Programa Oficial de Postgrado en Ciencias, Tecnología y Computación
Máster en Computación

Facultad de Ciencias – Universidad de Cantabria

TraceDisplayer: Intérprete de trazas de sistemas de tiempo real analizados en el entorno MAST

Emilio Ruiz Gutiérrez
emilioruizg@yahoo.es

Directores:
José María Drake Moyano
César Cuevas Cuesta
Grupo de Computadores y Tiempo Real
Departamento de Electrónica y Computadores

Santander, Octubre de 2012
Curso 2011/2012

Índice de Contenido:

1. Sistemas de tiempo real y entorno de diseño MAST 1
1.1. Sistemas de tiempo real. 1
1.2. Modelo MAST de un sistema de tiempo real 2
1.3. Trazas de ejecución 5
1.3.1. Breve introducción a XML 5
1.3.2. Empleo de XML con Java 7
1.3.3. Librería SWT 9
1.4. Objetivo del trabajo 9
2. Monitorización de trazas 10
2.1. Descripción de un fichero de trazas 10
2.2. Presentación de las trazas: organización, reactividad e iconos 11
2.2.1 Jerarquía de ventanas 11
2.2.2 Main_Win 12
2.2.3 All_E2EF_Win 14
2.2.4 Aux_All_E2EF_Win 15
2.2.5 Detailed_E2EF_Win 16
2.2.6 Aux_Detailed_E2EF_Win 16
2.2.7 All_Resources_Win 17
2.2.8 Aux_All_Resources_Win 17
2.2.9 Detailed_Resource_Win 18
2.3. Gestión del tiempo 18
2.4. Información reactiva de ayuda 19
3. Diseño del monitor de trazas 20
3.1. Criterios de diseño 20
3.2. Arquitectura de la aplicación 21
3.2.1. Paquete *core* 21
3.2.2. Paquete *visualization* 23
3.2.3. Paquete *graphicItems* 30
3.3. Casos de uso 30
3.3.1. *Especificación del fichero de trazas.* 30
3.3.2. *Representación de la actividad global del sistema* 31
3.3.3. Manipulación del intervalo temporal. 32
4. Ejemplo de monitorización de un sistema de tiempo real 34
4.1. PowerDistortion: Ejemplo de sistema de tiempo real 34
4.2. Modelo MAST del ejemplo 37
4.3 Generación del fichero de trazas 40
4.4 Visualización del fichero de trazas 41
5 Conclusiones 46
6 Referencias 47
7 Anexos 48
7.3 Mast_Trace.xsd 48

Índice de figuras:

Figura 1. Aplicación de tiempo real y entorno físico. 1
Figura 2: Elementos de modelado de la metodología MAST utilizados en este trabajo. 5
Figura 3. Generación de la estructura en árbol a partir del fichero. 8
Figura 4. Esquema del analizador de texto 8
Figura 5. Jerarquía de ventana de interacción con el usuario. 12
Figura 6. Ventana inicial de la interfaz con la pestaña de las transacciones seleccionada. 13
Figura 7. Panel inicial de la interfaz con la pestaña de los recursos seleccionada. 14
Figura 8. Ejemplo de la visualización gráfica de la evolución temporal de dos transacciones. 15
Figura 9. Ventana Aux_All_E2EF_WIN con las leyendas de la ventana 16
Figura 10. Detailed_E2EF_Win: visualización de una transacción junto a los recursos. 16
Figura 11. Aux_Detailed_E2EF_WIN con las leyendas de la ventana Detailed_E2EF_WIN 17
Figura 12. All_Resources_Win con la visualización de un procesador y un mutex. 17
Figura 13. Aux_All_Resource_Win con la leyenda de la ventana All_Resources_WIN 18
Figura 14. Detailed_Resources_Win con la visualización de un procesador un mutex y una transacción. 18
Figura 15. Panel de gestión del tiempo de cualquier ventana gráfica. 19
Figura 16. Ejemplo de visualización de la información asociada de un elemento 19
Figura 17. Información de entrada y salida de la aplicación JMastTraces. 20
Figura 18. Relaciones entre las clases del paquete core. 22
Figura 19. Detalle de la clase VisRepository. 23
Figura 20. Jerarquía de clases del paquete visualization. 24
Figura 21. VisProcessingResource 25
Figura 22. VisMutex 26
Figura 23. VisSchedulableResource 27
Figura 24. E2EF. 27
Figura 25. VisSegment 28
Figura 26. VisWorkloadEvent 29
Figura 27. Jerarquía de clases del paquete graphicItems 30
Figura 28. Diagrama de actividad para Enter trace file. 31
Figura 29. Diagrama de actividad para Display All E2EFs 32
Figura 30. Diagrama de actividad para Time Control GUI. 33
Figura 31.Plataforma y módulos lógicos de la aplicación PowerDistortion. 35
Figura 32. Actividad de la transacción WfProcessing. 36
Figura 33. Actividad de la transacción DistortionAdjust. 37
Figura 34. Ventana de lanzamiento de las herramientas del entorno MAST. 41
Figura 35. Ventana de control del simulador. JSimMast. 41
Figura 36. Ventana de control de selección de las trazas a visualizar (GUI). 42
Figura 37. Ventana que muestra la lista de recursos de procesamiento de las trazas a visualizar. 43
Figura 38. Selección de la transacción, WfProcessing_2e2f, y sus elementos asociados 44
Figura 39. Visualización de la transacción WfProcessing_e2ef junto a los recursos de la traza. 45
Figura 40. Visualización de la transacción WfProcessing_2e2f, junto a los recursos de la traza en el intervalo de tiempo Initial Time = 490170318ns y Final Time = 492123910ns 45

Índice de tablas:

Tabla 1. Glosario de términos en los modelos de tiempo real MAST. 3
Tabla 2. Modelo de tiempo real de la situación de tiempo real..... 37

1. Sistemas de tiempo real y entorno de diseño MAST

1.1. Sistemas de tiempo real.

Un sistema de tiempo real [1] es un sistema informático al que, por interaccionar con un entorno físico externo, se le requiere que atienda eventos hardware procedentes de él o procedentes de relojes internos, que aunque se generen asíncronamente respecto de su estado de ejecución, han de ser atendidos y las correspondientes respuestas generadas en plazos temporales acotados.

Los sistemas de tiempo real tienen un funcionamiento correcto no sólo cuando procesan la información de acuerdo con su especificación funcional, sino también cuando generan los resultados y ejecutan las acciones de interacción con el entorno en los tiempos previstos en su especificación de comportamiento temporal. Los sistemas de tiempo real pueden tener tamaño y complejidad muy variada: se extienden desde sistemas embebidos que controlan pequeños electrodomésticos hasta grandes sistemas distribuidos que controlan procesos industriales complejos.

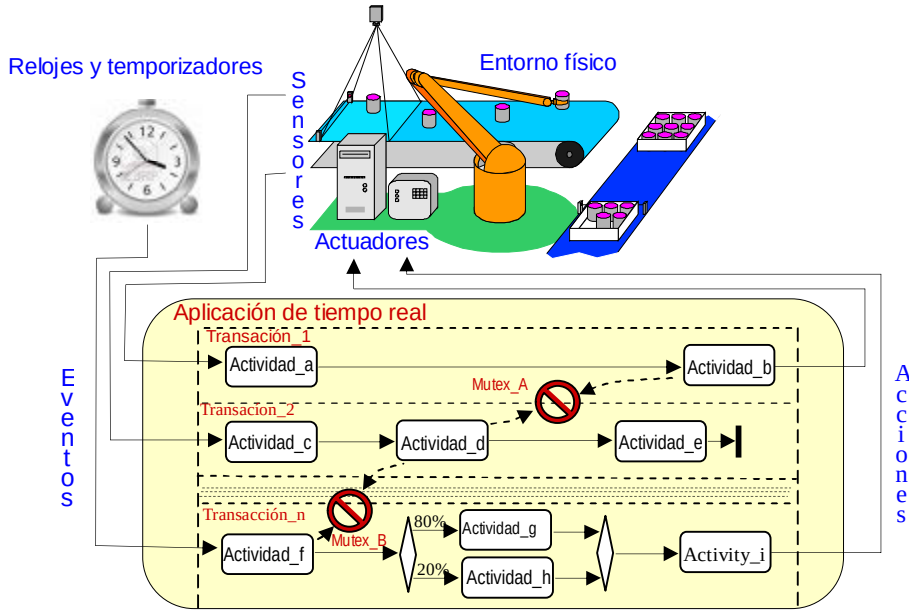

Figura 1. Aplicación de tiempo real y entorno físico

Según la naturaleza de sus restricciones temporales, los sistemas de tiempo real se clasifican [2] en sistemas de tiempo real estrictos (*hard real-time systems*) y sistemas de tiempo real laxos (*soft real-time systems*). En los primeros, si un plazo establecido en la especificación de la aplicación se pierde, esto representa un error fatal e irrecuperable y, por tanto, no puede ser permitido en ningún caso. Un ejemplo de este tipo es el sistema de control del freno de un coche, al que se requiere que desde que se aprieta el pedal hasta que se acciona el freno no se produzca un retraso superior a una cota preestablecida de algunos milisegundos. En los sistemas de tiempo real laxos se permite el incumplimiento aislado o bajo tasas especificadas de ciertos plazos en respuestas que son consideradas de menor importancia. Por ejemplo, en caso de sobrecarga del sistema, una pérdida de un plazo temporal no constituye un fallo, sino únicamente una reducción de sus prestaciones. En este caso, la especificación temporal se formula como tasas de fallos que deben ser satisfechas estadísticamente. Ejemplos de este tipo son los sistemas de procesado de señal de vídeo, en los que la pérdida de algunos plazos repercute en una reducción de la calidad de la imagen.

