarios de la aplicación de CPCs*

### *5.5.1.1. Escenario de Cargar sistema*

*(Ver figura 5.10)*. Este escenario comenzará con una petición del usuario de cargar un sistema. Dicha petición será pasada a un objeto de tipo Control mediante la interfaz de usuario, representada por el objeto de tipo intPrincipal. El objeto Control creará una ventana (objeto tipo JFileChoser) que se encargará de obtener del usuario la localización del archivo en el que se encuentra el sistema. A continuación, creará un nuevo objeto de tipo descSist (al que llamaremos sistema), que es el que almacenará toda la estructura del sistema.

Llegados a este punto, este escenario puede continuar de dos formas distintas:

- En caso de que el archivo del sistema esté en formato XML, pasaremos al escenario Desde XML.
- En caso de que el archivo esté en texto plano, pasaremos al escenario Desde txt.

#### *5.5.1.1.1. Escenario Desde XML*

*(Ver figura 5.11)*. Al comenzar este escenario tendremos un nuevo objeto de tipo descSist creado (llamado sistema) y la ruta al archivo del sistema que vamos a cargar.

El objeto sistema contendrá lo siguiente:

- Una lista con las variables observadas. Éstas se irán añadiendo según nos las vayamos encontrando.

- Una lista con las variables no observadas, que se añadirán de igual forma que las anteriores.
- Un conjunto de componentes.

En esta versión de archivo de entrada, los componentes están formados por un conjunto de ecuaciones; pero en nuestro modelo, los componentes están formados por modelos que, a su vez, están formados por ecuaciones. Por esto, para cada componente crearemos un modelo. Posteriormente, obtendremos toda la información sobre cada ecuación del componente y la añadiremos al modelo creado. Al finalizar, dicho modelo se añadirá al componente. Siguiendo con la jerarquía de elementos del sistema, las ecuaciones contendrán un conjunto de variables (observadas y no observadas), interpretaciones y condiciones.

Por último, las interpretaciones también podrán contener condiciones.

#### *5.5.1.1.2. Escenario Desde txt*

*(Ver figura 5.12)*. En el caso de los archivos de sistema en formato de texto plano sólo tendremos información sobre ecuaciones (y sus respectivas interpretaciones), por lo que crearemos un solo componente que contendrá un solo modelo en el que se incluirán todas las ecuaciones.

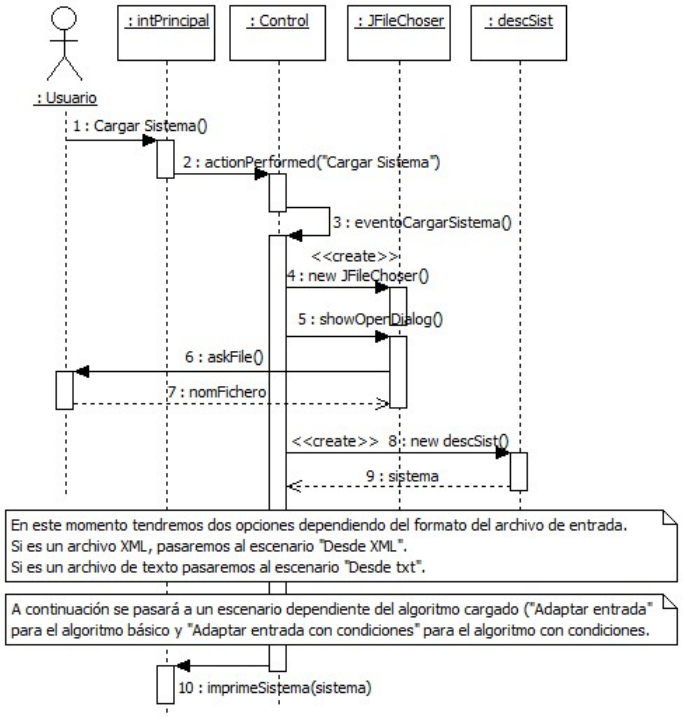



***Figura 5.10: Escenario de Cargar sistema***

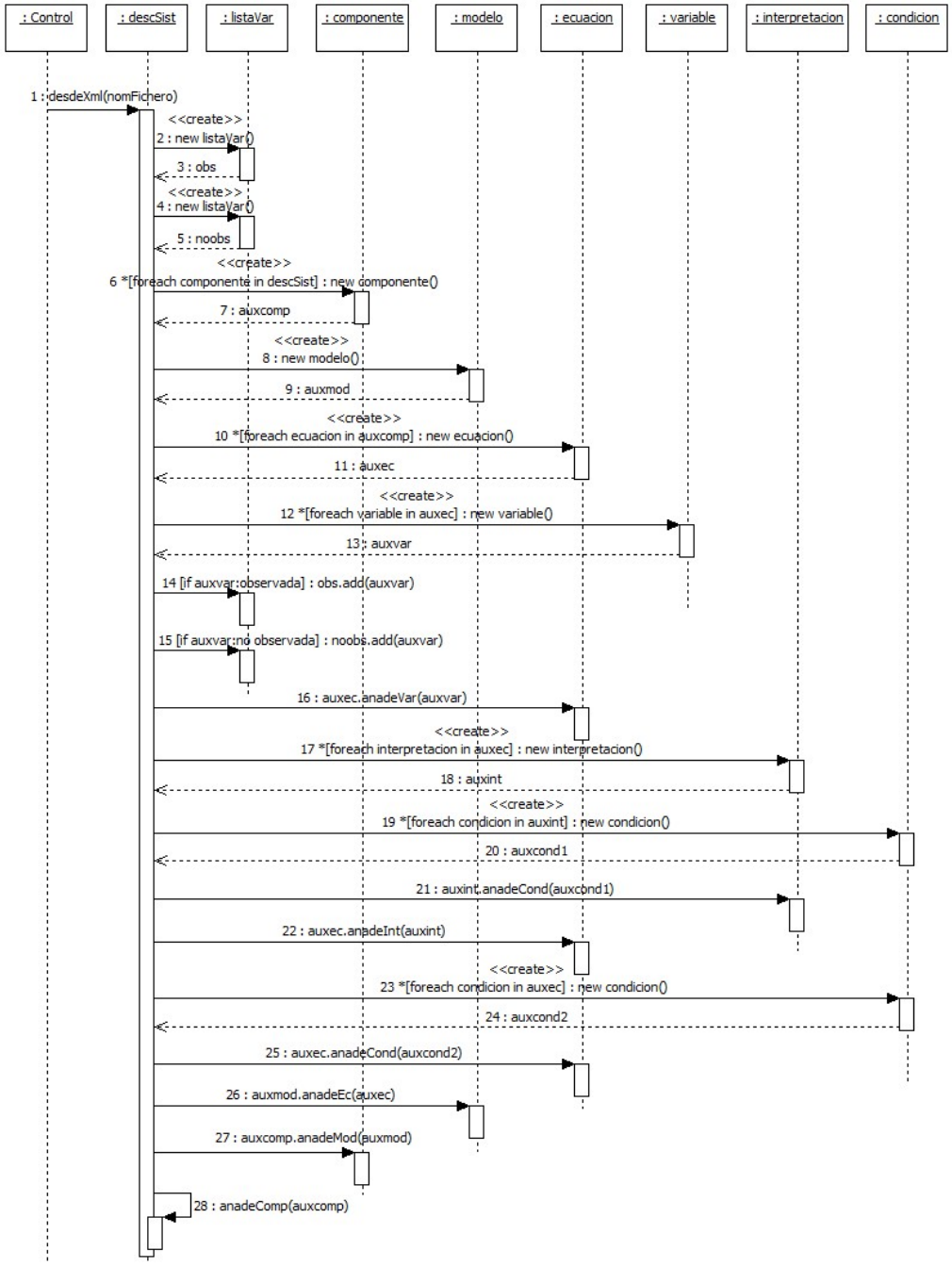



***Figura 5.11: Escenario de Cargar sistema desde XML***

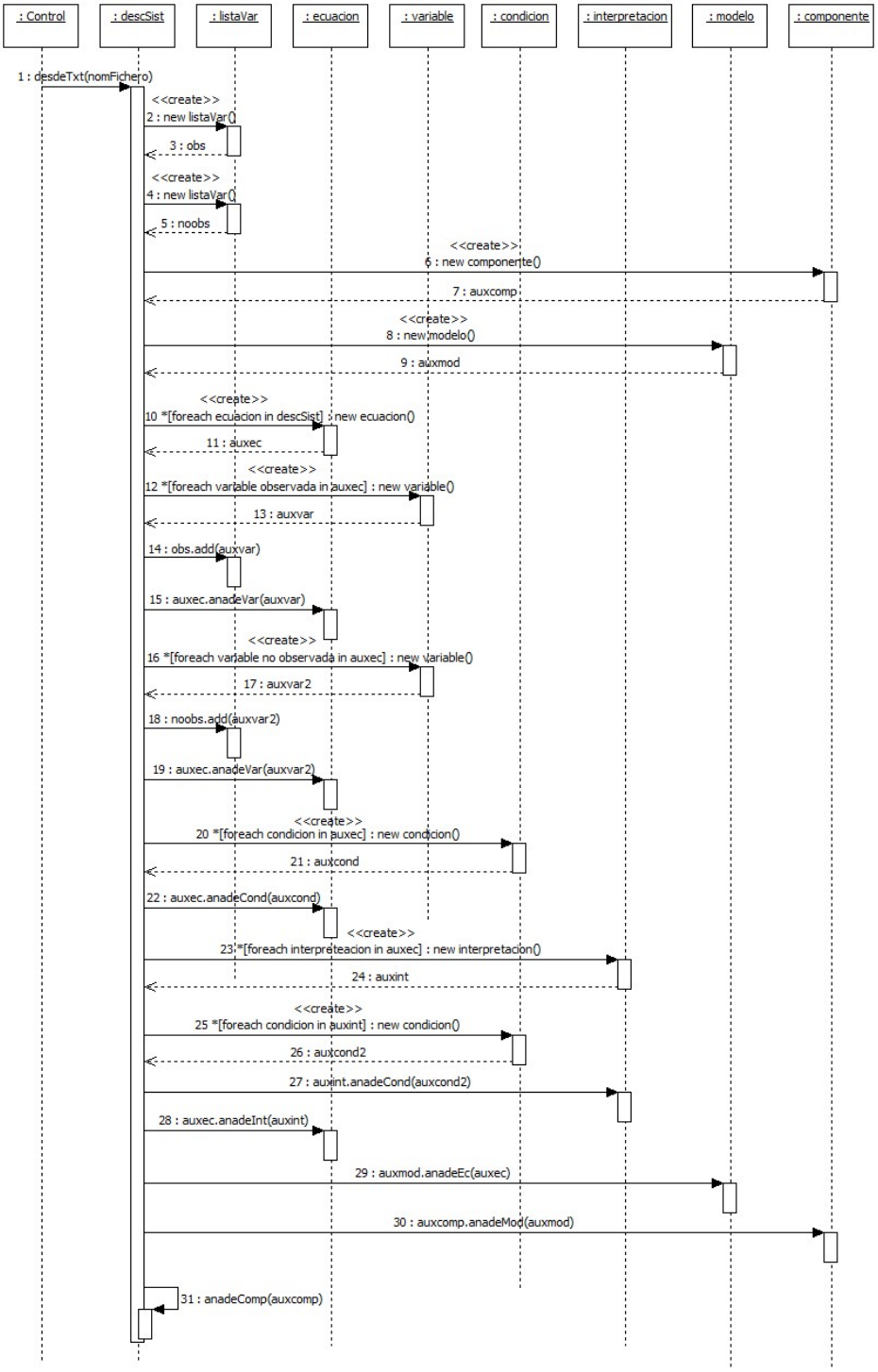



***Figura 5.12: Escenario de Cargar sistema desde txt***

## 5.5.2. Escenarios del algoritmo básico de CPCs

### 5.5.2.1. Escenario de Adaptar entrada

*(Ver figura 5.13)*. Este escenario ocurre una vez tengamos cargado de forma satisfactoria un algoritmo y un sistema.

Básicamente se proporcionará al objeto Control del algoritmo (que será de la clase CPCTS en este caso) la información del sistema.

### 5.5.2.2. Escenario de Calcular cadenas

*(Ver figura 5.14)*. Este escenario comienza cuando el usuario solicita calcular el conjunto de cadenas.

Para ello se debe haber completado con éxito el escenario de Adaptar entrada.

Como siempre, la orden llegará al objeto de la clase Control (en este caso dicha clase se denomina CPCTS) mediante la interfaz de usuario (llamada IntCPCTS).El proceso de cálculo de cadenas se iniciará con la función de creación del objeto de tipo CCEMTS.

El algoritmo de construcción de las cadenas es algo complejo, por lo que se verá con más detalle en el capítulo de Modelo de implementación. Sólo señalaremos aquí que el objeto de tipo CCEMTS estará compuesto por varios objetos CEMTS (cadenas evaluables minimales), las cuales estarán formadas por varias ecuaciones.

Finalizado el cálculo, el objeto CPCTS enviará la información a IntCPCTS para que muestre los resultados al usuario.

### 5.5.2.3. Escenario de Calcular modelos

*(Ver figura 5.15)*. Para este escenario es requisito indispensable haber completado con éxito el escenario de Calcular cadenas.

Este escenario es similar al anterior, solo que ahora crearemos un objeto de tipo CMEMTS, que estará formado por varios objetos tipo MEMTS (modelos evaluables minimales), que se habrán calculado a partir de un objeto tipo CEMTS y constarán de un conjunto de interpretaciones.

### 5.5.2.4. Escenario de Calcular conflictos

*(Ver figura 5.16)*. Este escenario requiere que se haya completado con éxito el escenario de Cálculo de modelos.

Tal como se explicó en la introducción, un PC es básicamente una cadena que tiene algún modelo asociado. Puesto que el objeto CPCTS ya tiene información sobre las cadenas y los modelos (que ya han sido previamente calculados) sólo tiene que

proporcionar esta información a la interfaz, que se limitará a mostrar las cadenas que cumplan la condición que hemos mencionado.

#### *5.5.2.5. Escenario de Generar salida*

*(Ver figura 5.17)*. Este escenario requiere que se haya completado con éxito el escenario Calcular CPC.

Para llevar a cabo este escenario necesitamos obtener dos datos del usuario:

- El formato en el que se guardará el archivo: se obtendrá gracias a un objeto tipo JOptionPane.
- La ruta del archivo donde se almacenarán los resultados: se usará un objeto de tipo JFileChoser.

Con esta información y la almacenada en el objeto de tipo CPCTS fruto de los cálculos realizados estaremos listos para realizar el guardado.

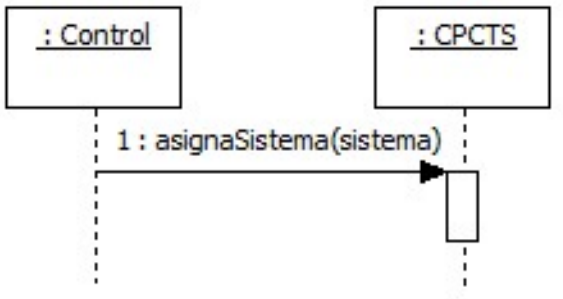



***Figura 5.13: Escenario de Adaptar entrada***

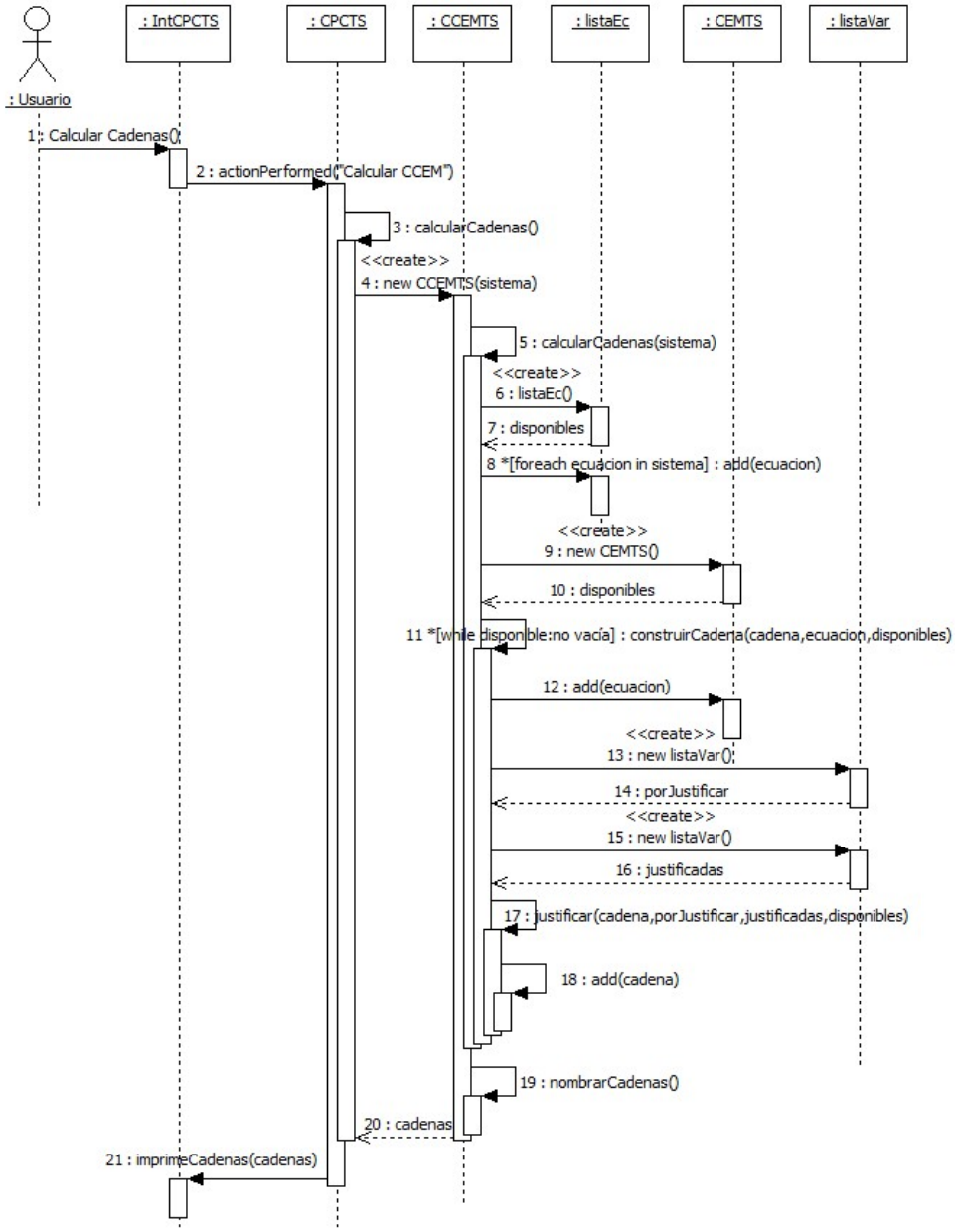



***Figura 5.14: Escenario de Calcular cadenas***

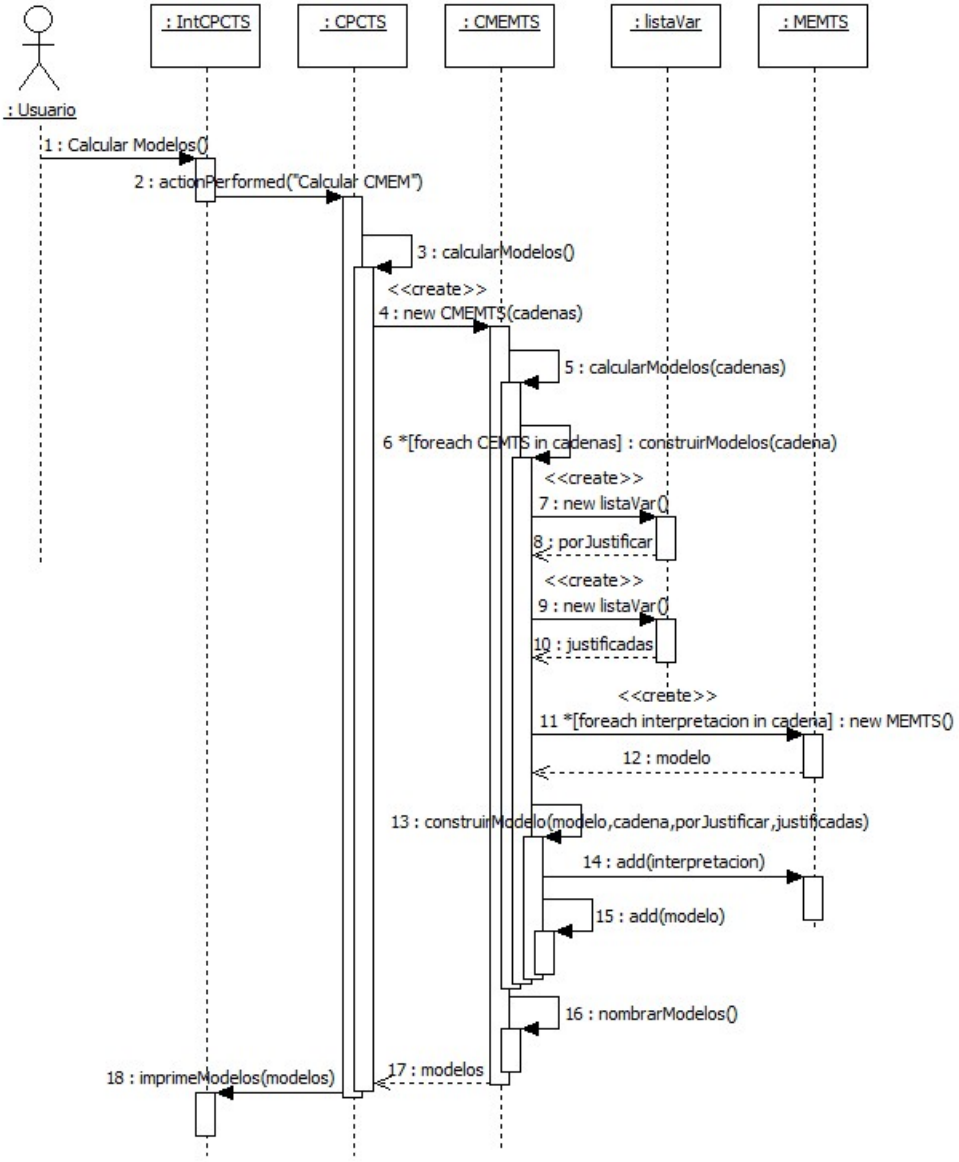



***Figura 5.15: Escenario de Calcular modelos***

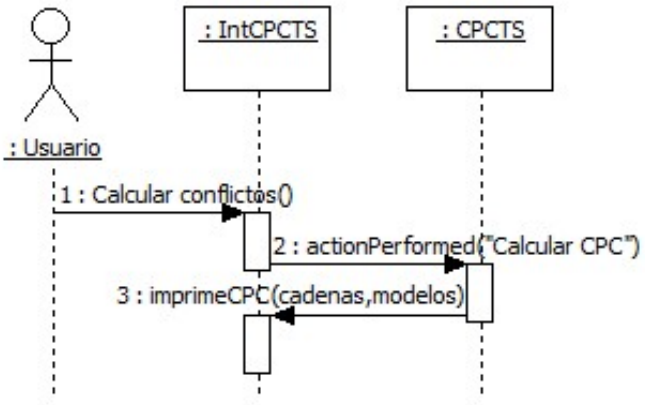



***Figura 5.16: Escenario de Calcular conflictos***

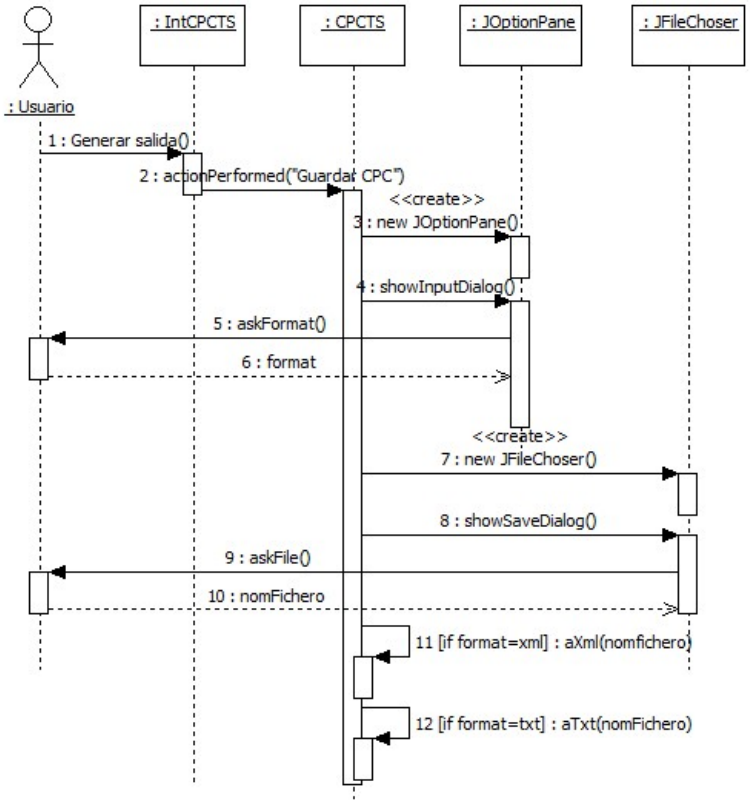



***Figura 5.17: Escenario de Generar salida***

## 5.5.3. Escenarios del algoritmo de CPCs con condiciones

Los escenarios del algoritmo con condiciones son similares a los del algoritmo básico. La única diferencia es que, a la hora de calcular las cadenas y modelos, antes de añadir cada elemento que forma dichas estructuras se comprobará que las condiciones del elemento a añadir son compatibles con las de los elementos ya añadidos.

## 5.5.4. Escenarios de la aplicación Visor

### 5.5.4.1. Escenario de Cargar resultados

*(Ver figura 5.18)*. Este escenario es similar al de Cargar sistema de la aplicación de CPCs. Ahora, el objeto que contendrá la información que vamos a cargar será de tipo CMEM y, como antes, la carga dependerá del formato del archivo de entrada (ambos serán archivos XML, pero con DTDs distintas). Una vez obtenida la información, se mostrará ésta al usuario. Para ello, primero mostraremos la información general del conjunto de MEMs cargado (mediante la instrucción cargaCMEM) y posteriormente mostraremos el primer MEM del conjunto.

#### 5.5.4.1.1. Formato nuevo

*(Ver figura 5.19)*. En este formato, lo primero que debemos hacer es almacenar en sendas listas el conjunto de variables e interpretaciones que aparecen en el sistema del cual se han obtenido los resultados con los que vamos a trabajar. Con esa información estaremos en disposición de leer y almacenar el conjunto de MEMs, con sus respectivas interpretaciones y condiciones.

#### 5.5.4.1.2. Formato antiguo

*(Ver figura 5.20)*. El formato antiguo ofrece menos información, pero el proceso de lectura es más sencillo; puesto que no requiere extraer la información de las variables y condiciones de la descripción completa del sistema, como ocurría en el caso anterior.

Este formato no admite condiciones.

### 5.5.4.2. Escenario de Seleccionar idioma

*(Ver figura 5.21)*. En este escenario la clase Control creará un objeto de tipo ResourceBundle con el idioma seleccionado como argumento. Este objeto tiene la capacidad de obtener mensajes en el idioma indicado partiendo de un código que será común a todos los idiomas.

Gracias a este objeto podremos notificar a la interfaz para que se actualice con el idioma seleccionado.

#### *5.5.4.3. Escenario de Añadir idioma*

*(Ver figura 5.22)*. En este escenario básicamente nos limitaremos a copiar un fichero con la descripción de un nuevo idioma (localizado en una ubicación que nos proporcionará el usuario) a la ubicación en la que el programa almacena los idiomas que tiene disponibles.

#### *5.5.4.4. Escenario de Seleccionar modelo*

*(Ver figura 5.23)*. El proceso de selección de un MEM es el siguiente:

Primero, el objeto Control enviará la orden a la interfaz (VentanaPrincipal) del MEM que se quiere visualizar. La interfaz creará un objeto de tipo GrafoMEM a partir del MEM seleccionado. Dicho grafo constará de un conjunto de vértices y arcos.

A continuación se creará un objeto de tipo LayoutMEM con la información del grafo que acabamos de crear. Dicho objeto almacenará información sobre la posición que ocupará cada vértice del grafo.

Con esta información, crearemos un objeto de tipo GraphZoomScrollPane. Básicamente, se trata del elemento de la interfaz que contendrá la representación del hipergrafo (además de permitir operaciones de traslación y ampliación/reducción de forma automática).

Por último, añadiremos este elemento a la interfaz principal.

#### *5.5.4.5. Escenario de Modificar condiciones*

*(Ver figura 5.24)*. Cuando el usuario modifica una condición, almacenaremos dicha condición y volveremos a dibujar el MEM actual teniendo en cuenta dicha información.

#### *5.5.4.6. Escenario de Generar PDF*

*(Ver figura 5.25)*. Para generar un archivo PDF con la representación gráfica del MEM actualmente seleccionado lo primero será obtener la ruta en la que se almacenará el fichero. Una vez obtenida redibujaremos el MEM actual cambiando ligeramente su aspecto (colores, fondo) para adecuarlo al archivo PDF.

Una vez almacenado, volveremos a dibujar el MEM con la configuración original de colores.

#### *5.5.4.7. Escenario de Consultar ayuda*

*(Ver figura 5.26)*.