Con la tecnología actual, los programas de tiempo real se diseñan como programas concurrentes compuestos por múltiples *threads*, cada uno de los cuales gestiona la ejecución de la respuesta a un evento. De esta forma, asignando a los parámetros de planificación de cada *thread* los valores adecuados, se puede conseguir que cada respuesta satisfaga sus requisitos temporales. La diferencia entre un programa concurrente y uno de tiempo real es que este último añade especificaciones de tiempo concretas para ciertas operaciones, mientras que el programa concurrente simplemente especifica la posibilidad o conveniencia de realizar operaciones simultáneas, pero sin imponer plazos concretos.

Para garantizar los requerimientos de respuesta de los sistemas de tiempo real, no sólo es necesario garantizar que las actividades propias de la aplicación tengan tiempos de ejecución acotados, sino también que los recursos de la plataforma (hardware y sistema operativo) y de sincronismo (semáforos, mutexes, etc.) se encuentren disponibles cuando se necesiten y den lugar a tiempos de bloqueo acotados. Los sistemas de tiempo real sólo pueden ser diseñados si el sistema operativo es de tiempo real, esto es, si ofrece algoritmos de planificación que gestionen de forma predecible el acceso al procesador, si ofrece mecanismos de sincronización que arbitren de forma predecible el acceso exclusivo a los recursos compartidos con otras aplicaciones concurrentes y si ofrece servicios con tiempo de respuesta acotado.

Para que se satisfagan los requisitos temporales de un sistema de tiempo real, su diseño requiere la configuración de todos los elementos que participan en la planificación de la ejecución de las actividades que constituyen las respuestas a los eventos. Ésta es una tarea compleja que requiere tener en consideración las secuencias de actividades que constituyen las respuestas que proporciona la aplicación, la cantidad de computación que requiere la ejecución de cada una de estas actividades, la capacidad de los procesadores y de las redes de comunicación para ejecutarlas, las interacciones que existen entre las diferentes respuestas como consecuencia de que utilizan de forma exclusiva ciertos recursos comunes y los patrones de ocurrencia de los eventos que requieren las respuestas. Toda esta información debe combinarse de la forma más desfavorable para estimar los tiempos de ejecución de las respuestas en el peor caso, y de esta forma garantizar que son inferiores a los plazos especificados para ellas. La organización, gestión y procesado de toda información es una tarea compleja que requiere de una metodología de modelado que identifique la información que ha de ser utilizada, la construcción del modelo que describe el comportamiento del sistema y de un conjunto de herramientas que la procesen y que, o bien permitan garantizar que se cumplen todos los requisitos temporales, o bien obtengan la configuración del sistema para que eso ocurra.

El presente trabajo aporta una nueva herramienta para la supervisión de los sistemas de tiempo real en el entorno de diseño de sistemas de tiempo real MAST [4], desarrollado por el Grupo de Computadores y Tiempo Real de la Universidad de Cantabria.

1.2. Modelo MAST de un sistema de tiempo real

La metodología de modelado de tiempo real que se utiliza en el entorno MAST se deriva de un modelo de referencia ampliamente aceptado por los expertos de este campo [3]. Según esta metodología, el modelo de un sistema de tiempo real se compone de cuatro elementos básicos:

- El modelo de comportamiento temporal de los elementos lógicos: describe desde el punto de vista del comportamiento temporal todos aquellos elementos de la aplicación que participan en la construcción lógica de las respuestas que constituyen la aplicación de tiempo real.
- Las operaciones: describen el tiempo de procesamiento que requieren las actividades, los *threads* y canales de comunicación en los que se planifica su ejecución y los mutexes que garantizan el acceso seguro a los datos que comparten las respuestas que se ejecutan concurrentemente.
- El modelo de los recursos de la plataforma: describe la capacidad y las posibilidades de configuración de los recursos de procesamiento (procesadores y redes de comunicación) y las características y la parte de esa capacidad que consumen las tareas de fondo que se utilizan para

su gestión, como los planificadores, los temporizadores, los drivers, etc. y que operan con independencia de la aplicación.

- El modelo reactivo: describe las respuestas a eventos que puede atender la aplicación, especificando para cada una de ellas las secuencias de actividades de que están constituidas, los patrones de generación de los eventos y los plazos temporales en los que deben ser ejecutadas completamente o cada una de sus fases.

El trabajo que se presenta parte de considerar que el sistema bajo estudio está modelado y ha sido analizado. Su objetivo es mostrar amigablemente al operador los diferentes escenarios de los fenómenos que pueden ocurrir en su evolución. Los datos temporales de la evolución del sistema se encuentran en ficheros de trazas, pero su interpretación debe hacerse en base a la información contenida en el modelo de tiempo real del sistema. Por ello, aunque en el trabajo no se construye el modelo, si que se necesita procesarlo para identificar los diferentes elementos que contiene.

En la siguiente tabla se enumeran los conceptos del modelo de tiempo real MAST que se han de gestionar en este trabajo:

Tabla 1. Glosario de términos en los modelos de tiempo real MAST.

Elementos generales	
Modelo de tiempo real *(MAST-Model)*	Conjunto de abstracciones y datos que describen el comportamiento temporal de un sistema de tiempo real que opera en un determinado entorno y bajo unos determinados requisitos temporales.
Situación de Tiempo Real *(Real time Situation)*	Modelo de comportamiento temporal que describe para el sistema un modo de operación resultante de interaccionar con un determinado entorno que establece los eventos que recibe, y para el que se han especificado unos determinados requisitos temporales en sus respuestas. Es el objeto de los análisis de tiempo real.
Recursos	
Recurso de procesamiento *(Processing Resource)*	Abstracción que describe un elemento de la plataforma con capacidad de ejecutar las actividades del sistema.
Procesador *(Processor)*	Tipo especializado de recurso de procesamiento que describe un procesador con capacidad de ejecutar código.
Reloj *(Timer)*	Describe un elemento temporizador asociado a un procesador. Su gestión requiere el uso de la capacidad del procesador en ciertos instantes (cuando se actualiza o cuando se atienden sus eventos).
Redes de comunicación *(Network)*	Tipo especializado de recurso de procesamiento que describe una red de comunicaciones capaz de transferir mensajes.
Planificador *(Scheduler)*	Elemento que se asocia a un recurso de procesamiento y que planifica las actividades que se ejecutan en él. Para operar (cambios de contexto) utiliza durante breves intervalos de tiempo el recurso de procesamiento.
Ente de planificación *(Schedulable resource)*	Elemento lógico que introduce la aplicación a fin de establecer las características de planificación con las que se ejecutan las actividades que se ejecutan en un recurso de procesamiento. Impone una ejecución secuencial de las actividades que tiene asignada. Son los elementos que gestionan los planificadores.
Recurso compartido *(Shared resource)*	Elemento de sincronización con el que se define el protocolo de acceso a un recurso común por parte de actividades que se ejecutan concurrentemente. El recurso debe ser accedido por ellas en régimen de exclusión mutua. En este trabajo los recursos compartidos pertenecen a un único procesador.
Elementos de flujo de control	
Transacción *(Transaction)*	Elemento del modelo que describe una respuesta del sistema. Es un contenedor que agrupa todos los elementos de la respuesta

	organizados por el flujo de control: eventos de activación, eventos internos y actividades.
Operación *(Operation)*	Descripción de una actividad del sistema, ya sea ejecución de un segmento de código o transmisión de un mensaje. Una operación puede ser ejecutada en diferentes respuestas del sistema.
Evento externo *(Workload event)*	Identifica un evento del entorno o del reloj, cuya ocurrencia activa una respuesta en el sistema.
Evento interno *(Internal Event)*	Identifica la transferencia de flujo de control entre dos actividades de una transacción. Su ocurrencia representa la finalización de la actividad de la que es salida y la activación de la actividad de la que es entrada.
Actividad *(Step)*	Ejecución de una operación de la aplicación en el contexto de una transacción.Incluye información de la entidad de planificación en que se lleva a cabo.
Segmento *(Segment)*	Secuencia de actividades que se ejecutan dentro de una misma entidad de planificación, con dependencia de flujo de control directa entre ellas.
Evento final de transacción *(Sink event)*	Evento último de una rama de una transacción. Este evento no activa ninguna nueva actividad de la transacción.
Evento final de segmento *(End segment event)*	Evento último de una secuencia de actividades que constituyen un segmento. Su ocurrencia representa una transferencia de flujo entre entidades de planificación diferentes. Son los únicos eventos que pueden llevar asociados requisitos temporales.
Requerimiento temporal *(Timing requirement)*	Requisito temporal asignado a un evento interno de una transacción. Hay definida una amplia gama de tipos de restricciones diferentes.
Plazo global *(Global deadline)*	Requisito temporal que requiere que un evento de la respuesta ocurra antes de que transcurra el plazo que especifica, relativo al evento externo que activó la respuesta de la que es parte.
Plazo local *(Local deadline)*	Requisito temporal que requiere que un evento de la respuesta ocurra antes de que transcurra el plazo que especifica, relativo al evento que activó el segmento del que es salida.
Retraso *(Delay / Offset)*	Elemento que describe un retraso introducido en una respuesta, bien por un temporizador o por un elemento del entorno. Mecanismo de transferencia de eventos de flujo de control dentro del ámbito de una transacción, y que implica una relación de flujo más compleja que una simple secuencia.

En la figura 2 se muestra a través de un diagrama de clases los elementos del modelo MAST que se gestionan en este trabajo, así como las asociaciones entre ellos y los atributos que se utilizan. Estos elementos no son exactamente los definidos en MAST. Por ejemplo, los Drivers se consideran asignados a los *Processors*, aunque en MAST, existiendo esta asociación implícita, se consideran agregados explícitamente en los *networks*. Asímismo, aunque ciertos elementos (por ejemplo los Segments) no tienen un nombre explícito en el modelo MAST, se le asigna un nombre deducido de los nombres de elementos relacionados, etc.