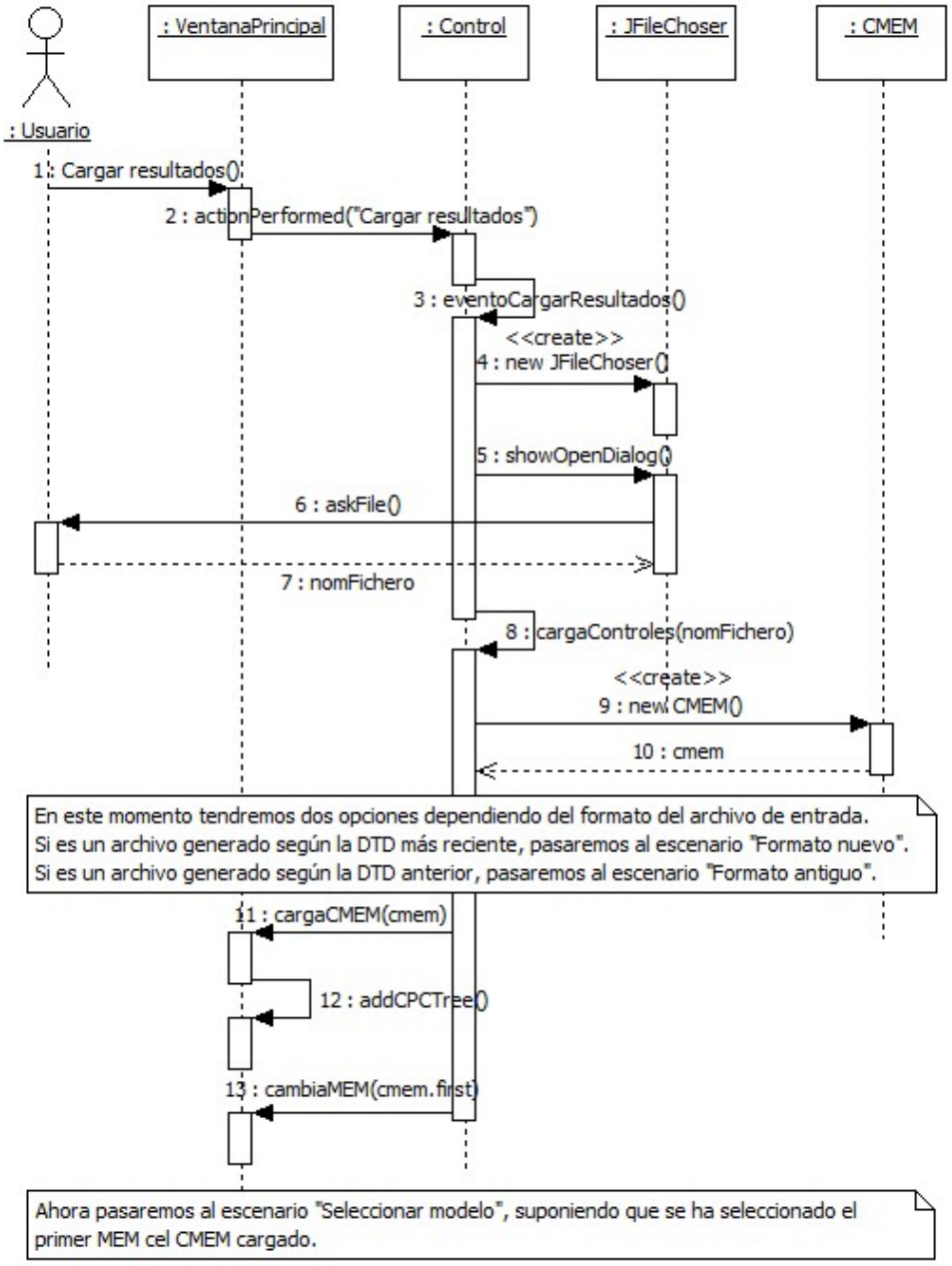



***Figura 5.18: Escenario de Cargar resultados***

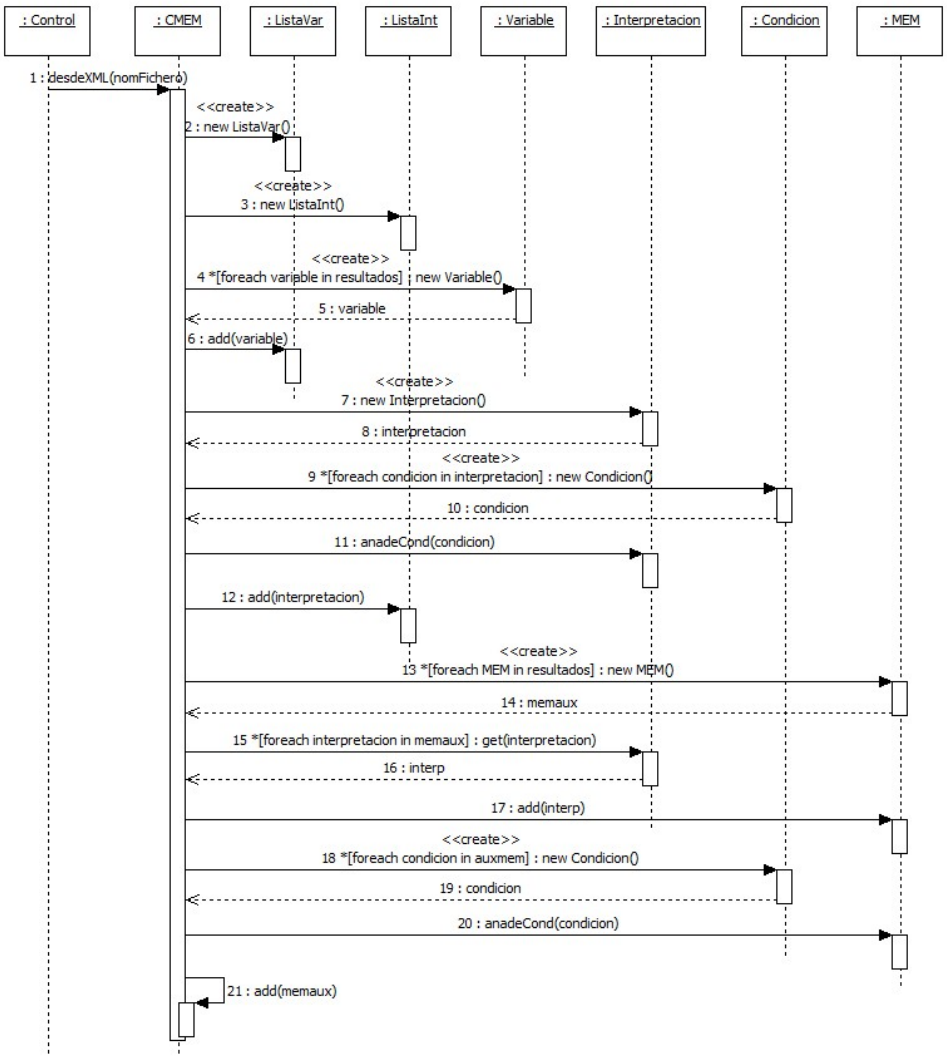



***Figura 5.19: Escenario de Cargar resultados (formato nuevo)***

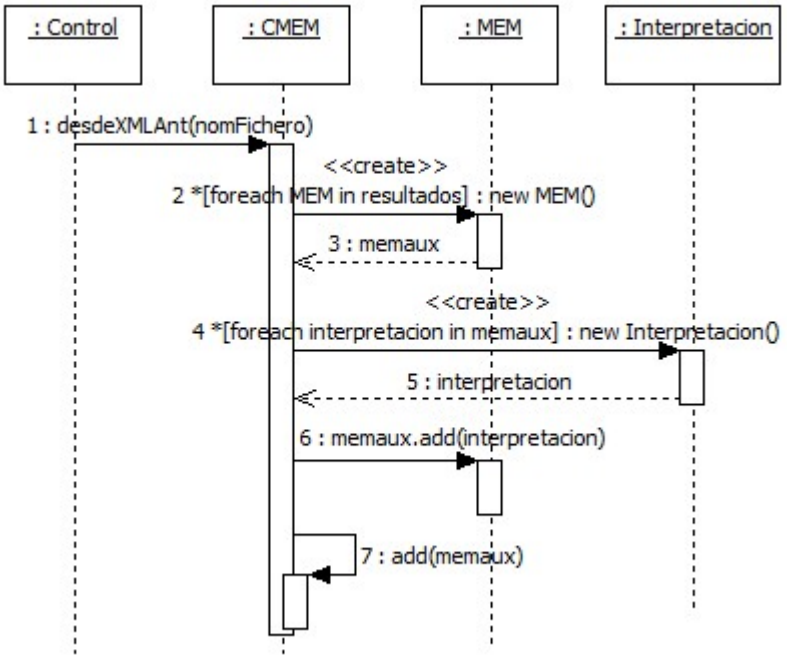



***Figura 5.20: Escenario de Cargar resultados (formato antiguo)***

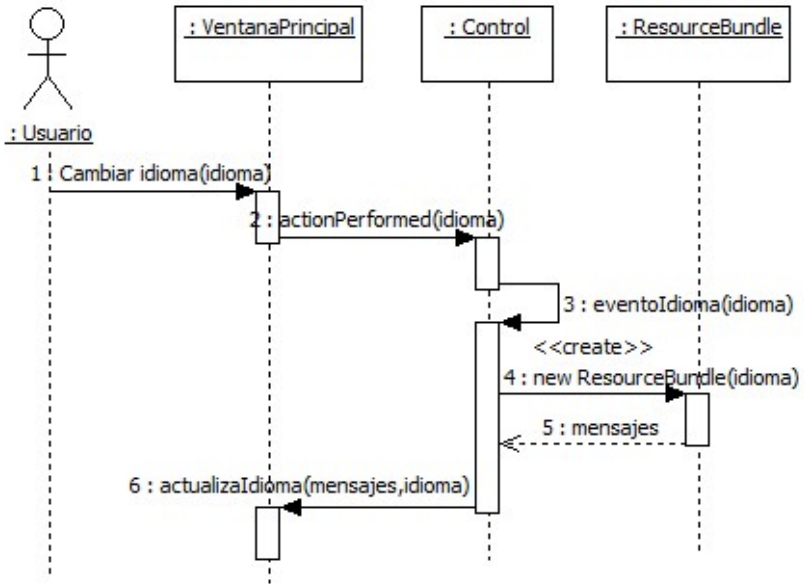



***Figura 5.21: Escenario de Cambiar idioma***

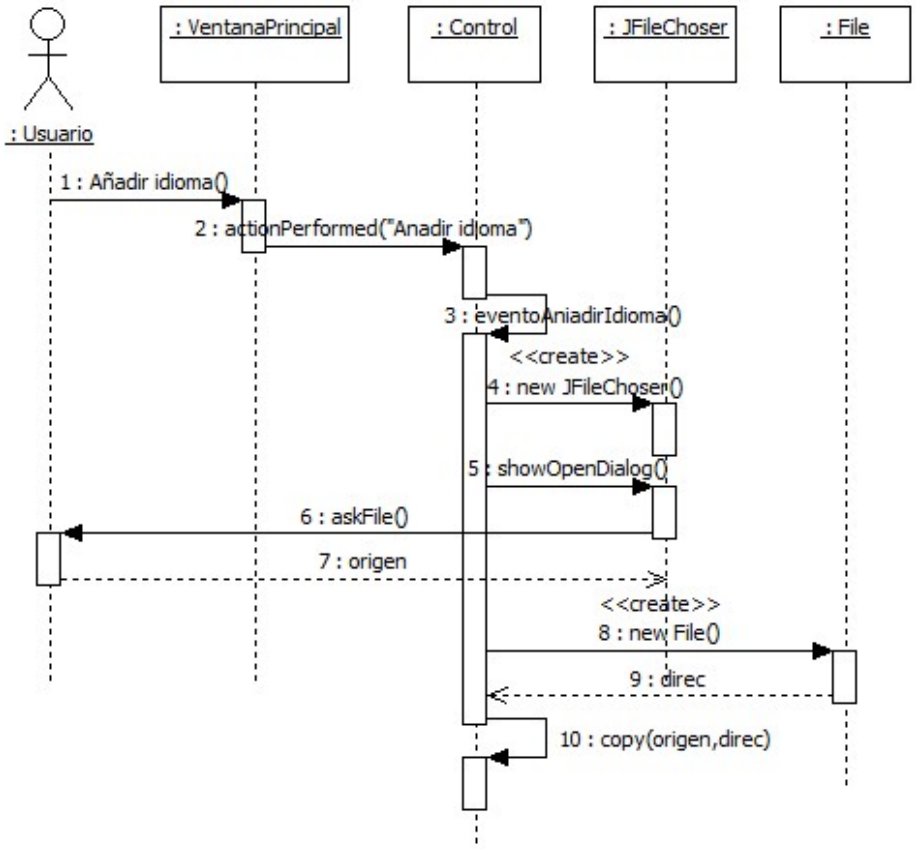



***Figura 5.22: Escenario de Añadir idioma***

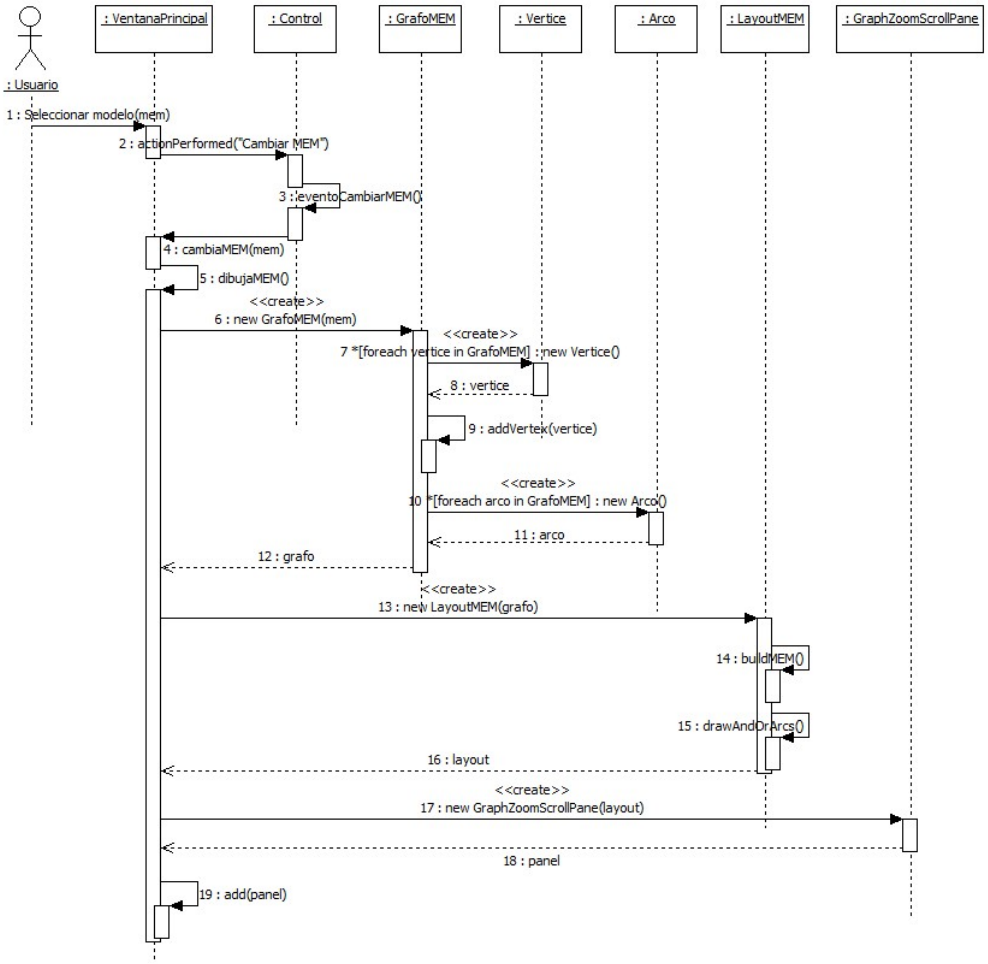



***Figura 5.23: Escenario de Seleccionar modelo***

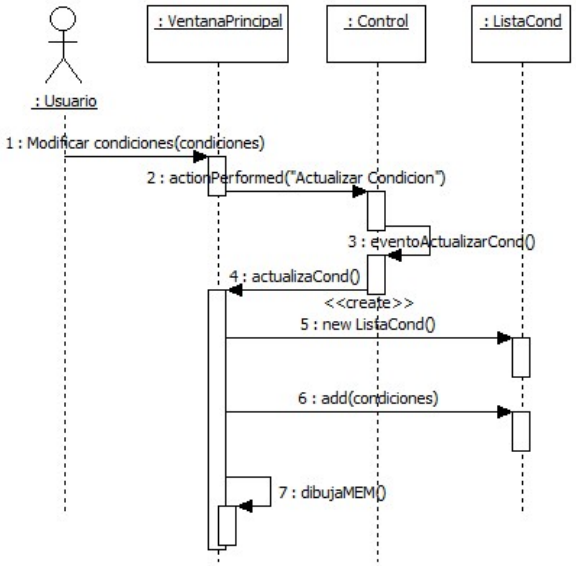



***Figura 5.24: Escenario de Modificar condiciones***

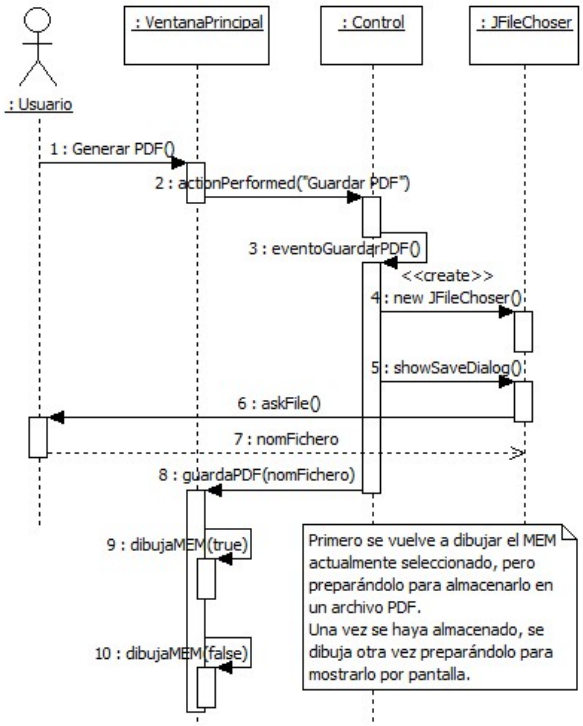



***Figura 5.25: Escenario de Generar PDF***

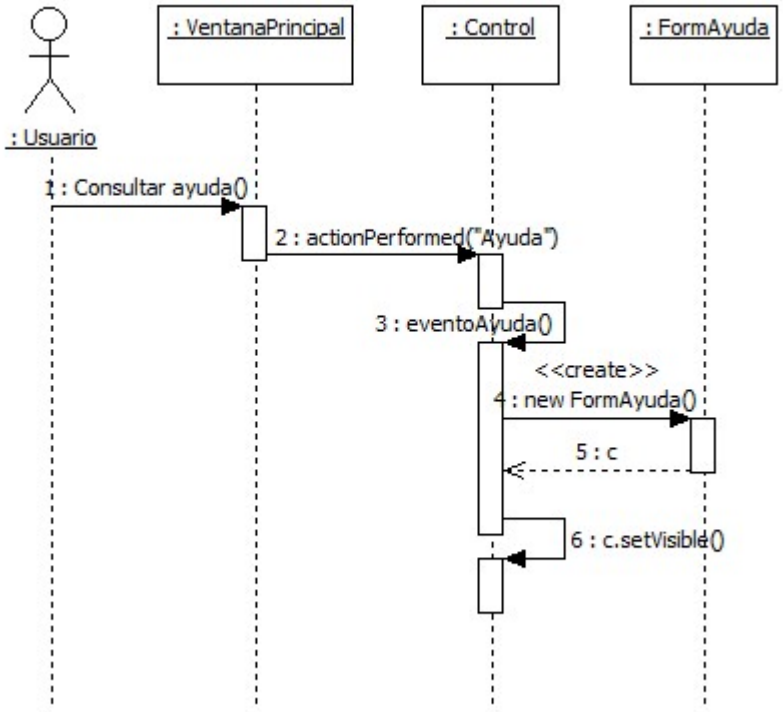



***Figura 5.26: Escenario de Consultar ayuda***

# 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 éstos mismos los que lo generen).

### *6.1.2. Alcance*

Este modelo de diseño proporcionará información sobre el proceso de implementación del proyecto "Revisión y adaptación de la herramienta para el cálculo de Posibles Conflictos".

### *6.1.3. Definiciones, Acrónimos, y Abreviaturas*

Ver el glosario general del proyecto (capítulo X).

### *6.1.4 Referencias*

En el presente capítulo encontramos referencias a los siguientes apartados del documento:

- Especificación de requisitos
- Glosario
- Referencias bibliográficas

### 6.1.5. 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.
- El visor: en este apartado daremos una descripción detallada del proceso de implementación de la nueva aplicación de visualización de resultados.

## 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 7).

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 Eclipse

El entorno de desarrollo que se va a utilizar será Eclipse. Se trata de un software libre (posee una licencia propia denominada EPL, Eclipse Public License) con el que el equipo de desarrollo ya había trabajado con anterioridad.

Se trabajará siempre con la versión más actualizada de la aplicación.

### 6.2.3. Sistema operativo Windows

El sistema operativo que se utilizará será Windows 7; aunque se realizarán pruebas en otros sistemas operativos.

# 6.3. Aportaciones a la herramienta de CPCs

## *6.3.1. Estructuras utilizadas*

Uno de los principales objetivos de este TFG ha sido la mejora de la eficiencia de los algoritmos de cálculo de CPCs que ya se hallaban implementados.

Tras analizar el algoritmo básico, se llegó a la conclusión de que un factor de gran peso en el tiempo de cálculo eran las sucesivas inserciones de elementos en las estructuras que almacenan el conjunto de CEM y MEM (y los propios CEM y MEM, ya que éstos son, a su vez, un conjunto de ecuaciones e interpretaciones respectivamente), en las que no puede haber elementos repetidos; por lo que se concluyó que el uso de estructuras ordenadas combinadas con algoritmos de búsqueda eficientes podrían disminuir significativamente el tiempo de cálculo.

Para realizar esta tarea el primer paso fue comprobar si el lenguaje de programación Java disponía de este tipo de estructuras con métodos de inserción, búsqueda y borrado optimizados. Nos encontramos con que el lenguaje poseía dos estructuras que se adaptaban a estas características. Una de ellas es la estructura TreeSet, basada en la creación de una estructura ordenada; la otra se denomina HashSet y usa los denominados *mapas hash* o *tablas de dispersión*. Ambas garantizan un coste de O(log(n)) en las tres operaciones básicas: add, remove y contains (añadir, eliminar y comprobar existencia de un elemento). Además, poseen otra característica, y es que de forma implícita no admiten elementos repetidos; por lo que no tendremos que preocuparnos de esta cuestión, haciéndose el código mucho más sencillo y legible.

Otra opción algo más artesanal que se nos presentaba era el uso de un vector estándar combinado con la función *binarySearch* de la clase *Collections* (clase genérica de la que descienden la gran mayoría de las estructuras de Java que almacenan varios elementos de cualquier tipo). Esta función realiza una búsqueda binaria (que también garantiza un coste de O(log(n))) de un elemento en el vector, devolviendo la posición en la que se encuentre. Si no lo encuentra devolverá un número entero negativo que nos permitirá calcular la posición en la que deberíamos insertar el elemento para que el vector siguiera estando ordenado mediante la siguiente fórmula:

$$Punto\ de\ inserción = -(ValorDevuelto) - 1$$

Las estructuras que usamos en las pruebas fueron las siguientes:

- Vector ordenado
- HashSet
- TreeSet

Para realizar estas pruebas se escribió un pequeño programa que realiza lo siguiente:

1) Crea un conjunto de partida P de entre 100 y 1000 elementos. Dichos elementos serán strings de 8 caracteres cada uno. Al número de elementos de este conjunto lo llamaremos N
2) Crea un conjunto K1 de entre N y $N^2$ elementos. A este número le llamaremos M. De este se han probado dos variaciones:
    - En la primera, cada elemento era la concatenación de 1 ó 2 elementos de P. Para evitar errores por falta de memoria se limitaba el tamaño de este conjunto a 850000 elementos
    - En la segunda, cada elemento era la concatenación de entre 10 y 20 elementos de P. Para evitar errores por falta de memoria se limitaba el tamaño de este conjunto a 250000 elementos
3) Crea un conjunto K2 de las mismas características que el anterior pero con la mitad de elementos.
4) Crea un Vector.
    a) Inserta en él de forma ordenada y sin repeticiones los elementos de K1. Muestra por pantalla el tiempo necesario para realizar dichas inserciones.
    b) Busca y elimina de este Vector los elementos de K2. Muestra por pantalla el tiempo necesario para realizar dichas búsquedas y borrados.
5) Crea un HashSet.
    a) Inserta en él sin repeticiones los elementos de K1. Muestra por pantalla el tiempo necesario para realizar dichas inserciones.
    b) Busca y elimina de este HashSet los elementos de K2. Muestra por pantalla el tiempo necesario para realizar dichas búsquedas y borrados.
6) Crea un TreeSet.
    a) Inserta en él sin repeticiones los elementos de K1. Muestra por pantalla el tiempo necesario para realizar dichas inserciones.
    b) Busca y elimina de este TreeSet los elementos de K2. Muestra por pantalla el tiempo necesario para realizar dichas búsquedas y borrados.

Tras la realización de las pruebas se ha concluido que el uso de la estructura Vector es, con una enorme diferencia, la menos eficiente de las tres. Respecto a las otras dos estructuras (HashSet y TreeSet), ambas parecen ser bastante eficientes, pero los mejores tiempos de cálculo se han obtenido con la estructura HashSet.

En la siguiente tabla podemos ver un resumen de los valores obtenidos durante las pruebas:

| Tiempo de cálculo (ms) | | Vector | | HashSet | | TreeSet | |
|---|---|---|---|---|---|---|---|
| #P | TAMAÑO | Inserción | Borrado | Inserción | Borrado | Inserción | Borrado |
| 1-2 | N=126 M=153950 | 390 | 171 | 63 | 16 | 172 | 63 |
| | N=226 M=98130 | 780 | 562 | 47 | 32 | 140 | 94 |
| | N=954 M=850000 | 88000 | 50545 | 1109 | 255 | 2793 | 1685 |
| | N=845 M=850000 | 70809 | 46115 | 437 | 219 | 2621 | 1561 |
| | N=493 M=850000 | 17690 | 14977 | 500 | 187 | 2169 | 1140 |
| | N=714 M=850000 | 49640 | 37254 | 469 | 218 | 2542 | 1499 |
| | N=430 M=850000 | 11513 | 9579 | 485 | 172 | 2074 | 1062 |
| | N=929 M=850000 | 83726 | 49125 | 438 | 218 | 2637 | 1529 |
| | N=47 M=158890 | 3978 | 1873 | 79 | 31 | 281 | 172 |
| | N=951 M=850000 | 87329 | 49953 | 438 | 218 | 2698 | 1608 |
| **PROMEDIO** | | **41385,50** | **26015,40** | **406,50** | **156,60** | **1812,70** | **1041,30** |
| 10-20 | N=150 M=4045 | 16 | 0 | 0 | 0 | 15 | 0 |
| | N=463 M=94070 | 2762 | 189 | 78 | 47 | 218 | 140 |
| | N=244 M=43989 | 609 | 47 | 32 | 15 | 78 | 47 |
| | N=459 M=16263 | 93 | 16 | 0 | 16 | 15 | 16 |
| | N=444 M=82312 | 2028 | 95 | 63 | 31 | 172 | 109 |
| | N=797 M=5344 | 15 | 0 | 0 | 0 | 16 | 0 |
| | N=158 M=8126 | 31 | 0 | 16 | 0 | 0 | 16 |
| | N=475 M=124260 | 4509 | 172 | 94 | 47 | 296 | 187 |
| | N=775 M=250000 | 18096 | 440 | 218 | 94 | 749 | 469 |
| | N=338 M=63625 | 1233 | 80 | 46 | 16 | 125 | 78 |
| **PROMEDIO** | | **2939,20** | **103,90** | **54,70** | **26,60** | **168,40** | **106,20** |

A la hora de implementar los algoritmos usando la estructura HashSet, nos encontramos con un inconveniente: aunque hemos comprobado que la búsqueda y la inserción eran considerablemente eficientes en esta estructura, el hecho de comparar dos objetos de este tipo, para comprobar si contienen los mismos elementos, se hacía considerablemente complicado e ineficiente; puesto que, por la naturaleza de las tablas de dispersión, no se garantiza el orden de los elementos. Esto nos hizo decantarnos finalmente por la estructura TreeSet.

El último paso sería entonces implementar el algoritmo con la estructura elegida y comprobar que realmente se produce una disminución notable en los tiempos de cálculo.

Para ello utilizaremos dos archivos de entrada:

- Archivo de entrada A1: "*modelo_robert_ia_ecosim_08_04_2011.txt*"
  - Número de ecuaciones: 52
  - Número de interpretaciones: 70
  - CEM obtenidas: 1058
  - MEM obtenidos: 776
  - PC obtenidos: 345
- Archivo de entarada A2: "*ent_larga.txt*"
  - Número de ecuaciones: 42
  - Número de interpretaciones: 53
  - CEM obtenidas: 458
  - MEM obtenidos: 14
  - PC obtenidos: 7

En la siguiente tabla mostraremos los tiempos de cálculo de la versión anterior del algoritmo básico de CPCs, comparándola con la del algoritmo implementado utilizando estas estructuras:

| Tiempo de cálculo (ms) | | Versión anterior (listas enlazadas) | | | Nueva versión (TreeSet) | | |
|---|---|---|---|---|---|---|---|
| | | Cálculo de CCEM | Cálculo de CMEM | TOTAL | Cálculo de CCEM | Cálculo de CMEM | TOTAL |
| **A1** | **Prueba 1** | 42338 | 3042 | 45380 | 22837 | 427 | 23263 |
| | **Prueba 2** | 41933 | 3073 | 45006 | 22737 | 550 | 23287 |
| | **Prueba 3** | 42089 | 2995 | 45084 | 23315 | 432 | 23747 |
| **Promedio** | | 42120 | 3037 | 45157 | 22963 | 470 | 23433 |
| **A2** | **Prueba 1** | 219 | 156 | 375 | 90 | 32 | 122 |
| | **Prueba 2** | 203 | 156 | 359 | 87 | 32 | 119 |
| | **Prueba 3** | 187 | 141 | 328 | 98 | 33 | 132 |
| **Promedio** | | 203 | 151 | 354 | 92 | 32 | 124 |

Como podemos ver, se ha producido una mejora sustancial en la eficiencia de los algoritmos en términos de tiempo de cálculo, especialmente en el cálculo de los MEM, en el que dicho tiempo se ha llegado a disminuir entre 5 y 6 veces.

Como término medio, basándonos en las pruebas que hemos realizado, para sistemas de entrada moderadamente grandes los tiempos de cálculo se han reducido aproximadamente a la mitad.

## 6.3.2. Algoritmos modificados

A continuación mostraremos el pseudocódigo de los dos algoritmos que se han modificado.

### 6.3.2.1. Algoritmo básico de CPCs

#### 6.3.2.1.1. Construcción de cadenas

```
ALGORITMO calcularCadenas(HSD, CCEM) ES
INICIO
   disponibles := HSD.extraerEcuaciones();
   MIENTRAS !disponibles.esVacío() HACER
      r:=disponibles.primera-ecuacion();
      cadena := nuevo CEM();
      disponibles.eliminar(r);
      construirCadena(CCEM, cadena, r, disponibles)
   FIN MIENTRAS
FIN
```

Con una llamada a esta función se comenzará el cálculo de las CEM. Básicamente, llamaremos a la función construirCadena de forma iterativa extrayendo cada vez de ésta una ecuación del sistema hasta que la lista de ecuaciones del sistema haya quedado vacía.

```
ALGORITMO construirCadena(CCEM, cadena, r, disponibles) ES
INICIO
   cadena.añadir(r);
   por-justificar := r.variablesNoObservables();
   justificadas := nueva listaVar();
   justificar(CCEM, cadena, por-justificar, justificadas, disponibles);
FIN
```

Esta función tratará de construir todas las CEM que contengan una ecuación concreta (r). Para ello deberemos tener en cuenta las variables no observables que contenga dicha ecuación (que almacenaremos en una lista de variables "por justificar"). Con esta información llamaremos a la función justificar, que es una función recursiva que se encargará de realizar la mayor parte de los cálculos.

```
ALGORITMO justificar(CCEM, cadena, por-justificar, justificadas,
disponibles) ES
INICIO
   SI por-justificar.esVacío() ENTONCES
      CCEM.añadir(cadena);
   SINO
      v := por-justificar.primeraVariable();
      PARA r1 := cada ecuación en disponibles HACER
         SI r1.variables.contiene(v) ENTONCES
            aux-cadena := cadena;
            SI aux-cadena.añadir(r1) tiene éxito ENTONCES
               aux-disponibles := disponibles;
               aux-por-juistificar := por-justificar;
               aux-justificadas := justificadas;
               aux-disponibles.eliminar(r1);
               aux-justificadas.añadir(v);
               PARA aux-variable := cada variable no observable en r1 HACER
                  SI (!aux-justificadas.contiene(aux-variable) Y
                      !aux-por-justificar.contiene(aux-variable)) ENTONCES
                     aux-por-justificar.añadir(aux-variable);
                  FIN SI
               FIN PARA
               aux-por-justificar.eliminar(v);
               justificar(CCEM, aux-cadena, aux-por-justificar,
                          aux-justificadas, aux-disponibles);
            FIN SI
         FIN SI
      FIN PARA
   FIN SI
FIN
```

En esta función recursiva tendremos un caso base en el que, si no quedan variables por justificar, la CEM estará completa y se añadirá al CCEM.

En caso contrario buscaremos una ecuación de entre las que todavía no hemos utilizado que contenga la primera de las variables que nos quedan por justificar. Dicha variable se convertirá en una variable justificada y el resto de variables no observables de la ecuación (que no hayan sido previamente "justificadas") pasarán a engrosar la lista de variables por justificar. Hecho esto volveremos a llamar a la función justificar de forma recursiva.

Es muy importante que las operaciones que hemos descrito en el párrafo anterior no modifiquen las estructuras con las que trabajamos; puesto que podríamos llegar a un punto en que la CEM no pueda ser construida con las variables que hemos añadido, por lo que habrá que realizar copias de dichas estructuras.

La diferencia entre la versión anterior del algoritmo y ésta es que tanto las operaciones de adición de la CEM al CCEM o de la ecuación a la CEM no necesitan de una previa comprobación de pertenencia, puesto que trabajamos con estructuras que hacen esto de forma automática.

*6.3.2.1.2. Construcción de modelos*

```
ALGORITMO calcularModelos(CMEM, CCEM) ES
INICIO
   PARA cadena := cada cadena en CCEM HACER
      construirModelos(CMEM, cadena);
   FIN PARA
FIN
```

El cálculo de MEMs partirá del conjunto de CEMs previamente calculado. Para cada CEM obtenida llamaremos a la función construirModelos.

```
ALGORITMO construirModelos(CMEM, cadena) ES
INICIO
   PARA r := cada ecuación en cadena HACER
      PARA i := cada interpretación en r HACER
         modelo := nuevo MEM(i);
         por-justificar := i.variablesNoObservables();
         justificadas := nueva listaVar()
         aux-cadena := cadena;
         aux-cadena.eliminar(r);
         construirModelo(CMEM, modelo, aux-cadena,
                         por-justificar, justificadas);
      FIN PARA
   FIN PARA
FIN
```

Esta función recorrerá cada interpretación de cada ecuación de la CEM de la que partimos. Para cada interpretación haremos lo siguiente:

- Crearemos un nuevo MEM al que añadiremos la interpretación.
- Añadiremos a una lista de variables por justificar todas las variables no observables de la interpretación (tanto de la cabeza como de la cola)
- Crearemos una copia de la CEM, de la que eliminaremos la ecuación a la que pertenece la interpretación actual
- Llamaremos a la función construirModelo.

```
ALGORITMO construirModelo(CMEM, modelo, sinUsar, por-justificar,
                          justificadas) ES
INICIO
   SI por-justificar={} ENTONCES
      CMEM.añadir(modelo);
   SINO
      PARA aux-ecuacion:=cada ecuación en sinUsar HACER
         calcular := falso;
         aux-interpretacion := aux-ecuacion.primeraInterpretacion();
```

```
         MIENTRAS aux-interpretacion != nulo Y !calcular HACER
            SI cabeza(aux-interpretacion) =
                      por-justificar.primeraVariable() ENTONCES
               calcular := verdadero;
            FIN SI
            SI !calcular ENTONCES
               aux-interpretacion := aux-ecuacion.siguienteInterp();
            FIN SI
         FIN MIENTRAS
         SI calcular ENTONCES
            MIENTRAS aux-interpretacion != nulo HACER
               i2 := aux-interpretacion;
               SI por-justificar.contiene(i2.cabeza()) ENTONCES
                  aux-modelo := modelo;
                  aux-sinUsar := sinUsar;
                  aux-por-juistificar := por-justificar;
                  aux-justificadas := justificadas;
                  aux-modelo.añadir(i2);
                  aux-sin-usar.eliminar(aux-ecuacion);
                  aux-por-justificar.eliminar(i2.cabeza());
                  PARA v2 := cada variable en i2.cola() HACER
                     SI !aux-justificadas.contiene(v2)
                        aux-por-justificar.añadir(v2);
                     FIN SI
                  FIN PARA
                  aux-justificadas.añadir(i2.cabeza());
                  construirModelo(CMEM, aux-modelo, aux-sinUsar,
                                  aux-por-justificar, aux-justificadas);
               FIN SI
               aux-interpretacion := aux-ecuacion.siguienteInterp();
            FIN MIENTRAS
         FIN SI
      FIN PARA
   FIN SI
FIN
```

Esta es la función recursiva que realizará la mayor parte de los cálculos (equivalente a la función justificar para el cálculo de las CEMs). Como en ésta, el caso base consistirá en añadir el MEM que estamos construyendo al CMEM en caso de que no queden variables por justificar.

En otro caso, empezaremos a recorrer las ecuaciones restantes de la CEM y, para cada una de ellas, sus interpretaciones, hasta hallar una cuya cabeza coincida con la primera variable por justificar.

Una vez encontrada, para la interpretación hallada y para el resto de las interpretaciones de la ecuación que cumplan esta condición realizaremos lo siguiente:

- Insertamos la interpretación en el modelo.
- Eliminamos la ecuación de la lista de ecuaciones que no hemos usado.
- Eliminamos de la lista de variables por justificar la variable cabeza de la interpretación y la insertamos en la lista de variables justificadas.
- Insertamos las variables no observables de la cola de la interpretación que no hayan sido justificadas anteriormente en la lista de variables por justificar.
- Llamamos recursivamente a la función cosntruirModelo

Por la misma razón que lo hicimos en la función justificar, realizaremos estas últimas operaciones sobre copias de las estructuras.

#### *6.3.2.2. Algoritmo de CPCs con condiciones*

En este nuevo algoritmo añadiremos condiciones a las ecuaciones y las interpretaciones del sistema.

Las condiciones podrán tener 3 valores: T, F y X (verdadero, falso e indiferente, respectivamente). Consideraremos que dos condiciones son incompatibles si poseen el mismo nombre y su valor es opuesto (una de ellas tiene valor T y otra F). El resto de posibles combinaciones se considera compatible.

La única modificación que requerirá esto en los algoritmos es que, a la hora de añadir una ecuación a una CEM o una interpretación a un MEM debemos comprobar si las condiciones de la ecuación/interpretación que nos disponemos a añadir son compatibles con todas las condiciones de cada una de las ecuaciones/interpretaciones que ya se han insertado en la CEM/el MEM.

Para realizar esto se ha sustituido la solución usada por Raúl Sánchez [San07], que consistía en añadir dos nuevas funciones llamadas compatible y merge, por una sola función que se implementará para las CEM y los MEM, y que hemos llamado esCompatible. Dicha función recibirá como argumento una lista de condiciones y comprobará, una por una, si todas ellas son compatibles con las condiciones de cada uno de los elementos incluidos en la estructura.

A continuación podremos ver el pseudocódigo de las funciones justificar y construirModelo (que son las que se verán afectadas por la presencia de condiciones). Hemos sombreado el código añadido con respecto al algoritmo básico.

##### *6.3.2.2.1. Construcción de cadenas*

```
ALGORITMO justificar(CCEM, cadena, por-justificar, justificadas,
disponibles) ES
INICIO
   SI por-justificar.esVacío() ENTONCES
      CCEM.añadir(cadena);
   SINO
      v := por-justificar.primeraVariable();
      PARA r1 := cada ecuación en disponibles HACER
         SI r1.variables.contiene(v) ENTONCES
```

```
            aux-cadena := cadena;
            SI aux-cadena.esCompatible(r1.condiciones) ENTONCES
               SI aux-cadena.añadir(r1) tiene éxito ENTONCES
                  aux-disponibles := disponibles;
                  aux-por-juistificar := por-justificar;
                  aux-justificadas := justificadas;
                  aux-disponibles.eliminar(r1);
                  aux-justificadas.añadir(v);
                  PARA aux-variable := cada variable no observable en r1
                  HACER
                     SI (!aux-justificadas.contiene(aux-variable) Y
                         !aux-por-justificar.contiene(aux-variable))
                     ENTONCES
                        aux-por-justificar.añadir(aux-variable);
                     FIN SI
                  FIN PARA
                  aux-por-justificar.eliminar(v);
                  justificar(CCEM, aux-cadena, aux-por-justificar,
                             aux-justificadas, aux-disponibles);
               FIN SI
            FIN SI
         FIN SI
      FIN PARA
   FIN SI
FIN
```

*6.3.2.2.2. Construcción de modelos*

```
ALGORITMO construirModelo(CMEM, modelo, sinUsar, por-justificar,
                          justificadas) ES
INICIO
   SI por-justificar={} ENTONCES
      CMEM.añadir(modelo);
   SINO
      PARA aux-ecuacion:=cada ecuación en sinUsar HACER
         calcular := falso;
         aux-interpretacion := aux-ecuacion.primeraInterpretacion();
         MIENTRAS aux-interpretacion != nulo Y !calcular HACER
            SI cabeza(aux-interpretacion) =
                      por-justificar.primeraVariable() ENTONCES
               calcular := verdadero;
            FIN SI
            SI !calcular ENTONCES
               aux-interpretacion := aux-ecuacion.siguienteInterp();
            FIN SI
         FIN MIENTRAS
         SI calcular ENTONCES
            MIENTRAS aux-interpretacion != nulo HACER
               i2 := aux-interpretacion;
```

```
               SI modelo.esCompatible(i2.condiciones) ENTONCES
                  SI por-justificar.contiene(i2.cabeza()) ENTONCES
                     aux-modelo := modelo;
                     aux-sinUsar := sinUsar;
                     aux-por-juistificar := por-justificar;
                     aux-justificadas := justificadas;
                     aux-modelo.añadir(i2);
                     aux-sin-usar.eliminar(aux-ecuacion);
                     aux-por-justificar.eliminar(i2.cabeza());
                     PARA v2 := cada variable en i2.cola() HACER
                        SI !aux-justificadas.contiene(v2)
                           aux-por-justificar.añadir(v2);
                        FIN SI
                     FIN PARA
                     aux-justificadas.añadir(i2.cabeza());
                     construirModelo(CMEM, aux-modelo, aux-sinUsar,
                                     aux-por-justificar, aux-justificadas);
                  FIN SI
               FIN SI
               aux-interpretacion := aux-ecuacion.siguienteInterp();
            FIN MIENTRAS
         FIN SI
      FIN PARA
   FIN SI
FIN
```

# 6.4. El visor

El desarrollo de la herramienta de visualización de Posibles Conflictos es otro de los grandes objetivos de este TFG. Puesto que las versiones anteriores se encontraban en un nivel de madurez bastante bajo, esta parte del TFG será la que mayor carga de trabajo suponga, especialmente en la fase de implementación.

El primer gran paso para el desarrollo del visor será elegir la herramienta o biblioteca de funciones que se usará para desarrollarlo.

## *6.4.1. Elección de la herramienta a utilizar*

Nuestra meta, en un principio, fue encontrar una herramienta o biblioteca que nos permitiese dibujar hipergrafos dirigidos de forma automática, obteniendo una representación similar a la que vemos en la figura 6.1. Se planteó incluso utilizar una aplicación que no estuviese escrita en lenguaje Java, aunque esto dificultase la integración con la aplicación existente.

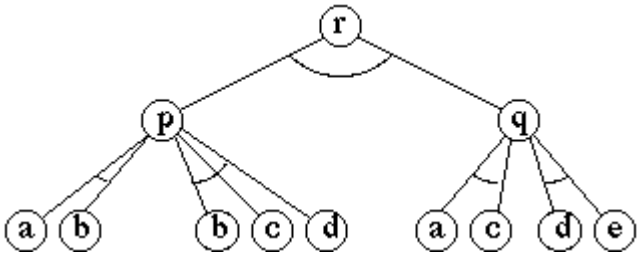



*Figura 6.1: Representación de un hipergrafo*

Tras una extensa búsqueda se abandonó este objetivo. Aunque se encontraron bibliotecas capaces de reconocer y realizar cálculos con grafos con arcos y/o, ninguna de ellas contaba con un método para dibujarlos, por lo que pasamos a analizar el resto de herramientas. Éstas debían proporcionarnos formas de distinguir de alguna manera el conjunto de arcos que forman un solo hiperarco (ya sea representándolos con diferentes colores o, mejor, uniéndolos con una pequeña línea curva, tal y como se muestra en la figura).