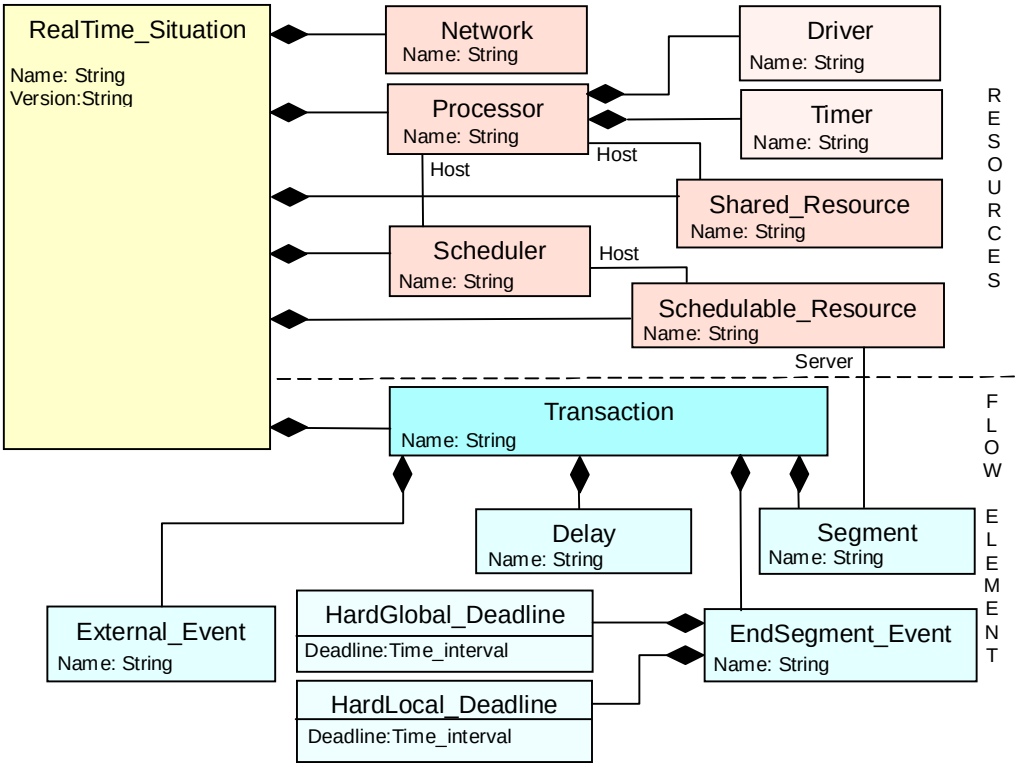

Figura 2: Elementos de modelado de la metodología MAST utilizados en este trabajo.

1.3. Trazas de ejecución

Habitualmente, la generación de un fichero de trazas y su análisis o visualización son procesos que se llevan a cabo en momentos diferentes, por lo que se necesita utilizar una estructura de datos que sirva de soporte de almacenamiento en memoria o en ficheros.

En el entorno MAST se define una estructura de datos que se propone como soporte de la información de trazas. A fin de facilitar la interoperatividad entre herramientas, su formato corresponde a un documento XML, que está formalizado de acuerdo con un W3C-Schema que define su contenido.

A continuación se hará una breve introducción a XML y a su utilización mediante Java.

1.3.1. Breve introducción a XML

XML (eXtensible *Markup Language*) es un metalenguaje utilizado para definir documentos que contienen datos estructurados.

Cada documento XML posee una estructura lógica y otra física. La estructura lógica del documento es una serie de declaraciones, elementos, comentarios, etc. que se indican en el documento mediante marcas explícitas. La estructura física del documento es una serie de unidades llamadas entidades, es decir, indica los datos que contendrá el documento. Las estructuras lógica y física deben anidarse de forma correcta.

Como lenguaje de anotación, las sentencias en XML consisten en una serie de etiquetas (llamadas *elementos*) con una serie de modificadores (llamados *atributos*). Las etiquetas pueden estar anidadas unas

dentro de otras, pero toda etiqueta que se abra se tiene que cerrar, y siempre en el mismo orden. En caso de que un elemento no tenga pareja (por no tener ningún contenido dentro), se le denomina elemento *vacío* y se indica con un / al final. Los elementos se agrupan en *documentos*. *Para mostrar esto, se presenta el siguiente ejemplo:*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<libro>
  <titulo></titulo>
  <capitulo>
    <titulo></titulo>
    <seccion>
      <titulo></titulo>
    </seccion>
  </capitulo>
</libro>
```

Como se puede ver, se permite definir el contenido de los datos de la forma que se desee, mientras sea conforme a la estructura general que XML requiere. Esta “portabilidad” de los datos es la mayor ventaja de XML.

XML es actualmente una *recomendación* completa de la World Wide Web Consortium (W3C) [6]. Esto quiere decir que es una versión final cerrada, y se denomina *recomendación* porque no se impone a nadie seguirla. Podemos ver la especificación 1.0 completa en http://www.w3.org/TR/REC-xml/. Otra cuestión importante relativa a los documentos XML es que puedan ser, aunque no es obligatorio que lo sean, *válidos*. Un documento *válido* es aquel que es conforme con su definición de tipo de documento (DTD) [5], donde se define la gramática y un conjunto de etiquetas para un formato XML específico. Si un documento especifica un DTD y sigue sus reglas se dice que es *válido*. Como una evolución de éstos han nacido los “esquemas XML” [5]. Los esquemas XML son mecanismos que usan la propia sintaxis XML para hacer el trabajo de describir los criterios de validación. En la Interfaz desarrollada en este proyecto se han utilizado esquemas tanto para el fichero de trazas como para un fichero XML de elementos seleccionados que es generado por la aplicación. En el caso de que un documento XML sea conforme a su esquema se dice que es *válido según el esquema*.

Otro concepto interesante de los documentos XML es el *espacio de nombres* [5], que permite escribir documentos XML que usen dos o más conjuntos de etiquetas XML de una forma modular. Con el concepto de espacio de nombres se resuelve la ambigüedad existente entre elementos o atributos que se llamen igual. Para explicar más claramente este concepto, se puede pensar en un supuesto práctico: supóngase que tenemos una hoja XML con libros y otra con discos (con sus correspondientes DTD) y se quiere mezclar ambas páginas. Habrá elementos que no se llamen igual (páginas o tiempo), pero otros que si (titulo, autor). Entonces se usarían ambos DTD y se utilizarían los espacios de nombre para distinguir aquellos elementos en los que no esté claro a que DTD pertenecen. La especificación de *espacio de nombres* define mecanismos para cualificar los nombres y así poder eliminar las ambigüedades. Esto permite escribir programas que usen información de otras fuentes y hagan cosas correctas con ella. Los espacios de nombres se aplican tanto a atributos como a elementos.

Para definir un espacio de nombres al que pertenece un elemento, es necesario añadir un atributo a la definición de elemento, donde el nombre del atributo sea *xmlns* ("xml namespace") y el valor puede ser una cadena cualquiera, aunque por convención suelen ser URLs.

Usando el atributo *xmlns*, cuando se necesite una única referencia a un espacio de nombres, no es mucho trabajo, pero cuando necesitamos hacer la misma referencia varias veces, añadir dicho atributo se convierte en una tarea algo pesada. También hace difícil cambiar el nombre del espacio de nombres, en caso de que se quiera modificar posteriormente.

La alternativa es definir un prefijo de espacio de nombres, que es tan sencillo como especificar *xmlns*, dos puntos (:) y el nombre del prefijo antes del valor del atributo, como se ve a continuación:

```
<sl:slideshow xmlns:sl='http://www.example/slideshow'
      ...>
   ...
</sl:slideshow>
```

Esta definición configura **sl** como un prefijo que puede usarse para cualificar el nombre del elemento actual y cualquier elemento dentro de él. Como el prefijo puede utilizarse en cualquier elemento contenido, tiene más sentido definirlo en el elemento raíz del documento XML.

La r*ecomendación* W3C [6] define un mecanismo para calificar los nombres en función de la utilización y la posición de las etiquetas, para lo cual asocia un espacio de nombres con un prefijo al elemento XML y el resultado es una etiqueta como <*mast_mdl:Transaction*>

1.3.2. Empleo de XML con Java

La programación de este proyecto se ha realizado íntegramente utilizando estas dos tecnologías, Java y XML. Los datos XML son datos portables, puesto que se pueden procesar e interpretar en cualquier plataforma, lo que le convierte en un compañero perfecto para las aplicaciones Java. Java, por definición, es código portable que a su vez necesita datos portables.

Java es el lenguaje de programación ganador para utilizar con XML. La mayoría de los analizadores de sintaxis de XML se escriben en Java y proporciona una colección comprensiva de APIs Java pensada específicamente para construir aplicaciones basadas en XML. De esta manera se puede construir una aplicación completamente portable con la habilidad de representar la entrada y salida de la misma con una capa de datos independiente del sistema.

El entorno de programación que nos ofrece Java para el trabajo con XML es ideal, ya que hoy en día presume de disponer el conjunto de API (*Application Programming Interface*) más completo que permite utilizar XML directamente en el código Java [5].

Con la finalidad de poder analizar y manipular datos XML, Java posee varias APIs. Entre ellas están SAX y DOM, que son las que hemos utilizado en nuestro caso.

DOM (*Document Object Model*) es una especificación definida por el W3C [6]. Su objetivo principal es proporcionar una interfaz estándar que pueda ser utilizada en una amplia variedad de entornos y lenguajes de programación (Java, Javascript, C++, PHP, Python, Ruby, etc.). Define una forma de procesar (*parser*) documentos. DOM está diseñado para proporcionar un medio de manipular los datos XML. Para ello lee el documento XML, lo procesa y genera un árbol jerárquico en memoria del documento o información en XML, almacenando todos los datos como *nodos*, de forma que se facilita el recorrido y manipulación de los mismos. Cada uno de estos nodos puede ser de muchos tipos: documento, elemento, atributo, comentario,... y todos se relacionan entre sí mediante relaciones padre-hijo.

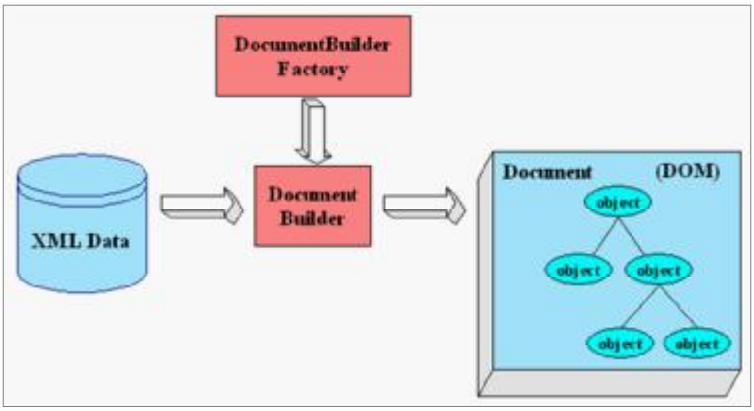

Figura 3. Generación de la estructura en árbol a partir del fichero.

DOM tiene una limitación importante. Al cargar el contenido del documento completamente en memoria, si éste es muy grande realizará un consumo de los recursos muy importante, pudiendo llegar a ralentizar e incluso paralizar la aplicación. Es por eso que no hemos utilizado DOM para el recorrido por el archivo de trazas, ya que éste puede llegar a ocupar un tamaño considerable.

SAX (Simple API for XML) es una especificación que proporciona un marco que le da al analizador medios para manipular y recorrer documentos XML [5]. SAX procesa la información *por eventos, a* diferencia de DOM, no carga en memoria el documento, sino que lo va recorriendo linealmente, de manera que se lanzan eventos a medida que va recorriendo el documento. Estos eventos y las acciones que se realizan en los mismos están definidos por la interfaz org.xml.sax.ContentHandler. Es un *parser* ideal para manipular archivos de gran tamaño, ya que no carga un *árbol en memoria.* Como desventaja frente a DOM, no permite manipular la información una vez procesada. En DOM no existe esta limitación ya que se genera el *árbol jerárquico en memoria* y es posible regresar a modificar nodos.

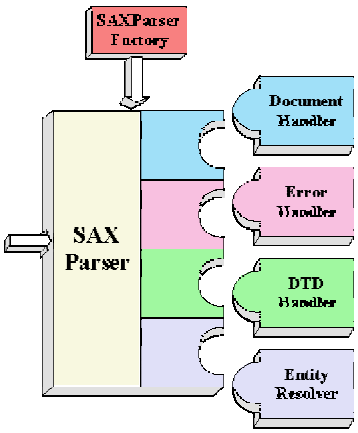

Figura 4. Esquema del analizador de texto

Estas APIs forman el juego de herramientas idóneo que los desarrolladores de Java necesitan para el manejo de XML.

Por tanto, usaremos las librerías SAX para leer en los datos XML y construir el árbol de objetos de datos que constituye el DOM.

1.3.3. Librería SWT

Para la representación gráfica de las trazas se hará uso de la librería SWT (*Standard Widget Toolkit*). Ésta ofrece un conjunto de componentes para construir interfaces gráficas nativas del sistema operativo en donde ejecutemos nuestra aplicación SWT en Java a través del uso de JNI (*Java Native Interface*). Esto quiere decir que con el mismo código visualizaremos en cada sistema operativo las ventanas como si hubieran sido creadas para ese SO específico [7].

Se utilizará la librería SWT ya que es más rápida y consume menos recursos que las librerías previas AWT y Swing que eran ofrecidas por Java. Asimismo, al crearse nativamente dependiendo del sistema operativo mantenemos una uniformidad con el resto de aplicaciones.

Con ayuda de la librería SWT podremos dibujar todos los elementos que necesitamos visualizar.

1.4. Objetivo del trabajo

El objetivo de este trabajo ha sido el desarrollo de una herramienta gráfica e interactiva que muestre y facilite el análisis de la evolución y comportamiento de un sistema de tiempo real que ha sido modelado utilizando la metodología MAST 2.0 y del que se dispone de un escenario de evolución temporal plasmado en un fichero de trazas generado mediante el simulador JSimMast [10]. La herramienta debe visualizar las trazas interpretándolas en base al modelo de tiempo real del que proceden.

En las primeras fases del desarrollo de nuevas estrategias de diseño de sistemas de tiempo real o de nuevas políticas de planificación, es habitual no disponer de herramientas analíticas para ellas y tener que utilizar un simulador que interpreta los modelos y genera trazas que describen la actividad correspondiente al sistema. Asimismo, durante las fases de desarrollo de las herramientas analíticas se utilizan los simuladores para validar los resultados que se obtienen con ellas.

Un fichero de trazas contiene multitud de eventos relativos a la actividad del sistema, cada uno de ellos con información de su origen, de su naturaleza y del instante del tiempo de simulación en el que ha ocurrido. Para que un operador humano pueda interpretarlas, se necesita disponer de una herramienta, tal como la que se ha diseñado en esta tesis, que lea las trazas, las interprete de acuerdo con el modelo del sistema del que proceden, y las presente al operador con diferentes niveles de resolución temporal y organizadas de acuerdo con las opciones que elija.

2. Monitorización de trazas

2.1. Descripción de un fichero de trazas

La generación de trazas y su análisis o visualización son procesos que se llevan a cabo en momentos diferentes, por lo que se necesita utilizar una estructura de datos que sirva de soporte de almacenamiento en memoria o en fichero. Las trazas tienen ciertas características específicas que condicionan las estructuras de datos adecuadas para contenerlas. Éstas son:

- Suelen contener una cantidad masiva de información, por lo que es necesario optimizar el formato de almacenamiento.
- Son generadas por diferentes herramientas de monitorización, simulación, generadores de alarmas, etc., por lo que el criterio de almacenamiento debe ser suficientemente flexible.
- Los procesos de análisis y visualización de trazas suelen ser muy parecidos, por lo que es conveniente que una misma herramienta se pueda aplicar a trazas procedentes de diferentes fuentes.

En el entorno MAST se define un modelo de datos que se propone como soporte de la información de trazas. A fin de facilitar la interoperatividad entre herramientas, su formato corresponde a un documento XML formalizado de acuerdo con una plantilla W3C-Schema que, más allá de las normas sintácticas impuestas por el propio lenguaje XML, define la estructura y restricciones de contenido de una forma muy precisa. Tal *schema* denomina Mast_Trace.xsd y su contenido se muestra en el apéndice 7.1. Así, la información de trazas contenida en un documento XML basado en tal plantilla se organiza en cuatro bloques, que se exponen a continuación a través de un ejemplo ilustrativo.

1. Cabecera: contiene diferentes datos globales del documento, relativos a su naturaleza y origen de la información en él contenida.

```
<?xml version="1.0" encoding="UTF-8"?>
<?mast file Type="XML-Mast-Trace-File" version="1.1"?>
<mast_trace:TRACE_FILE
       xmlns:mast_trace="http://mast.unican.es/xmlmast/trace"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://mast.unican.es/xmlmast/trace
../JSimMast_V_Schemas/Mast_Trace_V.xsd"
       Model_Name="A_Simple_1_0"
       Model_Date="2010-03-29T12:00:00"
       Generation_Tool="JSimMast_V0"
       Generation_Profile="TRACES"
       Generation_Date="2011-04-19T16:34:55"
       Start_Time="0"
       End_Time="0000">
```

2. Lista de tipos de mensaje: se trata de la relación de tipos de evento (o hito), de entre aquellos que componen la evolución temporal del sistema, sobre cuya ocurrencia se reporta en la información de trazas. Engloba un conjunto de elementos <Msg_Type>, cada uno especificando un nombre para el tipo de hito y un identificador entero con el que se le designa en la lista de trazas (bloque 4).

```
<mast_trace:Msg_Type_List>
       <mast_trace:Msg_Type Mid="0" Type="RegularProcessor/FREE"/>
       <mast_trace:Msg_Type Mid="1" Type="RegularProcessor/ATTENDING_TIMER"/>
       <mast_trace:Msg_Type Mid="2" Type="RegularProcessor/ATTENDING_INTERRUPT"/>
       <mast_trace:Msg_Type Mid="3" Type="RegularProcessor/ATTENDING_SCHEDULER"/>
       <mast_trace:Msg_Type Mid="4" Type="RegularProcessor/ATTENDING_CONTEXTSWITCH"/>
       ...
       <mast_trace:Msg_Type Mid="20" Type="Global deadline met"/>
</mast_trace:Msg_Type_List>
```

3. Lista de fuentes de trazas: se trata de la relación de elementos del sistema (objetos del modelo MAST)

que son potenciales fuentes de generación de aquellos tipos de eventos (bloque 2) sobre cuya ocurrencia se reporta en las trazas. Engloba un conjunto de elementos <Src>, cada uno especificando el nombre de la fuente, su tipo y un identificador entero que se utiliza como clave para designarlo en la lista de trazas (bloque 4).

```
<mast_trace:Src_List>
       <mast_trace:Src Sid="0" Name="Proc_1" Type="ComputingResource"/>
       <mast_trace:Src Sid="1" Name="Shared_Resource_1" Type="MutualExclusionResource"/>
       <mast_trace:Src Sid="2" Name="Trans_1/Trigger_1" Type="WorkloadEvent"/>
       <mast_trace:Src Sid="3" Name="Trans_1/End_1-segment" Type="Segment"/>
</mast_trace:Src_List>
```

4. Lista de mensajes de ocurrencia de eventos: se trata de la relación de eventos ocurridos durante la evolución del sistema y que son de alguno de los tipos descritos en el bloque 1. Engloba un conjunto de muchos miles (o cientos de miles) de mensajes en forma de elementos <Msg> que constituye la información de traza propiamente dicha. Por cada mensaje se proporciona el instante de tiempo en que se ha generado la traza, la fuente que ha generado el evento – mediante la clave definida en el bloque 3 – y el tipo de mensaje - mediante la clave definida en el bloque 2 –.

```
<mast_trace:Msg_List>
       <mast_trace:Msg T="7500000" S="2" M="16"/>
       <mast_trace:Msg T="7500000" S="3" M="15"/>
       <mast_trace:Msg T="7500000" S="0" M="4"/>
       ...
</mast_trace:Msg_List>
```

Con este formato, los ficheros de trazas reúnen las ventajas de limitar los tipos de eventos o mensajes que puede contener, de permitir cualquier tipo de elemento como origen de los mismos y a su vez, reducir la dimensión de los ficheros a un tamaño manejable.

2.2. Presentación de las trazas: organización, reactividad e iconos

En esta sección se muestran las características de la interfaz gráfica de usuario (GUI) del monitor de trazas, describiéndose:

- La jerarquía de ventanas.
- Los elementos que constituyen cada ventana y su reactividad.
- Los símbolos con los que se representa cada evento que se muestra.

2.2.1 Jerarquía de ventanas

La GUI del monitor de trazas está basada en una ventana principal que se abre al arrancar la aplicación así como un conjunto de ventanas emergentes cuya apertura se producirá a requerimiento del operador mediante interacción sobre la ventana principal, por lo que la reactividad de ésta deberá contemplar y manejar la gestión del resto de ventanas definidas en la GUI. A continuación se presenta la clasificación de los distintos tipos de ventanas en juego, primero ofreciendo una breve descripción de cada uno de ellos junto con el nombre con que se designan en esta memoria y seguidamente de forma visual mediante la figura 5. Posteriormente, en las siguientes subsecciones se detallan más en profundidad las características de cada tipo de ventana.

- *Main_Win*: ventana base de la aplicación, que se abre al arrancar la herramienta.
- *All_E2EF_Win*: ventana emergente con la gráfica de la actividad global del sistema.
- *Aux_All_E2EF_Win*: ventana emergente con información auxiliar (leyenda) que complementa a la

ventana anterior.

- *Detailed_E2EF_Win*: ventana emergente con la gráfica temporal detallada de una transacción concreta del sistema.
- *Aux_Datailed_E2EF_Win*: ventana emergente con la leyenda que complementa a la ventana anterior.
- *All_Resource_Win*: ventana emergente con la gráfica temporal del uso del conjunto de recursos del sistema.
- *Aux_All_Resource_Win*: ventana emergente con la leyenda que complementa a la ventana anterior.
- *Detailed_Resource_Win*: ventana emergente con la gráfica temporal del uso de un recurso concreto de procesamiento del sistema.
- *Aux_Detailed_Resource_Win*: ventana emergente con la leyenda que complementa a la ventana anterior.

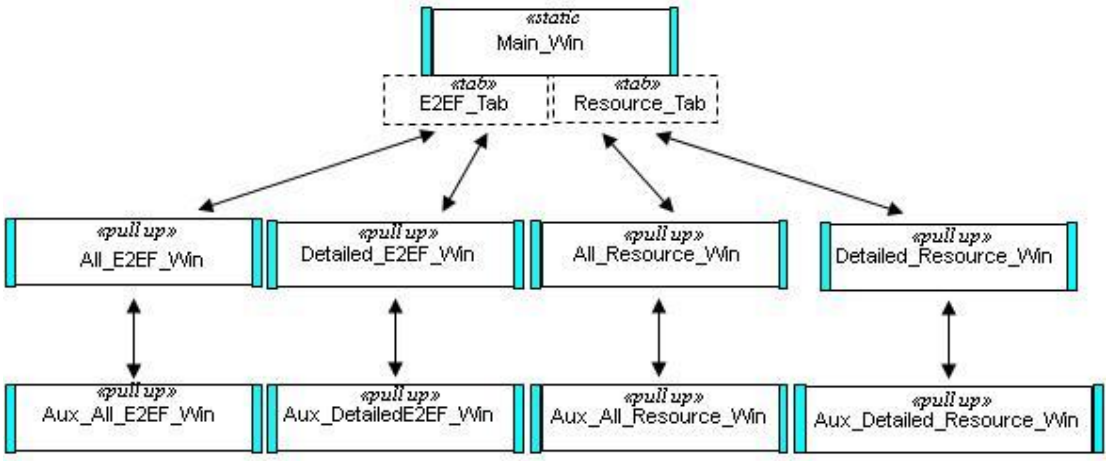

Figura 5. Jerarquía de ventana de interacción con el usuario.

2.2.2 Main_Win

Es la ventana inicial de la herramienta. Permite especificar el fichero de trazas sobre el que opera, el tipo de monitorización que va a realizar y los elementos que se van a mostrar.

El campo *TracesFile* permite al operador introducir la ruta del fichero de trazas cuya información se va a mostrar. Cuando se introduce la información y se acepta, se lee y analiza el fichero, y en el caso de que sea correcto se muestra en el campo *Model* el nombre del modelo al que corresponden las trazas, habilitándose el resto de los elementos de la ventana.

La ventana contiene dos pestañas ("*End To End Flow*" y "*Resources*") que representan las posibilidades de tipos de información que puede mostrar la herramienta:

Pestaña de selección de transacciones: Contiene la lista de las transacciones del modelo. A partir de la selección de una de ellas, se describe de forma estructurada los elementos que están relacionados con ella y se accede a las ventanas emergentes que describen su evolución temporal.

Permite seleccionar las transacciones que van a ser mostradas y acceder a su repesentación gráfica. En la figura 6 se muestra la ventana base con la pestaña de selección de transiciones:

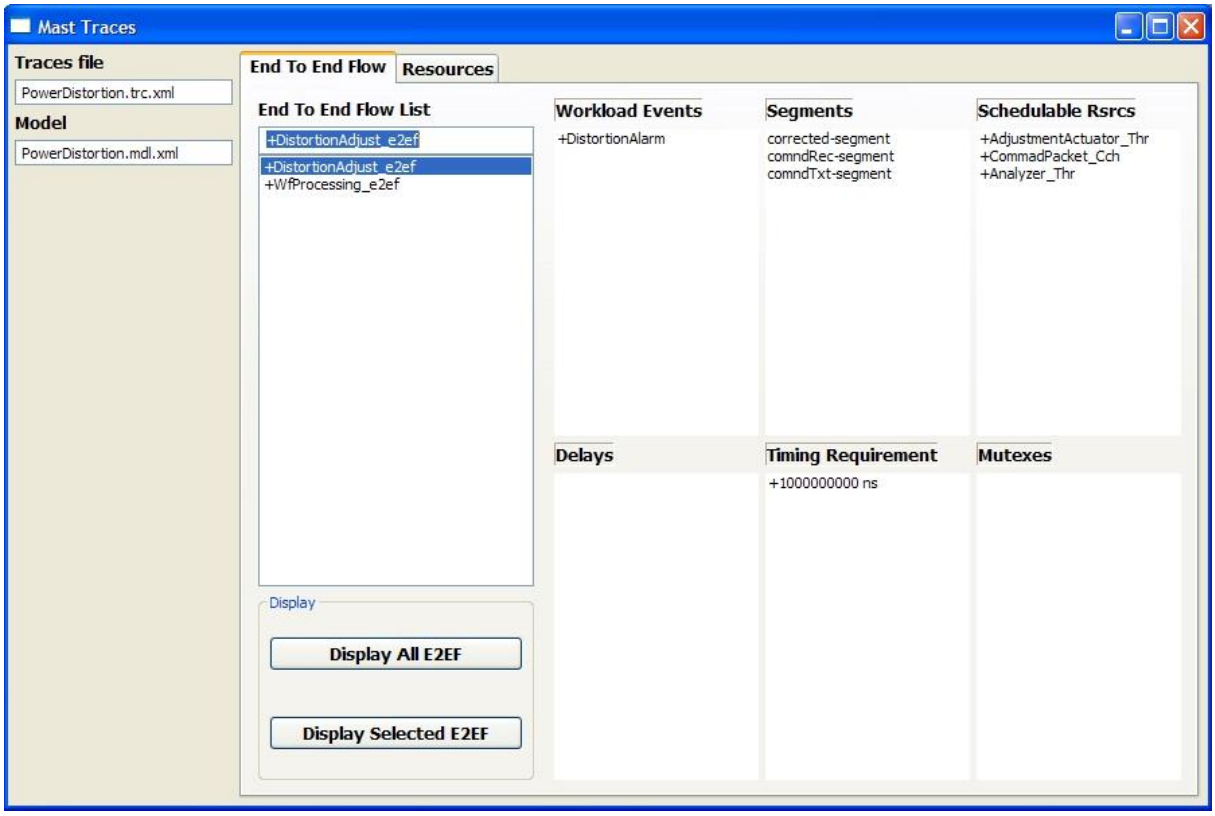

Figura 6. Ventana inicial de la interfaz con la pestaña de las transacciones seleccionada

Los botones del grupo *Display* permiten abrir la gráfica temporal de las transacciones en sus dos opciones: la transacción seleccionada completa o el conjunto de todas las transacciones de forma más simplificada.

Una vez seleccionada una de las transacciones de la lista se habilitará el botón *Display Selected E2EF*, dando la posibilidad de mostrarnos únicamente la evolución temporal de la transacción seleccionada.

Reactividad:

La reactividad asociada a esta ventana será la siguiente:

- Si se pulsa con el botón izquierdo del ratón sobre el nombre de cualquier elemento en una de las listas, se habilita o se deshabilita. Un elemento deshabilitado aparece señalado en la lista con un prefijo "–", mientras que un elemento habilitado aparece con un prefijo "+".
- Botón *Display All E2EF*: aparecen dos ventanas emergentes, una de tipo *All_E2EF_Win* y otra de tipo *Aux_All_E2EF_Win*.
- Botón *Display Selected E2EF*: aparecen dos ventanas emergentes, una de tipo *Detailed_All_E2EF_Win* y otra de tipo *Aux_Detailed_All_E2EF_Win*.