A continuación veremos un resumen de las opciones que se consideraron (la fecha de la última versión fue comprobada en octubre del año 2013):

| **JUNG** | |
|---|---|
| **Descripción** | Biblioteca de funciones que proporciona procedimientos para implementar y visualizar varios tipos de grafos. |
| **Tecnología** | Biblioteca de funciones Java |
| **Licencia** | BSD |
| **URL** | http://jung.sourceforge.net/ |
| **Última version** | 25/1/2010 (Estable. Versión 2) |
| **Evaluación** | Se trata de una aplicación estable, muy popular, bien documentada y con gran libertad a la hora de representar los grafos. |

| **TouchGraph** | |
|---|---|
| **Descripción** | Biblioteca centrada en la representación gráfica de grafos simples. |
| **Tecnología** | Biblioteca de funciones Java |
| **Licencia** | Apache License |
| **URL** | http://touchgraph.sourceforge.net/ |
| **Última version** | 5/12/2002 (Alpha) |
| **Evaluación** | Proporciona poca libertad a la hora de personalizar la representación.<br>Se encuentra en un bajo estado de madurez. Aunque en su página indican que se planea implementar futuras versiones lleva más de 10 años sin actividad. |

| AbuGraph | |
|---|---|
| **Descripción** | Aplicación para la representación de grafos dirigidos. Permite obtener los grafos de diversas formas (incluso de forma remota mediante un terminal telnet). |
| **Tecnología** | Aplicación Java |
| **Licencia** | BSD |
| **URL** | http://abugraph.sourceforge.net/ |
| **Última version** | 14/10/2007 (Alpha) |
| **Evaluación** | Esta aplicación se encuentra en un bajo estado de madurez. Ofrece algunas opciones de representación, pero se centra más en la forma de introducir los datos de los grafos. |

| JPowerGraph | |
|---|---|
| **Descripción** | Biblioteca de funciones que proporciona procedimientos para implementar y visualizar varios tipos de grafos. |
| **Tecnología** | Biblioteca de funciones Java |
| **Licencia** | GNU |
| **URL** | http://jpowergraph.sourceforge.net/ |
| **Última version** | 7/8/2007 (Beta) |
| **Evaluación** | Proporciona bastantes opciones de representación, pero se encuentra en fase beta y su documentación es escasa. |

| JDigraph | |
|---|---|
| **Descripción** | Biblioteca de funciones que proporciona procedimientos para implementar y visualizar grafos dirigidos. |
| **Tecnología** | Biblioteca de funciones Java |
| **Licencia** | BSD |
| **URL** | http://sourceforge.net/projects/jdigraph/ |
| **Última version** | 30/3/2003(Alpha) |
| **Evaluación** | La página web del proyecto no está disponible. Su estado de madurez es bajo y se han producido pocas versiones. No se ha encontrado documentación ni ejemplos sobre el uso de la biblioteca. |

| Grappa | |
|---|---|
| **Descripción** | Biblioteca de funciones que proporciona procedimientos para implementar y visualizar varios tipos de grafos. |
| **Tecnología** | Biblioteca de funciones Java |
| **Licencia** | Common Public License |
| **URL** | http://www2.research.att.com/~john/Grappa/ |
| **Última version** | 22/9/2010 (Estable) |
| **Evaluación** | Biblioteca estable, con buena documentación y con gran libertad a la hora de representar los grafos. |

| WilmaScope 3D Graph Visualizatio | |
|---|---|
| **Descripción** | Aplicación para la visualización de grafos en 3D |
| **Tecnología** | Aplicación Java |
| **Licencia** | GNU |
| **URL** | http://wilma.sourceforge.net/ |
| **Última version** | 26/11/2004 (Beta) |
| **Evaluación** | Se encuentra en bajo estado de madurez y no posee una comunidad activa de usuarios/desarrolladores.<br>Se centra en la visualización en 3D, cosa que, en principio, no nos interesa. |

| GINY | |
|---|---|
| **Descripción** | Biblioteca de funciones que proporciona procedimientos para implementar y visualizar varios tipos de grafos. |
| **Tecnología** | Biblioteca de funciones Java |
| **Licencia** | GNU |
| **URL** | http://csbi.sourceforge.net/ |
| **Última version** | 31/8/2005 (Beta) |
| **Evaluación** | Documentación aceptable, pero pocas opciones para personalizar la interfaz.<br>Se encuentra en versión beta, y no parece que se vaya a continuar su desarrollo. |

| HyperTree | |
|---|---|
| **Descripción** | Biblioteca para la implementación y visualización de árboles hiperbólicos |
| **Tecnología** | Biblioteca de funciones Java |
| **Licencia** | MIT |
| **URL** | http://hypertree.sourceforge.net/ |
| **Última version** | 5/9/2001 (Alpha) |
| **Evaluación** | La biblioteca no está disponible; según parece por algún problema de patentes con la empresa Xerox. |

| Hypergraph | |
|---|---|
| **Descripción** | Biblioteca para la implementación y visualización de árboles hiperbólicos |
| **Tecnología** | Biblioteca de funciones Java |
| **Licencia** | GNU |
| **URL** | http://hypergraph.sourceforge.net/ |
| **Última version** | 24/10/2005 (Beta) |
| **Evaluación** | Biblioteca centrada en la creación de grafos hiperbólicos para mapas de aplicaciones web. No parece inmediatamente aplicable a nuestro problema. |

Tras analizar estas herramientas comprobamos que había dos de ellas que se adecuaban perfectamente a nuestras necesidades y, además, poseían cierta estabilidad y estaban correctamente documentadas. Estas eran JUNG y Grappa.

Finalmente nos decidimos por JUNG con la esperanza de poder reutilizar parte del código de la aplicación realizada por Raúl Sánchez [San07].

## 6.4.2. La biblioteca gráfica JUNG

Existen grandes diferencias entre la versión de JUNG utilizada en la aplicación de Raúl Sánchez [San07] y la utilizada en este trabajo. A continuación explicaremos el funcionamiento de las clases más importantes incluidas en esta biblioteca que utilizaremos en la implementación de la aplicación.

### 6.4.2.1. El grafo

La biblioteca JUNG contiene múltiples clases para representar grafos de diversos tipos. A la hora de utilizarlos debemos indicar de qué tipo serán los vértices y los arcos de dichos grafos. Podemos, por ejemplo, crear un grafo en el que los vértices sean números enteros y los arcos cadenas de texto. También podemos usar como vértices y arcos clases definidas por nosotros. Denotaremos esto de la siguiente forma:

*Grafo<V,A>*

donde V es la clase los vértices y A la de los arcos.

Todas las clases para la definición de grafos descienden de forma directa o indirecta de una clase llamada ***HyperGraph***. Dicha clase incorpora los siguientes elementos y/o características:

- Hiperarcos (arcos que conectan un conjunto de vértices de cualquier tamaño)
- Arcos que unen dos vértices
- Lazos (arcos que unen un vértice consigo mismo)
- Arcos dirigidos y no dirigidos
- Arcos paralelos (varios arcos que unen un mismo conjunto de vértices)

En resumen, esta clase no impone ninguna restricción a la construcción del grafo.

Cada una de las clases que descienden de ésta será más restrictiva. En nuestro caso crearemos una extensión de la clase ***SparseGraph*** a la que denominaremos ***GrafoMEM***. La clase ***SparseGraph*** desciende de la clase ***Graph***, que a su vez desciende de ***Hypergraph***. Dicha clase sólo permite arcos que unen dos vértices, tanto dirigidos como no dirigidos.

Podría pensarse que nos convendría usar la clase ***Hypergraph***; puesto que permite la existencia de hiperarcos; pero dicha clase no proporciona ningún método de representación gráfica. En el apartado 6.4.3.1 explicaremos cómo solucionaremos esta limitación.

### *6.4.2.2. La representación gráfica*

Para la representación gráfica necesitaremos varias clases.

La más relevante será posiblemente la clase ***Layout***. Esta clase se encargará de recorrer un grafo asignando a cada vértice una posición en el plano, representada por dos coordenadas (x,y).

Se trata de una clase interfaz; es decir, sus métodos no están definidos. JUNG ofrece varias implementaciones para esta clase, dependiendo del tipo de grafo que queramos representar y de cómo queramos que aparezcan distribuidos los vértices; puesto que nosotros utilizaremos un tipo de grafo con unas características muy particulares debemos crear una implementación específica.

La implementación que crearemos (a la que llamaremos ***LayoutMEM***) está basada en el código de la clase ***TreeLayout***; puesto que los vértices y arcos de un MEM forman un árbol conexo con un único nodo raíz; pero a éste debemos añadirle lo que hemos denominado vértices "virtuales" para indicar cuándo varios arcos representan una misma interpretación (lo veremos con detalle en el apartado 6.4.3.2).

Una vez hayamos obtenido la posición de cada uno de los vértices del grafo gracias al objeto de tipo Layout, crearemos, a partir de éste, un objeto de tipo ***VisualizationModel***, que será el encargado de realizar la implementación.

Con este último crearemos un objeto de tipo ***VisualizationViewer***, que nos permitirá personalizar la interfaz (dar curvatura a los arcos "virtuales", representar los vértices con el nombre de la variable a la que representan y poner ésta en negrita en caso de que la variable sea observable, etiquetar las interpretaciones, etc.) y añadir controles de ratón para trasladar, ampliar y reducir la interpretación.

Por último, a partir del objeto ***VisualizationViewer***, crearemos un objeto ***GraphZoomScrollPane***, que incluirá la visualización en un panel con barras de desplazamiento. Este elemento podrá ser añadido de forma normal a nuestra interfaz basada en Java Swing.

## *6.4.3. Algoritmos importantes*

Tras haber detallado la forma en la que se trabaja con la biblioteca JUNG2, pasaremos a mostrar el pseudocódigo de las funciones que consideramos más importantes.

### *6.4.3.1. Construcción del grafo*

A continuación veremos el pseudocódigo de la función que se encarga de convertir uno de los MEMs del CMEM en el que hemos almacenado los resultados en un grafo.

```
ALGORITMO crearGrafo(mem) ES
INICIO
   memaux := mem;
   nivel := 0;
   var := memaux.primeraInterp().cabeza();
   vértice := nuevo vértice(var, nivel);
   nivelAnterior := nuevo Vector<Vértice>(vértice);
   grafo.añadir(vértice);
   grafo.root := vértice;
   MIENTRAS !memaux.esVacío() HACER
      nivel := nivel + 1;
      verticesAux := nuevo Vector<Vértice>();
      par := falso;
      PARA v1 := cada vértice en nivelAnterior HACER
         listaInterp := memaxu.interpConCabeza(v1);
         PARA interp := cada interpretación en listaInterp HACER
            memaux.eliminar(interp);
            SI interp.cola().tamaño() = 0
               v2 := nuevo vértice("vacío", nivel);
               grafo.añadir(v2);
               grafo.añadir(nuevo arco(interp, v2, v1, 0));
               verticesAux.añadir(v2);
            SINO
            i := 0;
               PARA var := cada variable en interp.cola() HACER
```

```
                v2 := nuevo vértice(var, nivel);
                grafo.añadir(v2);
                grafo.añadir(nuevo arco(interp, v2, v1, i));
                verticesAux.añadir(v2);
                i := i + 1;
             FIN PARA
          FIN SI
          /*VÉRTICES Y ARCOS "VIRTUALES"*/
          v3 := nuevo vérticeVirtual(interp, inicio);
          v4 := nuevo vérticeVirtual(interp, fin);
          grafo.añadir(v3);
          grafo.añadir(v4);
          grafo.añadir(nuevo arcoVirtual(interp, v3,v4,par));
          SI (par)
             par := falso;
          SINO
             par := verdadero;
          FIN SI
       FIN PARA
    FIN PARA
    nivelAnterior := verticesAux;
 FIN MIENTRAS
FIN
```

Este algoritmo creará el grafo por niveles, comenzando por la variable cabeza de la primera interpretación del MEM (nodo de discrepancia). Cada variable será representada por un nodo (o más de uno si la variable aparece en la cola de varias interpretaciones). Los arcos unirán cada una uno de los vértices que representan una variable de la cola de una interpretación con el vértice que representa la cabeza de la interpretación. Además, contendrán información sobre la interpretación a la que representan y la posición que ocupa en la cola de la interpretación la variable origen. En caso de que la cola de la interpretación esté vacía se añadirá un vértice que representará una variable inexistente a la que llamaremos "nula".

Una vez se han representado todas las interpretaciones del MEM pasaremos a crear lo que hemos llamado vértices y arcos "virtuales". Éstos nos permitirán dibujar pequeños arcos que unirán todos los arcos "reales" que representan una interpretación.

Crearemos, por tanto, dos vértices "virtuales" y un arco "virtual" que les una para cada una de las interpretaciones del MEM.

Además de esto, en la representación del MEM, los arcos "virtuales" serán los que se representarán junto a una etiqueta que identificará el nombre de la interpretación. Para evitar que estas etiquetas se solapen distinguiremos los arcos que ocupen una posición par o impar dentro de cada nivel, lo que nos permitirá posicionar una de cada dos etiquetas en una posición inferior; por esta razón necesitaremos para su creación el nombre de la interpretación a la que representa, los dos vértices que une y la posición (par o impar) que ocupa en su nivel del grafo.

### *6.4.3.2. Distribución espacial de los elementos del grafo*

Mediante estas funciones se determinará dónde irá colocado cada uno de los vértices que componen el grafo en el momento de realizar la representación gráfica.

```
ALGORITMO crearLayout(grafo, distanciaX, distanciaY) ES
INICIO
   buildMEM();
   drawAndOrArcs();
FIN
```

Este algoritmo consta de dos fases. La primera consistirá en la colocación de los vértices "reales", mediante la función ***buildMEM***. Una vez finalizado esto, se pasará a colocar los vértices "virtuales" (con la función ***drawAndOrArcs***), puesto que su posición dependerá de la de los anteriores.

```
ALGORITMO buildMEM() ES
INICIO
   puntoActual := (0, 0);
   root := grafo.root;
   calculateDimensionX(root);
   alreadyDone := nuevo Vector<Vértice>();
   puntoActual.x:=puntoActual.x+(posicionBase.obtener(root) / 2 +
                  distanciaX);
   buildMEM(root, puntoActual.x);
FIN
```

Comenzaremos en el punto (0,0). A esta posición la denominaremos posición actual y será compartida por todas las funciones del algoritmo.

El primer nodo que vamos a colocar será el nodo raíz del grafo. Para ello primero invocaremos la función ***calculateDimensionX***. Esta función almacenará la anchura del subárbol que tiene como raíz a cada uno de los vértices del grafo de forma recursiva en una estructura a la que llamaremos ***posicionBase***. El nodo raíz estará colocado en el punto medio del árbol, por lo que desplazaremos el punto actual a dicha posición.

A continuación llamaremos a la función ***buildMEM***, usando como argumentos el vértice raíz y la posición que ocupará en el eje horizontal (aunque la función se llame igual que ésta se trata de una función diferente, puesto que recibe un número distinto de argumentos).

```
ALGORITMO buildMEM(v, x) ES
INICIO
   SI !alreadyDone.contiene(v)
      alreadyDone.añadir(v);
      puntoActual.y := puntoActual.y + distanciaY;
      puntoActual.x := x;
      v.fijaPosicion(puntoActual);
      ultimoX := x- (posicionBase.obtener(v)) / 2;
```

```
      PARA interp := cada interpretación en grafo.getIntConCabeza(v) HACER
         finInterp := falso;
         i := 0 ;
         REPETIR
            arco := grafo.obtenerArco(interp, i);
            SI arco = nulo ENTONCES
               finInterp := verdadero;
            SI NO
               origen := arco.origen();
               tamañoXdeHijo := posicionBase.obtener(origen);
               inicioXdeHijo := ultimoX + tamañoXdeHijo / 2;
               buildMEM(origen, inicioXdeHijo);
               ultimoX:=ultimoX + tamañoXdeHijo + distanciaX;
            FIN SI
            i := i + 1;
         HASTA QUE fininterp
      FIN PARA
      puntoActual.y := puntoActual.y - distanciaY;
   FIN SI
FIN
```

Esta es la función recursiva que se encargará de fijar la posición de todos los vértices del grafo.

Primero comprobará que el vértice no ha sido ya colocado. A continuación descenderá el punto actual un nivel en sentido vertical y lo fijará horizontalmente en la posición indicada como argumento. Dicho punto será la posición del vértice.

A continuación se pasará a colocar los vértices que representan las variables que están en la cola de las interpretaciones que tienen como cabeza el vértice actual.

Para ello, primero calcularemos el punto situado más a la izquierda del subárbol que tiene como raíz el vértice actual (es el valor almacenado en la variable ***ultimoX***).

La posición en el eje de horizontal del siguiente vértice a colocar será el valor de ***ultimoX*** menos la mitad del valor del vector ***posicionBase*** almacenado para dicho vértice.

Una vez hayamos calculado su posición llamaremos de forma recursiva a la función ***buildMEM***, usando como argumentos el nuevo vértice y la nueva posición.

A continuación incrementaremos el valor de ***ultimoX*** con el valor de ***posicionBase*** para el vértice que acabamos de colocar, añadiéndole la distancia horizontal mínima entre vértices que hemos fijado y repetiremos las mismas operaciones para el siguiente vértice del nivel en que nos encontramos.

```
ALGORITMO calculateDimensionX(v) ES
INICIO
   dimX := 0;
   PARA interp:=cada interpretación en grafo.getIntConCabeza(v) HACER
      finInterp := falso;
```

```
      i := 0;
      REPETIR
         arco := grafo.obtenerArco(interp, i);
         SI arco = nulo ENTONCES
            finInterp := verdadero;
         SI NO
            dimX := dimX + calculateDimensionX(arco.origen()) + distX;
         FIN SI
         i := i + 1;
      HASTA QUE fininterp
   FIN PARA
   dimX := máximo(0, dimX - distX);
   posicionBase.fijar(v, dimX);
FIN
```

Esta es la función encargada de calcular la anchura del subárbol que tiene como raíz cada uno de los vértices del grafo; es decir, los valores del vector ***posicionBase***.

```
ALGORITMO drawAndOrArcs() ES
INICIO
   PARA v:=cada vértice en grafo.getArcosVirtuales().getOrigenes() HACER
      interpretación := v.getInterpretacion();
      padre := grafo.getArco(interpretación, 0).destino();
      hijo0 := grafo.getArco(interpretación, 0).origen();
      n := interpretación.cola().tamaño()-1;
      hijoUlt := grafo.getArco(interpretación, n).origen();
      x:=(hijo0.obtienePosicion.x + padre.obtienePosicion.x) * 2 / 3;
      y:=(hijo0.obtienePosicion.y + padre.obtienePosicion.y) * 2 / 3;
      v.fijaPosicion(x, y);
      v2:=vértice destino del arco "virtual" asociado a interpretación;
      x:=(hijoUlt.obtienePosicion.x + padre.obtienePosicion.x) * 2 / 3;
      y:=(hijoUlt.obtienePosicion.y + padre.obtienePosicion.y) * 2 / 3;
      v2.fijaPosicion(x, y);
   FIN PARA
FIN
```

Finalmente llegamos a la función que colocará los vértices "virtuales".

Lo primero que haremos será recorrer todos los vértices origen de los arcos "virtuales". Cada uno de ellos estará ligado a una interpretación. Su posición se encontrará entre el vértice que representa la cabeza de la interpretación y el que representa la primera variable de la cola (algo más próximo a la cabeza); mientras que el vértice virtual "destino" se encontrará entre el vértice que representa la cabeza de la interpretación y el que representa la última variable de la cola (en caso de que la cola sólo posea una variable, o ninguna, ambos vértices se situarán en el mismo punto; pero esto no supone ningún problema).

### *6.4.4. Salida en formato PDF*

Para finalizar el apartado de implementación daremos una breve descripción sobre la forma en que se genera la salida en formato PDF. Para esta tarea hemos utilizado una biblioteca llamada ***iText*** [Ite14], bajo licencia AGPL.

Los pasos a seguir serán los siguientes:

1. Redibujaremos el MEM actualmente seleccionado, cambiando determinados detalles (principalmente, colores de fondo y relleno de los elementos)
2. Almacenaremos en una imagen temporal con extensión JPG la representación contenida en el objeto de tipo ***VisualizationViewer***.
3. Creamos un archivo PDF en la ruta indicada por el usuario.
4. Añadimos al archivo PDF el nombre del MEM y la imagen debidamente escalada (y rotada en caso de que su anchura supere su altura) para ajustarse a un tamaño Din-A4.
5. Cerramos el archivo PDF y eliminamos la imagen temporal.
6. Volvemos a dibujar el MEM con la configuración original de colores.

# 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 apartado proporcionará información sobre las pruebas realizadas en la aplicación desarrollada durante el proyecto "Revisión y adaptación de la herramienta para el cálculo de Posibles Conflictos".

### *7.1.3. Definiciones, Acrónimos, y Abreviaturas*

Ver el glosario general del proyecto.

### *7.1.4 Referencias*

En el presente capítulo encontramos referencias a los siguientes apartados del documento:

- Glosario

### 7.1.5. Perspectiva general

En este documento encontraremos información sobre las pruebas realizadas. Estas pruebas las dividiremos en los siguientes apartados:

- Pruebas sobre la aplicación principal
- Pruebas sobre el plugin de CPCs
- Pruebas sobre el plugin de CPCs con condiciones
- Pruebas sobre la aplicación de visualización de resultados

## 7.2. Pruebas realizadas

### 7.2.1. Pruebas sobre la aplicación principal

| CP-01 | Carga de sistema desde txt sin condiciones |
|---|---|
| **Objetivo** | Cargar la información de un sistema sin condiciones desde un archivo de texto. |
| **Procedimiento** | Seleccionar la opción de "Cargar sistema" y elegir como archivo de entrada un archivo en formato txt que contenga una descripción correcta de un sistema sin condiciones. |
| **Salida esperada** | Se deberá mostrar la información del sistema cargado. Dicha información deberá coincidir con la contenida en el archivo de entrada. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-02 | Carga de sistema desde txt con condiciones |
|---|---|
| **Objetivo** | Cargar la información de un sistema con condiciones desde un archivo de texto. |
| **Procedimiento** | Seleccionar la opción de "Cargar sistema" y elegir como archivo de entrada un archivo en formato txt que contenga una descripción correcta de un sistema con condiciones. |
| **Salida esperada** | Se deberá mostrar la información del sistema cargado. Dicha información deberá coincidir con la contenida en el archivo de entrada. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-03 | Carga de sistema desde XML sin condiciones |
|---|---|
| **Objetivo** | Cargar la información de un sistema sin condiciones desde un archivo en formato XML. |
| **Procedimiento** | Seleccionar la opción de "Cargar sistema" y elegir como archivo de entrada un archivo en formato XML que contenga una descripción correcta de un sistema sin condiciones. |
| **Salida esperada** | Se deberá mostrar la información del sistema cargado. Dicha información deberá coincidir con la contenida en el archivo de entrada. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-04 | Carga de sistema desde XML con condiciones |
|---|---|
| **Objetivo** | Cargar la información de un sistema con condiciones desde un archivo en formato XML. |
| **Procedimiento** | Seleccionar la opción de "Cargar sistema" y elegir como archivo de entrada un archivo en formato XML que contenga una descripción correcta de un sistema con condiciones. |
| **Salida esperada** | Se deberá mostrar la información del sistema cargado. Dicha información deberá coincidir con la contenida en el archivo de entrada. |
| **Resultado** | El resultado obtenido es correcto. |

### *7.2.2. Pruebas sobre el plugin de CPCs*

| CP-05 | Añadir algoritmo |
|---|---|
| **Objetivo** | Añadir el nuevo algoritmo creado a la lista de algoritmos disponibles para la aplicación. |
| **Procedimiento** | Seleccionar la opción de "Añadir algoritmo" y elegir el archivo jar donde se encuentra el plugin. |
| **Salida esperada** | El sistema deberá confirmar que se ha añadido el algoritmo de forma correcta y éste deberá aparecer en la lista de algoritmos disponibles. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-06 | Cargar algoritmo |
|---|---|
| **Objetivo** | Cargar el nuevo algoritmo de CPCs. |
| **Procedimiento** | Seleccionar la opción de "Cargar algoritmo" y, a continuación, seleccionar el algoritmo llamado "CPC TreeSet". |
| **Salida esperada** | Deberá presentarse en la ventana una nueva pestaña con la opción de realizar el cálculo de CCEM y deberá confirmarse en la parte inferior que el algoritmo "CPC TreeSet" ha sido cargado. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-07 | Realizar cálculos |
|---|---|
| **Objetivo** | Realizar el cálculo del conjunto de PCs siguiendo todos los pasos necesarios (cálculo de CCEM, cálculo de CMEM y cálculo de CPC). |
| **Procedimiento** | Una vez cargado el sistema y el algoritmo de cálculo de PCs, se selecciona la opción "Calcular CPC". |
| **Salida esperada** | Aparecerán en la ventana tres pestañas con las cadenas, los modelos y los PCs calculados. |
| **Resultado** | El resultado obtenido es correcto. |
| **CP-08** | **Calcular CCEM** |
| **Objetivo** | Realizar el cálculo del CCEM. |
| **Procedimiento** | Tras haber cargado el sistema y el algoritmo de cálculo de PCs, se selecciona la opción "Calcular CCEM". |
| **Salida esperada** | Se mostrarán en la ventana las CEM obtenidas. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-09 | Calcular CMEM |
|---|---|
| **Objetivo** | Realizar el cálculo del CMEM. |
| **Procedimiento** | Tras haber calculado con éxito el CCEM, se selecciona la opción "Calcular CMEM". |
| **Salida esperada** | Se mostrarán en la ventana las MEM obtenidas. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-10 | Calcular CPC |
|---|---|
| **Objetivo** | Realizar el cálculo del CPC. |
| **Procedimiento** | Tras haber calculado con éxito el CMEM, se selecciona la opción "Calcular CPC". |
| **Salida esperada** | Se mostrarán en la ventana los PCs obtenidos. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-11 | Guardar resultados en formato txt |
|---|---|
| **Objetivo** | Almacenar los resultados obtenidos en formato txt. |
| **Procedimiento** | Tras haber calculado con éxito el CPC, se selecciona la opción "Guardar", se elige el formato "Texto plano" y se especifica la ruta y el nombre de archivo en el que se almacenarán los resultados. |
| **Salida esperada** | Se generará un archivo en formato txt con los resultados obtenidos. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-12 | Guardar resultados en formato XML |
|---|---|
| **Objetivo** | Almacenar los resultados obtenidos en formato XML. |
| **Procedimiento** | Tras haber calculado con éxito el CPC, se selecciona la opción "Guardar", se elige el formato "XML" y se especifica la ruta y el nombre de archivo en el que se almacenarán los resultados. |
| **Salida esperada** | Se generará un archivo en formato XML con los resultados obtenidos. |
| **Resultado** | El resultado obtenido es correcto. |

### *7.2.3. Pruebas sobre el plugin de CPCs con condiciones*

| CP-13 | Añadir algoritmo |
|---|---|
| **Objetivo** | Añadir el nuevo algoritmo creado a la lista de algoritmos disponibles para la aplicación. |
| **Procedimiento** | Seleccionar la opción de "Añadir algoritmo" y elegir el archivo jar donde se encuentra el plugin. |
| **Salida esperada** | El sistema deberá confirmar que se ha añadido el algoritmo de forma correcta y éste deberá aparecer en la lista de algoritmos disponibles. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-14 | Cargar algoritmo |
|---|---|
| **Objetivo** | Cargar el nuevo algoritmo de CPCs. |
| **Procedimiento** | Seleccionar la opción de "Cargar algoritmo" y, a continuación, seleccionar el algoritmo llamado "CPC TreeSet con condiciones". |
| **Salida esperada** | Deberá presentarse en la ventana una nueva pestaña con la opción de realizar el cálculo de CCEM y deberá confirmarse en la parte inferior que el algoritmo "CPC TreeSet con condiciones" ha sido cargado. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-15 | Realizar cálculos |
|---|---|
| **Objetivo** | Realizar el cálculo del conjunto de PCs siguiendo todos los pasos necesarios (cálculo de CCEM, cálculo de CMEM y cálculo de CPC). |
| **Procedimiento** | Una vez cargado el sistema y el algoritmo de cálculo de PCs, se selecciona la opción "Calcular CPC". |
| **Salida esperada** | Aparecerán en la ventana tres pestañas con las cadenas, los modelos y los PCs calculados. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-16 | Calcular CCEM |
|---|---|
| **Objetivo** | Realizar el cálculo del CCEM. |
| **Procedimiento** | Tras haber cargado el sistema y el algoritmo de cálculo de PCs, se selecciona la opción "Calcular CCEM". |
| **Salida esperada** | Se mostrarán en la ventana las CEM obtenidas. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-17 | Calcular CMEM |
|---|---|
| **Objetivo** | Realizar el cálculo del CMEM. |
| **Procedimiento** | Tras haber calculado con éxito el CCEM, se selecciona la opción "Calcular CMEM". |
| **Salida esperada** | Se mostrarán en la ventana las MEM obtenidas. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-18 | Calcular CPC |
|---|---|
| **Objetivo** | Realizar el cálculo del CPC. |
| **Procedimiento** | Tras haber calculado con éxito el CMEM, se selecciona la opción "Calcular CPC". |
| **Salida esperada** | Se mostrarán en la ventana los PC obtenidos. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-19 | Guardar resultados en formato txt |
|---|---|
| **Objetivo** | Almacenar los resultados obtenidos en formato txt. |
| **Procedimiento** | Tras haber calculado con éxito el CPC, se selecciona la opción "Guardar", se elige el formato "Texto plano" y se especifica la ruta y el nombre de archivo en el que se almacenarán los resultados. |
| **Salida esperada** | Se generará un archivo en formato txt con los resultados obtenidos. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-20 | Guardar resultados en formato XML |
|---|---|
| **Objetivo** | Almacenar los resultados obtenidos en formato XML. |
| **Procedimiento** | Tras haber calculado con éxito el CPC, se selecciona la opción "Guardar", se elige el formato "XML" y se especifica la ruta y el nombre de archivo en el que se almacenarán los resultados. |
| **Salida esperada** | Se generará un archivo en formato XML con los resultados obtenidos. |
| **Resultado** | El resultado obtenido es correcto. |

### *7.2.4. Pruebas sobre la aplicación de visualización de resultados*

| CP-21 | Cargar resultados desde archivo XML (formato antiguo) |
|---|---|
| **Objetivo** | Cargar un archivo de resultados obtenido con los algoritmos antiguos. |
| **Procedimiento** | Seleccionar la opción de "Cargar PCs" y elegir como archivo de entrada un archivo XML generado a partir de uno de los algoritmos antiguos de cálculo de CPCs. |
| **Salida esperada** | Se mostrará información sobre todos los MEM que contiene el CPC (en forma de lista desplegable) y una representación gráfica en forma de hiperárbol dirigido del primer MEM. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-22 | Cargar resultados desde archivo XML (formato nuevo) |
|---|---|
| **Objetivo** | Cargar un archivo de resultados obtenido con los algoritmos nuevos. |
| **Procedimiento** | Seleccionar la opción de "Cargar PCs" y elegir como archivo de entrada un archivo XML generado a partir de uno de los algoritmos nuevos de cálculo de CPCs. |
| **Salida esperada** | Se mostrará información sobre todos los MEM que contiene el CPC (en forma de lista desplegable) y una representación gráfica en forma de hiperárbol dirigido del primer MEM. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-23 | Añadir idioma |
|---|---|
| **Objetivo** | Añadir un nuevo idioma para mostrar la interfaz de la aplicación. |
| **Procedimiento** | Seleccionar la opción de "Añadir idioma" y elegir como archivo de entrada un archivo que contenga la configuración del idioma a añadir. |
| **Salida esperada** | Se mostrará un mensaje confirmando que el idioma ha sido añadido y, la próxima vez que se inicie la aplicación, el idioma estará disponible. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-24 | Modificar idioma |
|---|---|
| **Objetivo** | Modificar el idioma en el que se encuentra la interfaz. |
| **Procedimiento** | Dentro del menú "Idioma", seleccionar el idioma al que queremos cambiar la interfaz. |
| **Salida esperada** | Todos los elementos de la interfaz que dependan del idioma deberán ser actualizados, mostrándose en el idioma seleccionado. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-25 | Seleccionar MEM |
|---|---|
| **Objetivo** | Visualizar la representación gráfica de un MEM concreto. |
| **Procedimiento** | Seleccionar el MEM que se desea visualizar. |
| **Salida esperada** | Se deberá mostrar el hiperárbol dirigido que representa al MEM seleccionado. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-26 | Modificar condiciones |
|---|---|
| **Objetivo** | Visualizar el efecto de la modificación en los valores de las condiciones en el MEM que se está visualizando. |
| **Procedimiento** | Modificar el valor de las condiciones que aparecen junto al MEM. |
| **Salida esperada** | La representación del MEM deberá modificarse, mostrándose en un color más claro los arcos correspondientes a las interpretaciones que son incompatibles con los valores de las condiciones seleccionados. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-27 | Generar PDF |
|---|---|
| **Objetivo** | Almacenar la representación del MEM seleccionado en un archivo PDF. |
| **Procedimiento** | Seleccionar la opción "Guardar PDF" y especificar la ruta y el nombre de archivo en el que se almacenará la representación. |
| **Salida esperada** | Se generará un archivo en formato PDF que contenga la representación del MEM actualmente seleccionado. |
| **Resultado** | El resultado obtenido es correcto. |

| CP-28 | Consultar ayuda |
|---|---|
| **Objetivo** | Consultar la ayuda de la aplicación. |
| **Procedimiento** | Seleccionar la opción "Ayuda". |
| **Salida esperada** | Se mostrará una nueva ventana con ayuda sobre el funcionamiento de la aplicación. |
| **Resultado** | El resultado obtenido es correcto. |

# 8

# Manual de instalación

Para instalar la aplicación, simplemente extraiga el contenido del archivo PCs4.tar que encontrará en el CD en la ubicación deseada.

Para iniciar la aplicación de cálculo deberá ejecutar el archivo PCs4.exe (Windows) o PCs4.jar (resto de sistemas operativos).

Para iniciar el visor ejecutar el archivo Visor.exe (Windows) o Visor.jar (resto de sistemas operativos).

Deberá haber instalado previamente una versión de Java reciente (igual o superior a la 1.7) desde la siguiente dirección: https://www.java.com/es/download/.

# 9

# Manual de usuario

A continuación se mostrará el manual de usuario. Dividiremos el manual en dos partes: la primera explicará el funcionamiento de los nuevos algoritmos incluidos en la aplicación de cálculo y la segunda se centrará en las funciones del Visor.

## 9.1. Manual de la aplicación de cálculo

Para usar los algoritmos seleccionados procederemos de la siguiente forma:

1. En el menú "Algoritmo" seleccionamos la opción "Cargar algoritmo" o pulsamos el segundo botón de la barra de inicio rápido (el que tiene forma de rueda dentada)

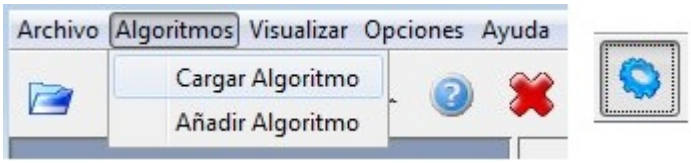



***Figura 9.1: Cargar algoritmo (1)***

Posteriormente, seleccionamos el algoritmo que deseemos cargar (los algoritmos creados en este TFG son "CPC TreeSet" y "CPC TreeSet con condiciones"). Para confirmar pulsamos el botón "OK". También podremos cargar el algoritmo pulsando el segundo botón de la barra de inicio rápido (la que tiene forma de rueda dentada).


***Figura 9.2: Cargar algoritmo (2)***

2. Ahora debemos cargar un archivo de descripción del sistema. Para ello podemos pulsar el primer botón de la barra de inicio rápido (el que muestra una carpeta) o seleccionar la opción "Cargar sistema…" del menú "Archivo". A continuación se mostrará una ventana en la que podemos elegir el archivo que deseamos cargar.