Pestaña de selección de recursos: Contiene la lista de los recursos de procesamiento del modelo. A partir de la selección de uno de ellos, se muestran de forma estructurada los elementos que están relacionados con él. A través de ella se selecciona un recurso de procesamiento y se accede a las ventanas emergentes que describen su evolución temporal. En la figura 7 se muestran los elementos de que se compone.

Reactividad:

La reactividad de esta ventana es similar a la descrita para la pestaña de transacciones:

- Si se pulsa con el botón izquierdo del ratón sobre el nombre de cualquier elemento se habilita o se deshabilita. Un elemento deshabilitado aparece señalado en la lista con un prefijo "–" mientras que un elemento habilitado aparece con un prefijo "+".

- Botón *Display All Resources*: aparecen dos ventanas emergentes, una de tipo *All_Resource_Win* y otra de tipo *Aux_All_Resource_Win*.
- Botón *Display Selected Resources*: aparecen dos ventanas emergentes, una de tipo *Detailed_Resources_Win* y otra de tipo *Aux_Detailed_Resources_Win*.

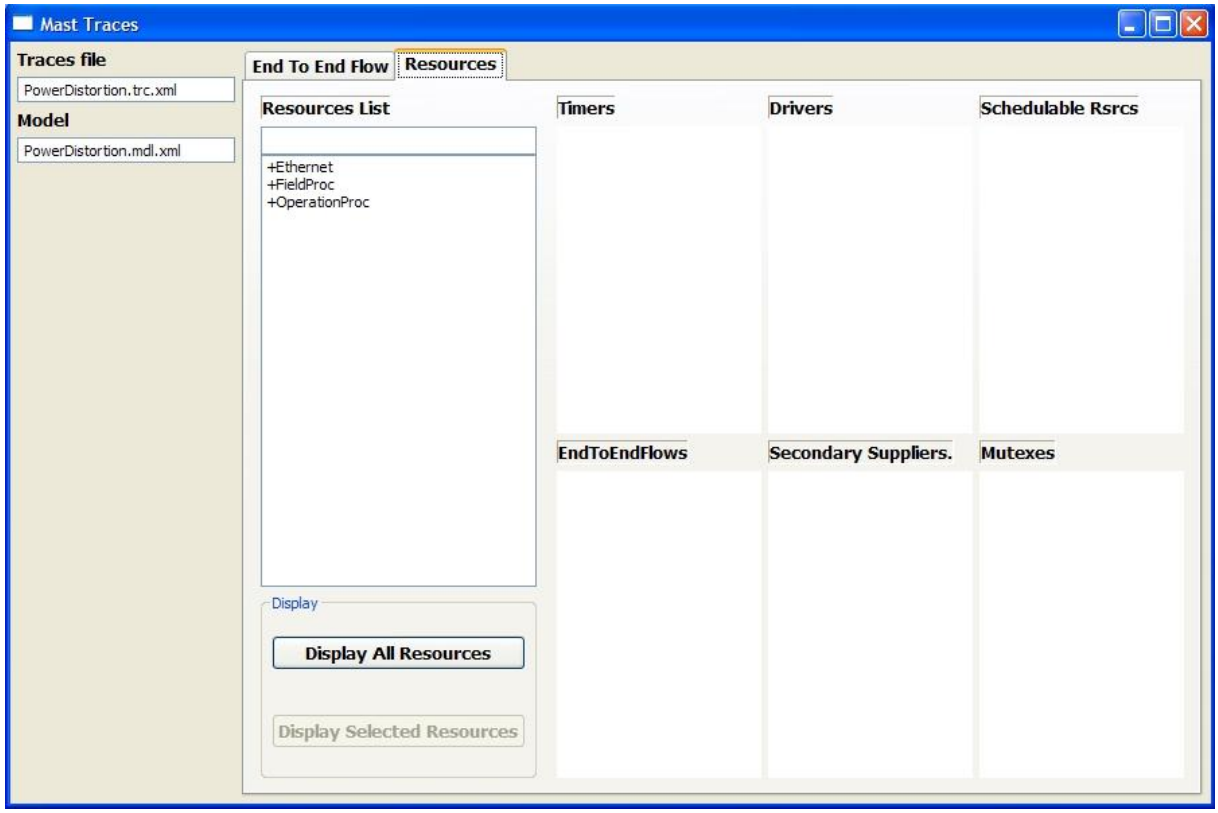

Figura 7. Panel inicial de la interfaz con la pestaña de los recursos seleccionada.

2.2.3 All_E2EF_Win

Permite visualizar la evolución temporal del sistema a través de la representación temporal de los estados de todas sus transacciones. Se incluyen todas las transacciones, *threads*, segmentos, eventos (externos y regulares) y *deadlines* (locales y globales, cumplidos y perdidos) que no hayan sido deshabilitados, la información temporal de la transacción y los botones *X2*, */2* e *Ini*, Se muestra en la figura 8.

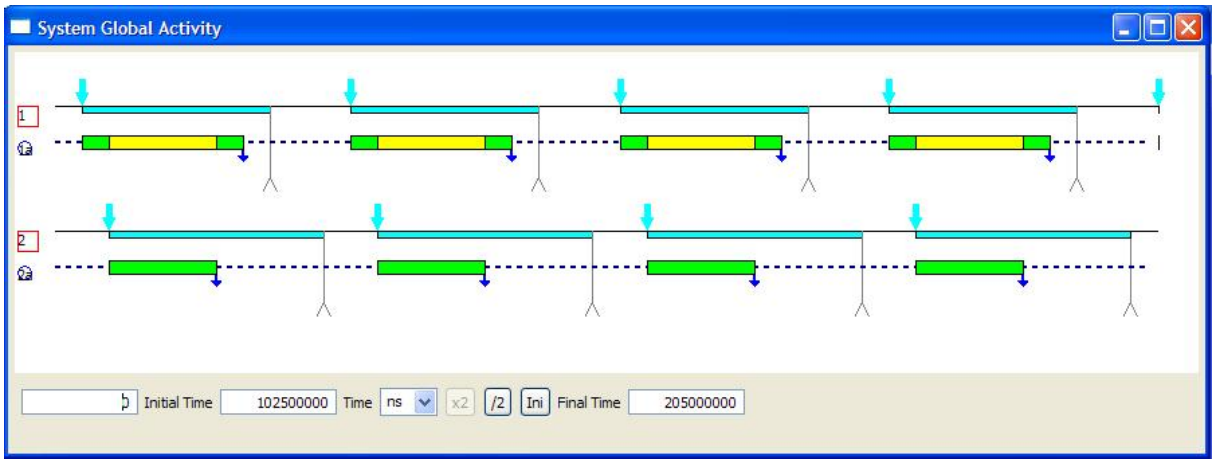

Figura 8. Ejemplo de la visualización gráfica de la evolución temporal de dos transacciones.

Reactividad:

- Botón *Ini*: se establece como tiempo de visualización la totalidad del tiempo de la traza (*Initial time* = 0, *Final time* = duración de la traza y *time*= punto medio).
- Botón *X2*: se duplica el tiempo de visualización, manteniéndose el tiempo time. (Time no se afecta, nuevo *Initial time*= *Time* – 2* (*Time*-previo *Initial Time*), nuevo *Final Time*= *Time* + 2*(previo *Final Time* - *Time*)). En ningún caso Nuevo *Initial Time*< 0 y nuevo *Final Time* > Tiempo de la traza.
- Botón */2*: se reduce el tiempo a la mitad, manteniéndose el Time constante. En cualquier caso la escala debe tener al menos un *ns* por pixel.
- Selector de unidades de tiempo: Permite seleccionar *s*, *ms*, *us* o *ns*. Sólo afecta a los valores numéricos en las cajas de texto y no a las gráficas.
- Pulsar en cualquier punto de las gráficas con el botón izquierdo supone poner el tiempo al valor de la columna del pixel pulsado. Afecta a la caja de texto Time y a la marca en la escala de tiempos. No afecta a las gráficas.
- Pulsar en cualquier punto de las gráficas con el botón derecho visualiza la información correspondiente al elemento pulsado.
- Pulsando el botón X se hace desaparecer la ventana gráfica, la ventana auxiliar y la ventana de información de los elementos si existiera.

2.2.4 Aux_All_E2EF_Win

Esta ventana se muestra en la figura 9, y constituye la leyenda de la representación temporal de la ventana anterior All_E2EF_Win. Por un lado tendremos una lista de las transacciones que estamos dibujando mientras que en el otro el significado de cada uno de los objetos de visualización.

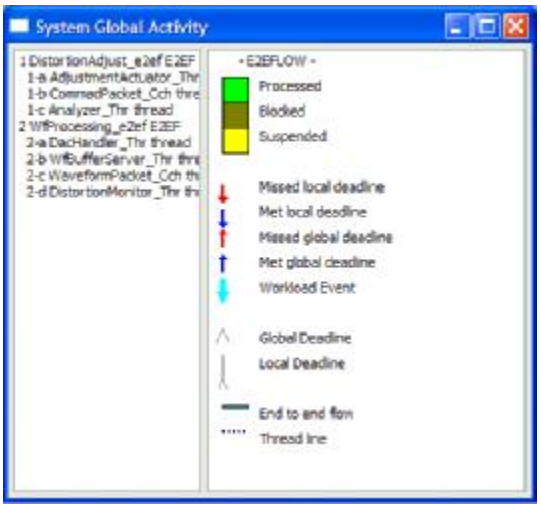

Figura 9. Ventana Aux_All_E2EF_WIN con las leyendas de la ventana

2.2.5 Detailed_E2EF_Win

Se muestra en la figura 10, y sirve para visualizar la evolución temporal de una transacción. Se incluye la misma información gráfica que en una ventana de All_E2EF_Win para la transacción seleccionada, y así mismo representa todos los recursos de procesamiento que ejecutan actividades de ellas y sus mútexes agregados que no hayan sido inhibidos.

Reactividad:

La reactividad de esta ventana será la misma que en la ventana emergente de tipo All_E2EF_Win.

Figura 10. Detailed_E2EF_Win: visualización de una transacción junto a los recursos

2.2.6 Aux_Detailed_E2EF_Win

En la figura 11 se muestra la ventana Aux_Detailed_E2EF_Win, que contiene la información textual que explica la naturaleza de los símbolos que aparecen en las ventanas del tipo Detailed_E2EF_Win.

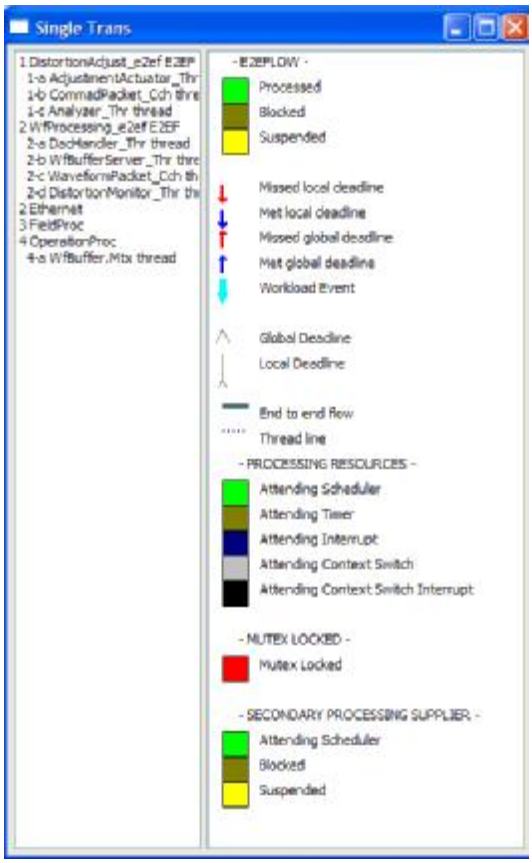

Figura 11. Aux_Detailed_E2EF_WIN con las leyendas de la ventana Detailed_E2EF_WIN

2.2.7 All_Resources_Win

En la figura 12 se muestra la ventana Ventana All_Resources_Win. Es una ventana emergente que describe la evolución del estado de todos los recursos de procesamiento y mútexes asociados a cada uno que no han sido deshabilitados.

La reactividad de esta ventana es similar a la de las otras ventanas con gráficas temporales.

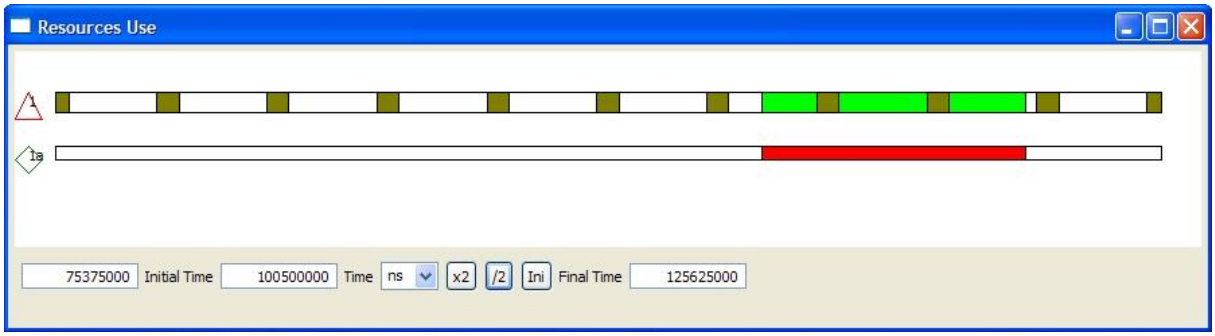

Figura 12. All_Resources_Win con la visualización de un procesador y un mutex

2.2.8 Aux_All_Resources_Win

En la figura 13 se muestra la ventana Aux_All_Resources_Win. Esta ventana muestra, al igual que las anteriores ventanas auxiliares, la información textual que explica la naturaleza de los símbolos.

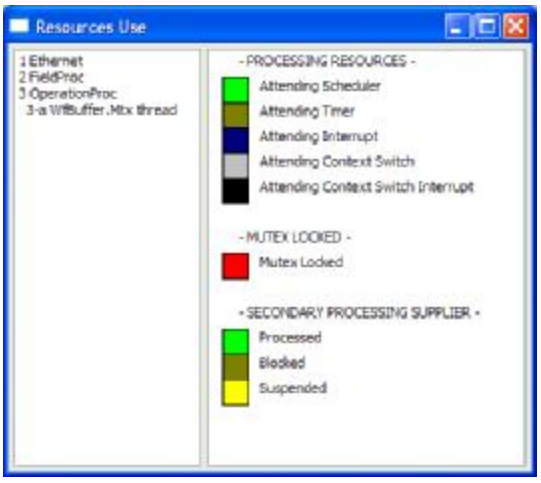

Figura 13. Aux_All_Resource_Win con la leyenda de la ventana All_Resources_WIN

2.2.9 Detailed_Resource_Win

En la figura 14 se muestra la ventana Detailed_Resource_Win. Describe los estados por los que evoluciona un recurso de procesamiento y los mútexes que se encuentran asociados. Los estados se pueden visualizar de forma correlacionada a la evolución de una determinada transacción que se elige como referencia.

Reactividad:

La reactividad de esta ventana es similar a la de las otras ventanas con gráficas temporales.

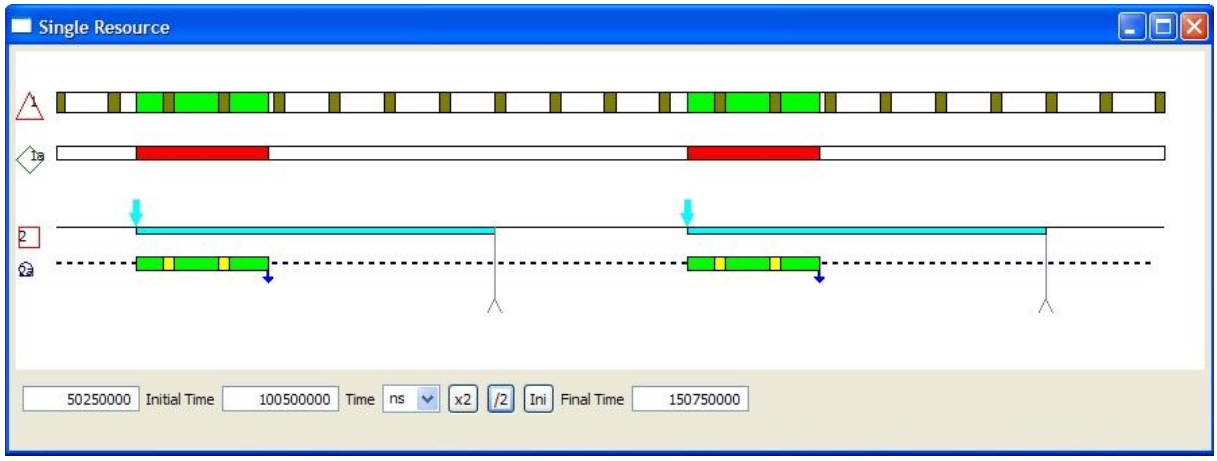

Figura 14. Detailed_Resources_Win con la visualización de un procesador un mutex y una transacción

2.3. Gestión del tiempo

En la parte inferior de todas las ventanas gráficas hay disponible información relativa al intervalo de tiempo sobre el que se están visualizando las evoluciones temporales. En la figura 15 se muestra un ejemplo de esta sección.

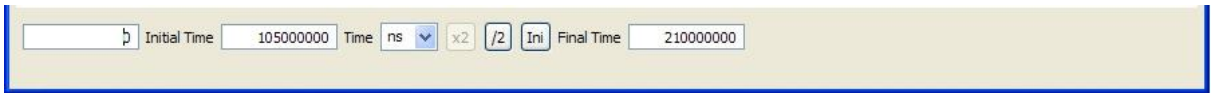

Figura 15. Panel de gestión del tiempo de cualquier ventana gráfica.

Esta información es: *Initial Time*, *Time*, *Final Time*

Junto a esta información también hay una serie de botones que completan la interfaz dotándola de los mecanismos necesarios para un estudio más profundo de las trazas:

- Botón *Ini*: Se establece como tiempo de visualización la totalidad del tiempo de la traza (*Initial time*=0, *Final time*= duración de la traza y *time*= punto medio). Nos permite volver al punto inicial en cualquier momento.
- Botón *X2*: Se duplica el tiempo de visualización, manteniéndose el tiempo *time*. (*Time* no se afecta, nuevo *Initial time*= *Time* – 2* (*Time*-previo *Initial Time*), nuevo *Final Time*= *Time* + 2*(previo *Final Time- Time*)). En ningún caso Nuevo *Initial Time*< 0 y nuevo *Final Time* > Tiempo de la traza.
- Botón *%2*: Se reduce el tiempo a la mitad manteniéndose el Time constante. En cualquier caso la escala debe tener al menos un *ns* por pixel.
- Selector de Unidades de tiempo: Permite seleccionar *s, ms*, *us* o *ns*. Sólo afecta a los valores numéricos en las cajas de texto y no a las gráficas.
- Pulsar en cualquier punto de las gráficas con el botón izquierdo supone poner el tiempo al valor de la columna del pixel pulsado. Afecta a la caja de texto *Time* y a la marca en la escala de tiempos. No afecta a las gráficas.
- Al movernos con el ratón dentro del área de diseño la aplicación nos mostrará en que instante de tiempo nos encontramos en cada momento.

2.4. Información reactiva de ayuda

Como se muestra en la figura 16, al pulsar sobre un elemento gráfico con el botón derecho del ratón en cualquiera de las ventanas gráficas se abrirá un cuadro de diálogo con información relevante de dicho elemento gráfico.

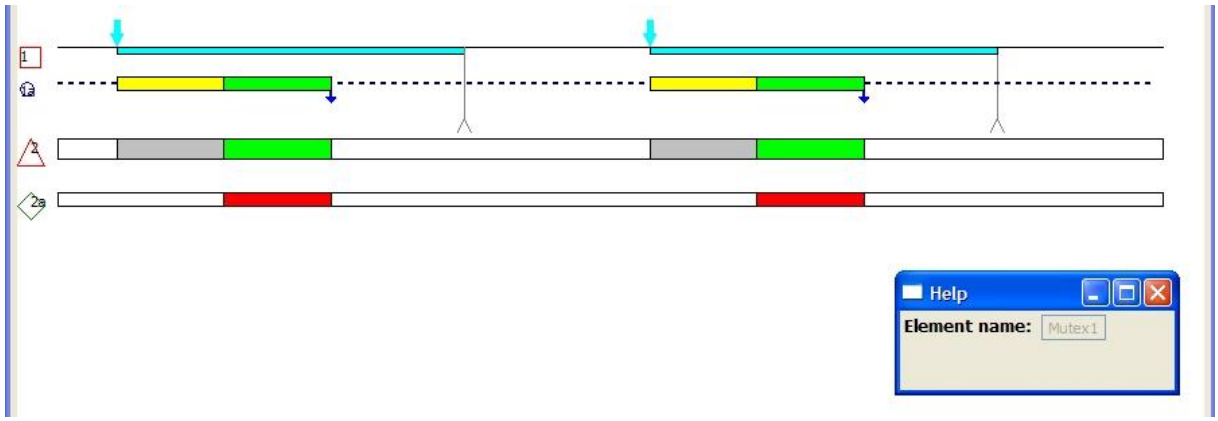

Figura 16. Ejemplo de visualización de la información asociada de un elemento

Este punto no está completado para todos los elementos gráficos, y quedará pendiente como una futura mejora de la aplicación. Se han dado los primeros pasos y se han indicado las pautas a seguir para implementarlo. Para mayor usabilidad, se propone mostrar toda la información relevante asociada al elemento seleccionado. A día de hoy, únicamente se muestra el nombre del elemento seleccionado.

3. Diseño del monitor de trazas

3.1. Criterios de diseño

El monitor de trazas es una aplicación compleja que requiere gestionar dos informaciones de entrada:

a) El modelo MAST del sistema de tiempo real, el cual debe ser analizado para interpretar el origen y las relaciones entre los elementos del sistema al que corresponden las trazas que se están tratando de visualizar.

b) El propio fichero de trazas, que contiene los detalles sobre los cambios de estado y los eventos que se han producido en el intervalo de tiempo al que corresponden las trazas.

Asimismo, la información de salida es de tipo gráfico, lo que significa que debe mantenerse información de todos los elementos gráficos que se encuentran en la pantalla, los cuales pueden ser seleccionados por el operador y ha de existir la capacidad de proporcionar información sobre su origen.

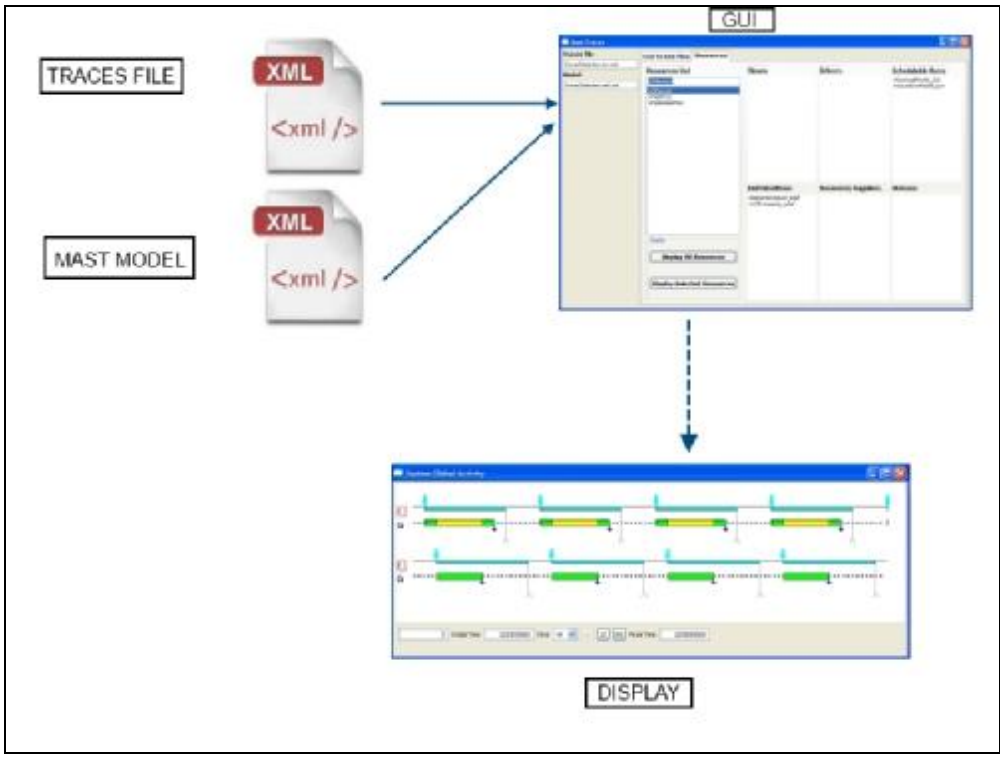

Figura 17. Información de entrada y salida de la aplicación JMastTraces.

Las características de la información de entrada y salida que debe ser gestionada condicionan totalmente el diseño de la aplicación:

- Modelo del sistema. Se trata, en general, de una información de volumen reducido pero de estructura compleja, ya que sus elementos están fuertemente interrelacionados. En el proceso de lanzamiento de la herramienta, se carga en memoria toda esta información relativa al modelo, la cual se analiza para extraer la parte de información que es útil para la visualización y se reorganiza de acuerdo con los requisitos de datos que se van a necesitar en los procesos de visualización y de respuesta a los requerimientos del operador. Tiene una gran importancia en el diseño, ya que los datos de la aplicación se organizan en base a ella.