***Figura 9.3: Cargar sistema (1)***

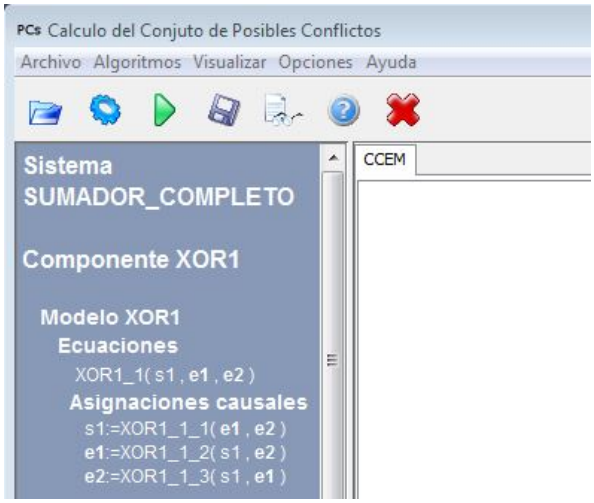



***Figura 9.4: Cargar sistema (2)***

3. Para calcular el CCEM, oprimiremos el botón "Calcular CCEM" que aparece en la pestaña CCEM.

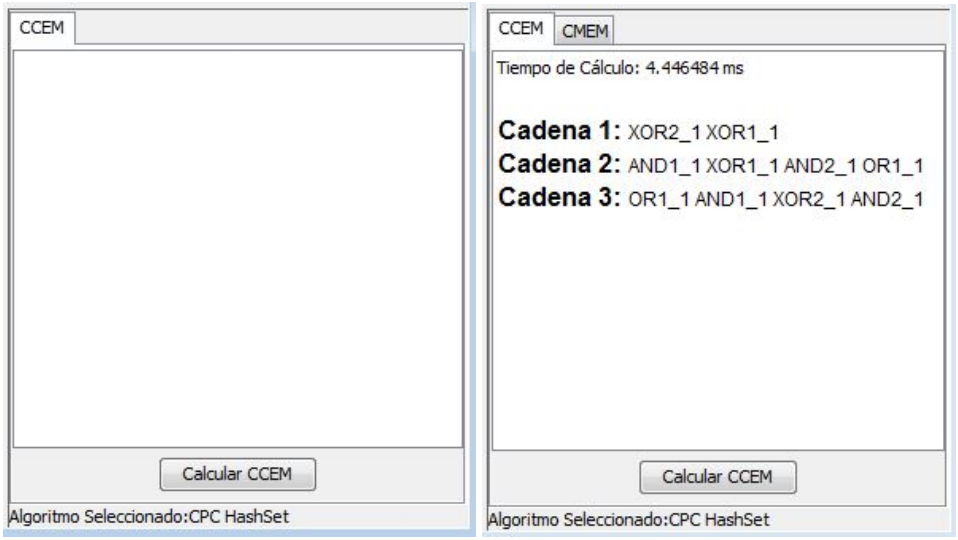



***Figura 9.5: Calcular CCEM***

4. Una vez finalizado el paso anterior, podremos proceder a calcular el CMEM. Para ello iremos a la pestaña "CMEM" y oprimiremos el botón "Calcular CMEM".

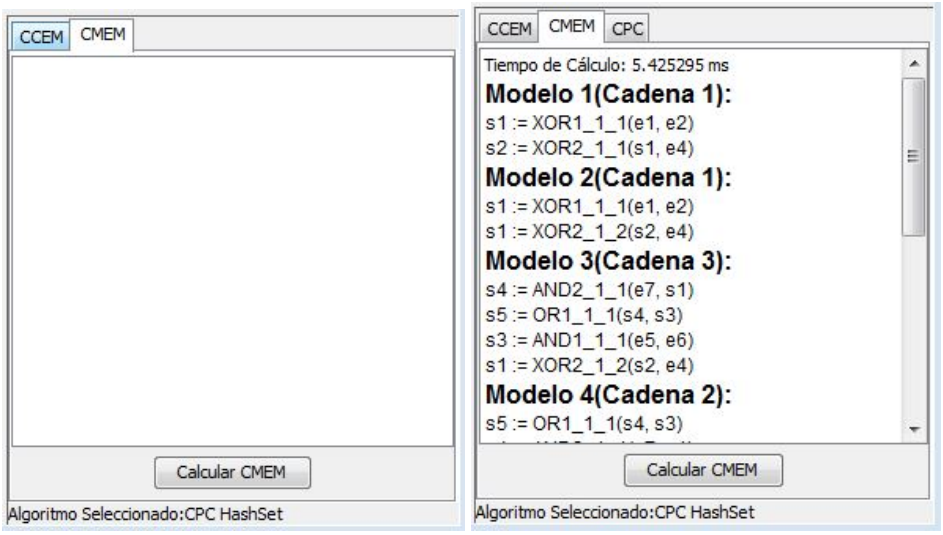



***Figura 9.6: Calcular CMEM***

5. Finalmente tendremos disponible la opción "Calcular CPC" en la nueva pestaña "CPC" que ha aparecido para obtener la lista definitiva de posibles conflictos

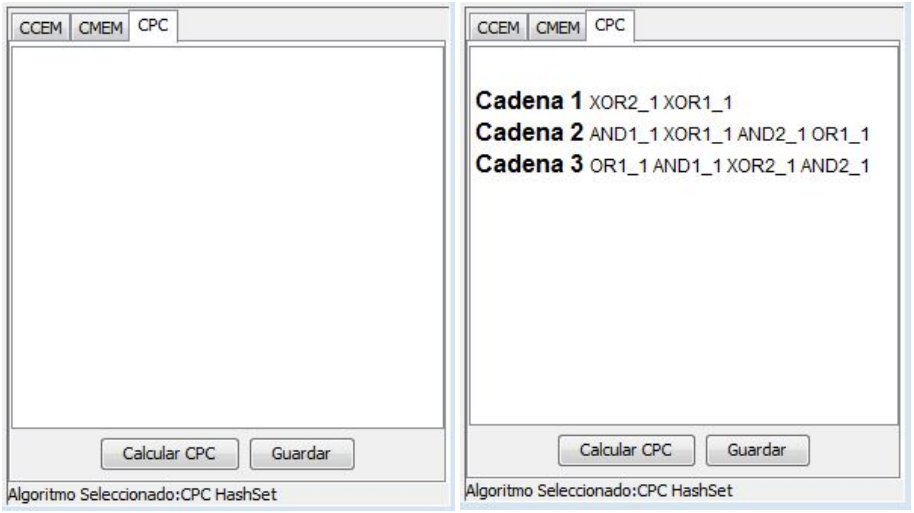



***Figura 9.7: Calcular CPC***

Estas tres últimas operaciones pueden ejecutarse simultáneamente pulsando el tercer botón de la barra de inicio rápido (el que muestra un triángulo verde).

***Figura 9.8: Realizar cálculos***

6. Para guardar los resultados obtenidos pulsaremos el botón "Guardar" o el cuarto botón de la barra de inicio rápido (disquete).

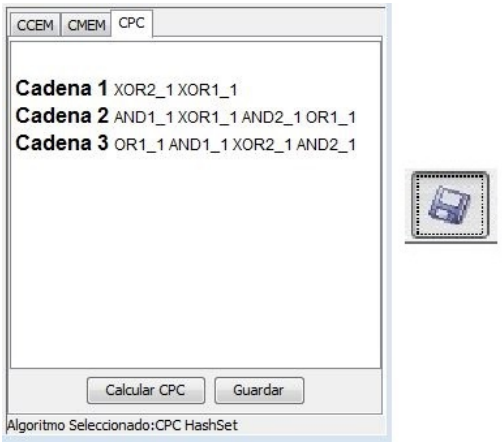



***Figura 9.9: Guardar resultados (1)***

Se nos mostrará una ventana en la que podremos elegir el formato del archivo de salida ("XML" o "Texto plano") y, posteriormente, otra ventana para elegir la ubicación y el nombre del archivo.

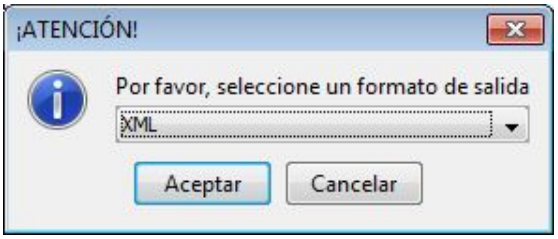



***Figura 9.10: Guardar resultados (2)***

7. Para ejecutar el Visor elegiremos la opción "Visualizar resultados" del menú "Visualizar" o pulsaremos el quinto botón de la barra de inicio rápido (documento y gafas). En caso de que hayamos guardado los resultados en formato XML podremos visualizar directamente los resultados que acabamos de obtener. Es imprescindible tener resultados generados y guardados para poder visualizarlos.

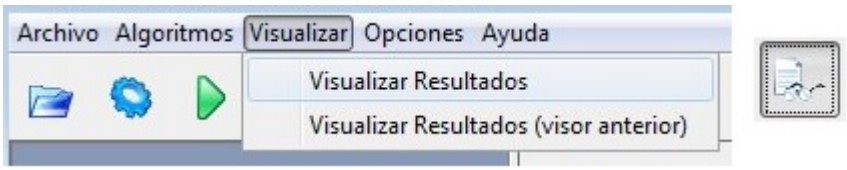



***Figura 9.11: Visualizar resultados***

# 9.2. Manual del visor

A continuación describiremos las operaciones que podemos realizar con el Visor:

1. Carga de resultados: podremos realizarla seleccionando la opción "Cargar PCs…" del menú "Archivo" o pulsando el primer botón de la barra de inicio rápido (carpeta). A continuación se mostrará una ventana en la que elegiremos el archivo XML que deseamos cargar.

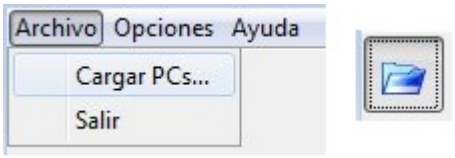



***Figura 9.12: Cargar resultados***

2. Reducción/ampliación de la representación: podremos controlar el zoom de la representación usando la rueda del ratón o los botones "-" y "+" que aparecen en la parte inferior de la ventana.

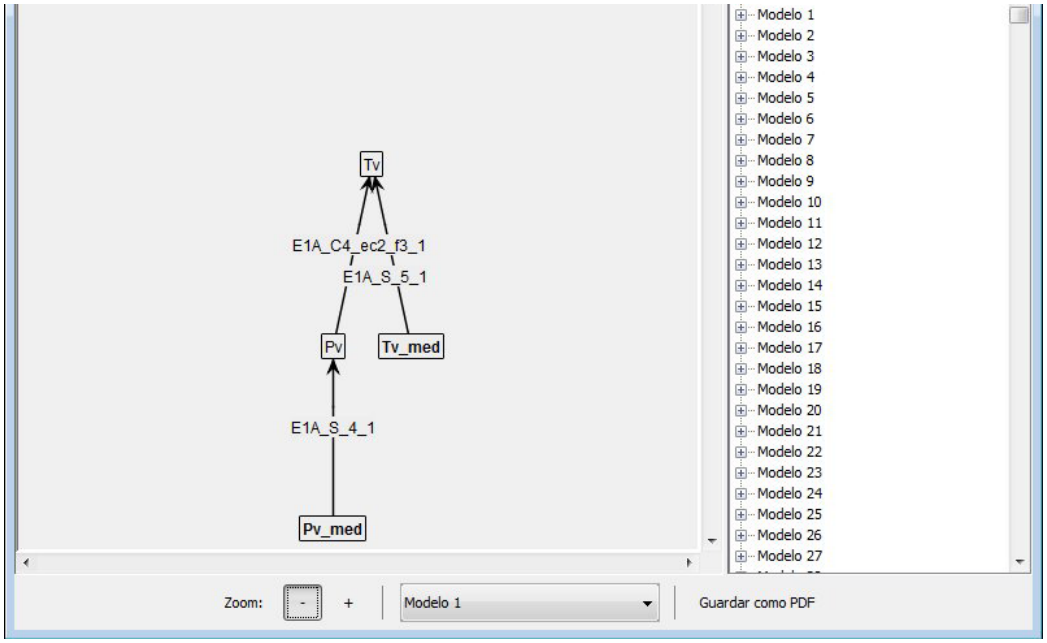



***Figura 9.13: Reducir vista***

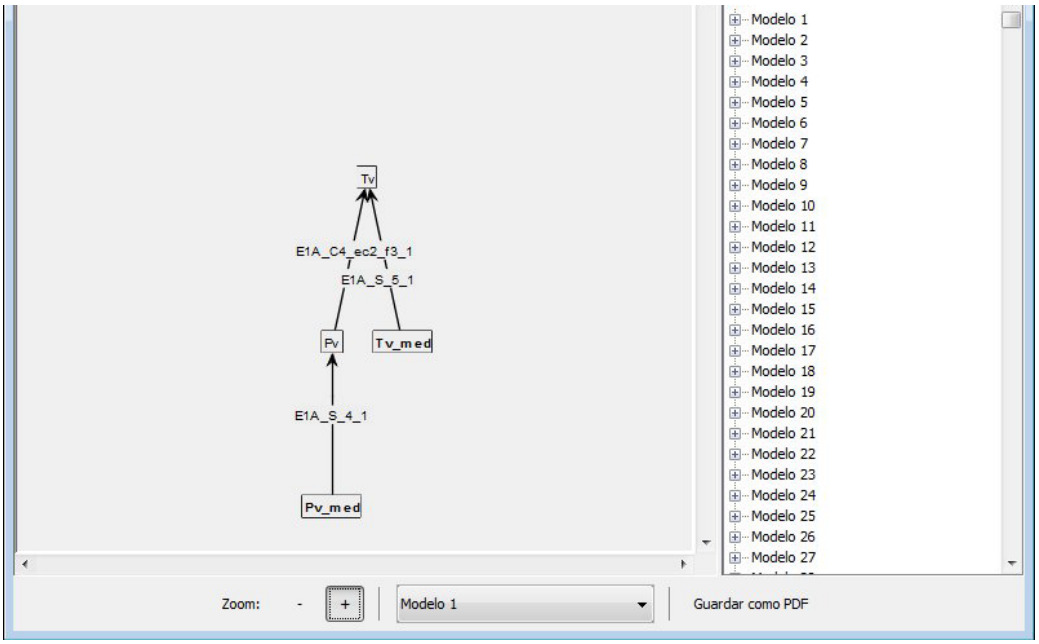



***Figura 9.14: Ampliar vista***

3. Traslación de la representación: podremos desplazarnos por la representación haciendo clic y manteniendo el botón izquierdo del ratón y desplazando el cursor o usando las barras de desplazamiento que aparecen en la ventana de visualización.
4. Selección de MEM: en el menú desplegable que aparece en la parte inferior de la ventana podremos elegir el MEM que deseamos visualizar.

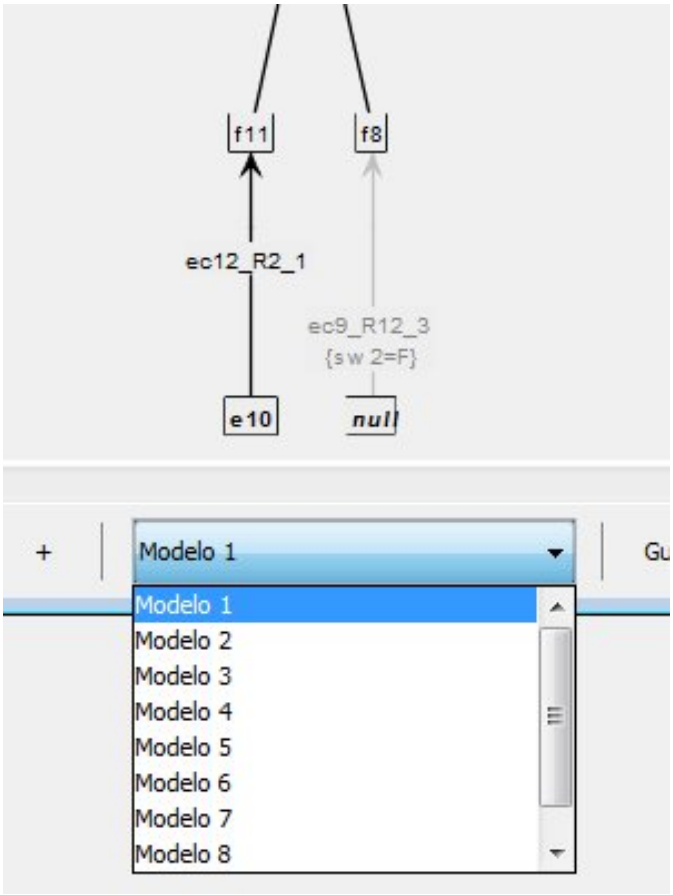



***Figura 9.15: Seleccionar MEM***

5. Guardar como PDF: para guardar la visualización de un MEM en un archivo PDF pulsaremos el botón "Guardar como PDF" que aparece en la parte inferior de la ventana.

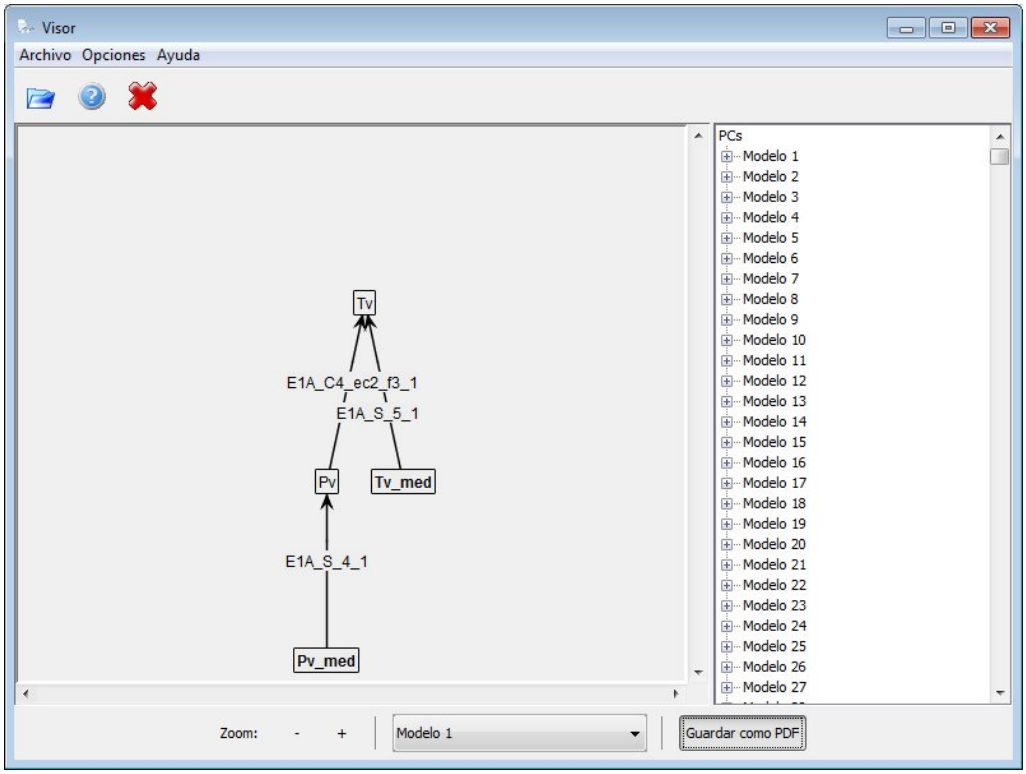



***Figura 9.16: Guardar PDF***

6. Modificación de condiciones: en caso de que el MEM seleccionado esté afectado por alguna condición ésta aparecerá en la parte izquierda de la ventana, junto con un desplegable que muestra su valor actual. Seleccionando otro valor en dicho desplegable veremos qué interpretaciones dejarán de ser compatibles con el nuevo valor seleccionado.

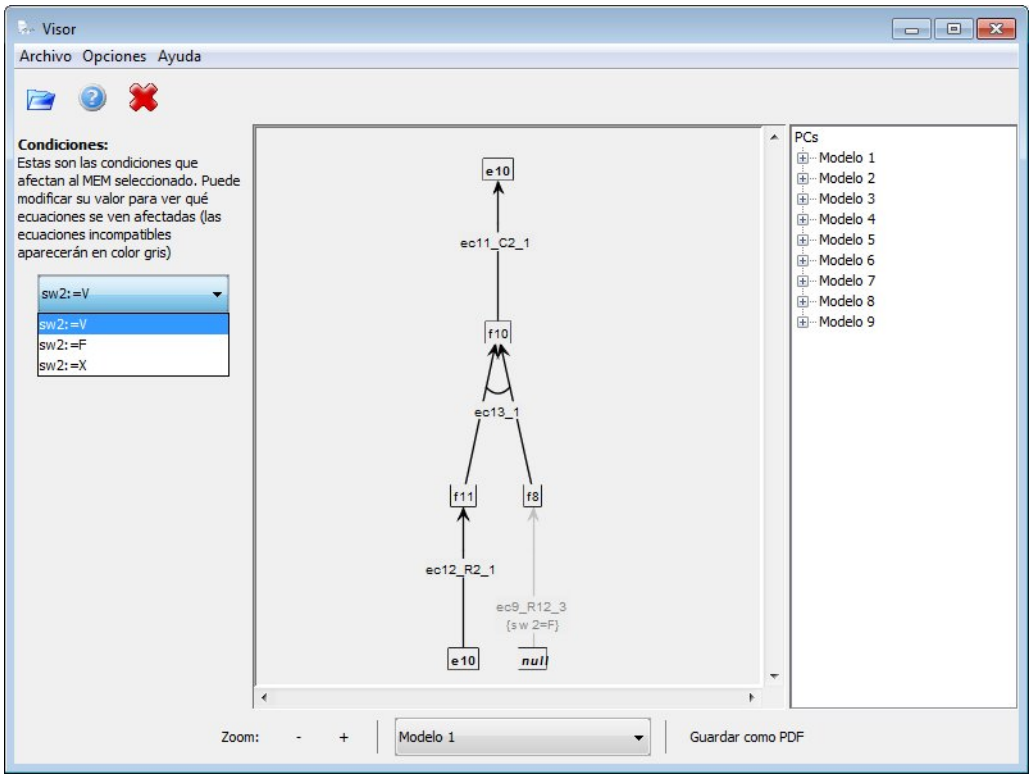



***Figura 9.17: Modificar condiciones***

7. Información sobre el CPC: en la parte derecha de la pantalla veremos una lista con los MEM de las CEM que forman parte del CPC. Pulsando el botón "+" que aparece junto a cada MEM podremos ver las asignaciones causales (interpretaciones) que forman el MEM y las ecuaciones que contiene la CEM a partir de la cual se ha calculado.

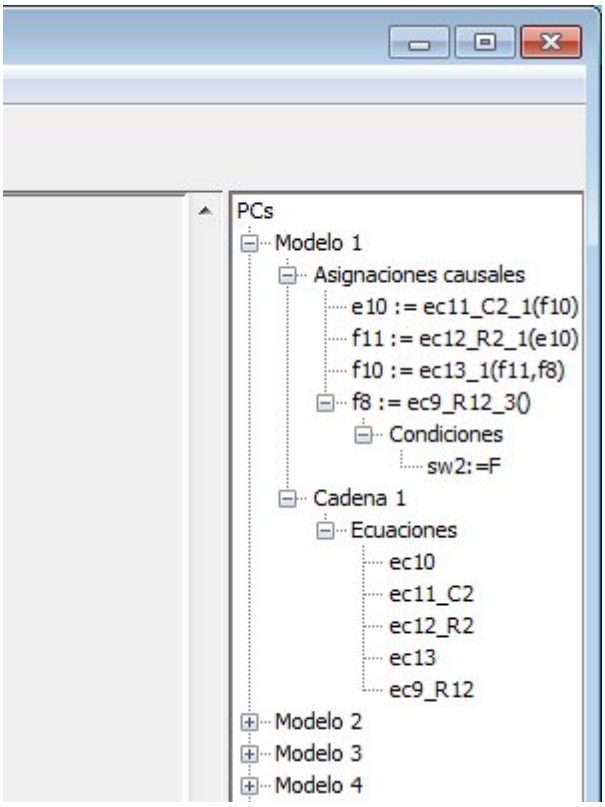



***Figura 9.18: Información sobre los PCs***

8. Añadir nuevo idioma: para esto seleccionaremos la opción "Añadir idioma…" del menú "Opciones" y posteriormente indicaremos el archivo en el que se encuentra la configuración del idioma a añadir.

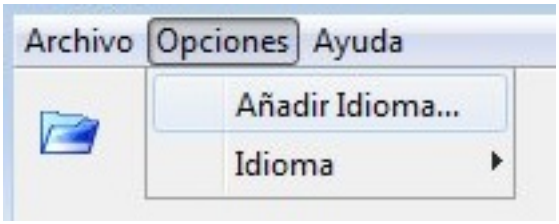



***Figura 9.19: Añadir idioma***

9. Cambiar idioma: para cambiar el idioma de la interfaz seleccionaremos el idioma que deseamos en el menú "Opciones", submenú "Idioma".

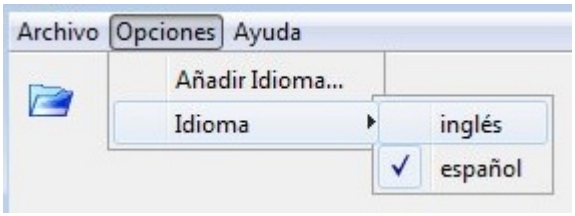



***Figura 9.20: Cambiar idioma***

10. Consultar ayuda: podremos acceder a la ayuda con la opción "Contenidos" del menú "Ayuda" o pulsando el segundo botón de la barra de acceso rápido (?).

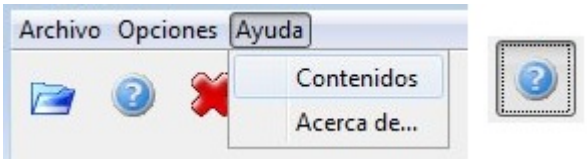



***Figura 9.21: Consultar ayuda (1)***

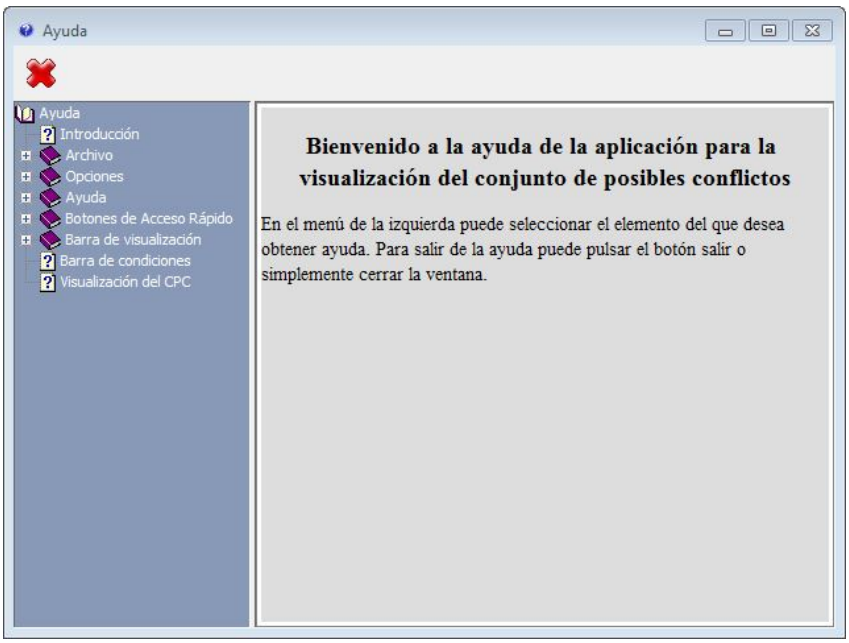



***Figura 9.22: Consultar ayuda (2)***

# 10

# Conclusiones y posibles ampliaciones futuras

En este capítulo se presentan las conclusiones obtenidas tras el desarrollo de este proyecto y se ofrecen algunas sugerencias para una posible ampliación futura.

## 10.1 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 las versiones previas de la aplicación y los distintos algoritmos y buscar posibles formas de mejora.
- Se ha realizado una nueva versión del algoritmo de cálculo de posibles conflictos básico.
- Se ha realizado una nueva versión del algoritmo de cálculo de posibles conflictos con condiciones.
- Se ha mejorado la representación gráfica actual de los posibles conflictos.

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

Además, se ha comprobado de forma práctica la importancia de un análisis del rendimiento temporal y de consumo de memoria de los algoritmos antes de su implementación.

Cabe destacar también la realización del proceso de búsqueda de soluciones ya implementadas para la solución de un problema relativamente complejo (la visualización gráfica de hipergrafos dirigidos).

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.

## 10.2 Ampliaciones Futuras

Para finalizar, se proponen las siguientes ampliaciones al presente TFG:

- Reimplementar el resto de algoritmos desarrollados usando estructuras de datos ordenadas y algoritmos de búsqueda optimizados.
- Añadir la visualización de ciclos (se puede encontrar más información sobre los ciclos en la memoria de Raúl Sánchez [San07]).

# 11

# Glosario

- ***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
- ***CPC***: Conjunto de Posibles Conflictos
- ***DBM***: Diagnóstico Basado en Modelos
- ***TFG***: Trabajo de Fin de Grado.
- ***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.
- ***PC***: Posible Conflicto: Cualquier CEM que presente al menos un MEM.

# 12

# Bibliografía

[Abu07] AbuGraph (2007). *abugraph.sourceforge.net*. Consultado en octubre de 2013, desde http://abugraph.sourceforge.net/.
Aplicación para la representación de grafos dirigidos. Permite obtener los grafos de diversas formas (incluso de forma remota mediante un terminal telnet).

[Bal98] Balakrishnan, K., & Honavar, V. (1998). Intelligent diagnosis systems. *Journal of Intelligent Systems*, 8(3-4), 239-290.

[Bib14] bibme (2014). *bibme.org*. Consultado en junio de 2013 desde http://www.bibme.org/ citation-guide/.
Guía sobre el formato de citas y referencias bibliográficas.

[Gin05] GINY (2005). *csbi.sourceforge.net*. Consultado en octubre de 2013 desde http://csbi.sourceforge.net/.
Biblioteca de funciones que proporciona procedimientos para implementar y visualizar varios tipos de grafos.

[Gra10] Grappa (2010). *research.att.com*. Consultado en octubre de 2013 desde http://www2.research.att.com/~john/Grappa/.
Biblioteca de funciones que proporciona procedimientos para implementar y visualizar varios tipos de grafos.

[Hgr05] Hypergraph (2005). *hypergraph.sourceforge.net*. Consultado en octubre de 2013 desde http://hypergraph.sourceforge.net/.
Biblioteca para la implementación y visualización de árboles hiperbólicos.

[Htr01] HyperTree (2001). *hypertree.sourceforge.net*. Consultado en octubre de 2013 desde http://hypertree.sourceforge.net/.

Biblioteca para la implementación y visualización de árboles hiperbólicos

[Ite14] iText (2014). *sourceforge.net/projects/itext*. Consultado en mayo de 2014 desde http://sourceforge.net/projects/itext/.
Biblioteca de funciones para la gestión de documentos en formato PDF.

[Jav14] JavaDoc (2014). *docs.oracle.com*. Consultado en mayo de 2014 desde http://docs.oracle.com/javase/7/docs/api/.
Documentación sobre las clases de la versión 1.7 de Java.

[Jdi03] JDigraph (2003). *sourceforge.net/projects/jdigraph*. Consultado en octubre de 2013 desde http://sourceforge.net/projects/jdigraph/.
Biblioteca de funciones que proporciona procedimientos para implementar y visualizar grafos dirigidos.

[Jpo07] JPowerGraph (2007). *jpowergraph.sourceforge.net*. Consultado en octubre de 2013 desde http://jpowergraph.sourceforge.net/.
Biblioteca de funciones que proporciona procedimientos para implementar y visualizar varios tipos de grafos.

[Jun10] JUNG (2010). *jung.sourceforge.net*. Consultado en abril de 2014 desde http://jung.sourceforge.net/.
Biblioteca de funciones que proporciona procedimientos para implementar y visualizar varios tipos de grafos.

[Laj07] Lajo, R. (2007). Herramienta de cálculo del conjunto de Posibles Conflictos. *Proyecto Fin de Carrera, Ingeniería Superior Informática. Escuela Técnica Superior de Ingeniería Informática. Universidad de Valladolid*.

[Pul97] Pulido, B., & Alonso, C. (1997). La información sobre la topología y el dominio simplifica el cálculo de conflictos minimales para el Diagnóstico basado en Modelos. *VII Conferencia de la Asociación Española para la Inteligencia Artificial*.

[Pul01] Pulido, B., & Alonso, C. (2001). Revision del concepto de posible conflicto como tecnica de pre-compilacion. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, *5*(14), 41-53.

[San07] Sánchez , R. (2007). Ampliación de la Plataforma para el Cálculo y Visualización del Conjunto de Posibles Conflictos. *Proyecto Fin de Carrera, Ingeniería Superior Informática. Escuela Técnica Superior de Ingeniería Informática. Universidad de Valladolid*.

[Tou02] TouchGraph (2002). *touchgraph.sourceforge.net*. Consultado en octubre de 2013 desde http://touchgraph.sourceforge.net/.
Biblioteca centrada en la representación gráfica de grafos simples.

[UPE14] UPEDU (2014). *upedu.org*. Consultado en mayo de 2014 desde http://www.upedu.org/.
Web de UPEDU (Unified Process for EDUcation), proporciona información sobre la metodología usada para realizar este TFG, así como plantillas de la documentación generada.

[Wil04] WilmaScope 3D Graph Visualization (2004). *wilma.sourceforge.net*. Consultado en octubre de 2013 desde http://wilma.sourceforge.net/.
Aplicación para la visualización de grafos en 3D

[Xom13] XOM (2013). *xom.nu*. Consultado en diciembre de 2013 desde http://www.xom.nu/.
Biblioteca Java de lectura y creación de documentos en formato XML.

# Apéndices

## Apéndice A: DTD de fichero de entrada

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by LEGO (LEGO) -->
<!ELEMENT sistema (componente+, OBS+, NOBS*)>
<!ATTLIST sistema
   nombre ID #REQUIRED
   nombre_onto CDATA #REQUIRED
>
<!ELEMENT componente (ecuacion)+>
<!ATTLIST componente
   nombre_com ID #REQUIRED
   tipo CDATA #REQUIRED
>
<!ELEMENT ecuacion (variable+, interpretacion+,condicion*)>
<!ATTLIST ecuacion
   id ID #REQUIRED
   tipo (diferencial | estatica) #IMPLIED
>
<!ELEMENT variable EMPTY>
<!ATTLIST variable
   nombre ID #REQUIRED
   observada CDATA #REQUIRED
   tipo CDATA #IMPLIED
>
<!ELEMENT interpretacion (condicion*)>
<!ATTLIST interpretacion
   nombre ID #REQUIRED
   cabeza IDREFS #REQUIRED
   cola IDREFS #REQUIRED
   tipo (integral | derivada) #IMPLIED
>
<!ELEMENT OBS EMPTY>
<!ATTLIST OBS
   variables IDREFS #REQUIRED
>
<!ELEMENT NOBS EMPTY>
<!ATTLIST NOBS
   variables IDREFS #REQUIRED
>
<!ELEMENT condicion (restriccion+)>
<!ATTLIST condicion
   nombre CDATA #REQUIRED
>
```

```
<!ELEMENT restriccion (#PCDATA)>
<!ATTLIST restriccion
   id ID #REQUIRED
   valor (T | F | X) #REQUIRED
>
```

# Apéndice B: DTD de fichero de salida

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT RESULTADOS (CCEM+, sistema)>
<!ELEMENT CCEM (CADENA+)>
<!ATTLIST CCEM
   nom_stma ID #REQUIRED
>
<!ELEMENT CADENA (MEM*, condicion*)>
<!ATTLIST CADENA
   id_cadena CDATA #REQUIRED
   si_PC CDATA #REQUIRED
   ecuaciones IDREFS #REQUIRED
>
<!ELEMENT MEM (ciclo*, solucion*, condicion*)>
<!ATTLIST MEM
   id CDATA #REQUIRED
   usado CDATA #REQUIRED
   interpretaciones IDREFS #REQUIRED
>
<!ELEMENT ciclo (#PCDATA)>
<!ATTLIST ciclo
   id CDATA #REQUIRED
   roto (si | no) #REQUIRED
>
<!ELEMENT solucion (interpretacion+)>
<!ELEMENT interpretacion EMPTY>
<!ATTLIST interpretacion
   nombre ID #REQUIRED
   cabeza IDREFS #REQUIRED
   cola IDREFS #REQUIRED
>
<!ELEMENT sistema (componente+, OBS+, NOBS*, restriccion*)>
<!ATTLIST sistema
   nombre ID #REQUIRED
   nombre_onto CDATA #REQUIRED
>
<!ELEMENT componente (ecuacion+)>
<!ATTLIST componente
   nombre_com ID #REQUIRED
   tipo CDATA #REQUIRED
>
<!ELEMENT ecuacion (variable+, interpretacion+,condicion*)>
<!ATTLIST ecuacion
   id ID #REQUIRED
   tipo (diferencial | estatica) #IMPLIED
>
<!ELEMENT variable EMPTY>
<!ATTLIST variable
   nombre ID #REQUIRED
   observada CDATA #REQUIRED
   tipo CDATA #IMPLIED
>
```

```
<!ELEMENT interpretacion (condicion*)>
<!ATTLIST interpretacion
   nombre ID #REQUIRED
   cabeza IDREFS #REQUIRED
   cola IDREFS #REQUIRED
   tipo (integral | derivada) #IMPLIED
>
<!ELEMENT OBS EMPTY>
<!ATTLIST OBS
   variables IDREFS #REQUIRED
>
<!ELEMENT NOBS EMPTY>
<!ATTLIST NOBS
   variables IDREFS #REQUIRED
>
<!ELEMENT restriccion EMPTY>
<!ATTLIST restriccion
   nombre ID #REQUIRED
   descripcion CDATA #IMPLIED
>
<!ELEMENT condicion EMPTY>
<!ATTLIST condicion
   nombre IDREF #REQUIRED
   valor (T | F | X) #REQUIRED
>
```

# Apéndice C: Contenido del CD

- ***CodigoFuente***: Carpeta que contiene el código fuente de todas las clases de la aplicación.
- ***DocumentacionAdicional***: Carpeta con documentación para el programador sobre las clases del visor.
- ***Manuales***: Carpeta que contiene los manuales de instalación y uso de la aplicación desarrollada.
- ***memoria.pdf***: Archivo PDF con el contenido del presente documento.
- ***PCs4.jar***: Archivo extraíble que contiene una versión ejecutable de la aplicación desarrollada.