- Trazas. Es una información con un volumen masivo, por lo que no puede ser cargada toda ella en memoria, sino que se tiene almacenada en fichero y es leída cada vez que se debe obtener parte de ella. De esta manera, está temporalmente en memoria sólo aquella que se necesita para generar la información gráfica que en cada momento requiere el operador. Dado que los cambios sólo se producen en respuesta a requerimientos del operador, sólo hay que ser eficiente en su gestión para que la respuesta de la herramienta sea ágil.
- Información gráfica que se muestra en el monitor. Se trata de información también muy masiva y que por tanto no puede ser almacenada en memoria de forma completa. En cada interacción del operador, la parte más masiva se genera a partir de la información de las trazas, pero sólo temporalmente y con el objetivo de visualizarla como píxeles de la imagen del monitor. No obstante, hay que guardar en memoria una parte reducida de ella, con el objetivo de responder a los requerimientos del operador y poder establecer relación entre la información gráfica por la que se guía el operador y la información del modelo con la que se interpreta.

3.2. Arquitectura de la aplicación

La aplicación TracesDisplayer presentada en esta tesis de máster se ha diseñado siguiendo el paradigma de orientación a objetos (OO). Las diferentes clases que constituyen su arquitectura se encuentran agrupadas en los siguientes paquetes:

- *core*: contiene las clases principales de la aplicación, esto es las clases que implementan las diversas ventanas de interacción con el usuario así como una clase repositorio de objetos de visualización entorno a la cual pivota todo el resto de la aplicación.
- *visualization*: contiene el conjunto de clases, análogas a las de un subconjunto de las clases del modelo MAST, cuyas instancias son los elementos visualizables en que se convierten los elementos del modelo de entrada (modelo del sistema de tiempo real).
- *graphicItems*: contiene una librería de clases que representan figuras geométricas, con capacidad de ser dibujadas y componer, sobre la GUI de la herramienta, la representación visual que se ofrece al usuario de la evolución temporal del sistema.

3.2.1. Paquete *core*

Este paquete contiene tres clases que representan los tres tipos de ventanas que conforman la GUI de la aplicación, tal y como ha sido expuesto en la subsección 2.2, junto con la clase nuclear de la herramienta. Las tres primeras son TD_ControlGUI, TD_TimeGUI y TD_LegendGUI, mientras que la última es la clase VisRepository.

- TD_ControlGUI se corresponde con la ventana principal de la aplicación, la cual se abre en el arranque, por lo que es la que posee el punto de entrada – método *main()* en la implementación Java –.
- TD_TimeGUI se corresponde con cualquiera de las cuatro variantes de ventanas que se abren a requerimiento del operador cuando solicita que se muestren las gráficas de evolución temporal de la actividad global del sistema, de una transacción concreta, del uso de todos los recursos o de uno en concreto, respectivamente. La variable que denota qué variante de ese segundo tipo de ventana se va a instanciar – a efectos de la representación gráfica en el lienzo de dibujo – viene representada por el atributo correspondiente al modo, del tipo enumerado ModeType. En una instancia TD_TimeGUI siempre se dispone de la información relativa a los extremos del intervalo temporal que se está visualizando.
- TD_LegendGUI se corresponde con cualquiera de las cuatro variantes de ventanas auxiliares que juegan el papel de leyenda. El modo de instanciación a efectos de la representación gráfica que albergue su lienzo viene determinado por el modo en que se haya instanciado la ventana de tipo

TD_TimeGUI correspondiente, a la cual se encuentra subordinada.

- VisRepository constituye el espacio en que se crea el modelo de visualización a partir del modelo MAST de un sistema de tiempo real. Es una clase compuesta en su totalidad por miembros estáticos. Los campos son básicamente contenedores de objetos de visualización que se rellenan cuando, tras arrancar la aplicación, se lee el modelo de entrada MAST del sistema de tiempo real. Aunque en primer nivel sólo haya contenedores dedicados a los recursos de procesamiento, las transacciones y los recursos compartidos, todos los objetos de visualización se encuentran almacenados aquí, bien sea directamente o mediante relaciones de composición descritas por las tres clases de contención superior. Durante una sesión de funcionamiento de la herramienta, el acceso a los objetos de visualización contenidos en el repositorio es constante. En cuanto a los métodos, la clase proporciona diversos servicios estáticos como por ejemplo el análisis del modelo de trazas que se quiere representar.

Las relaciones entre las clases de este paquete se muestran en la figura 18, mientras que la clase VisRepository se muestra en mayor detalle en la figura 19..

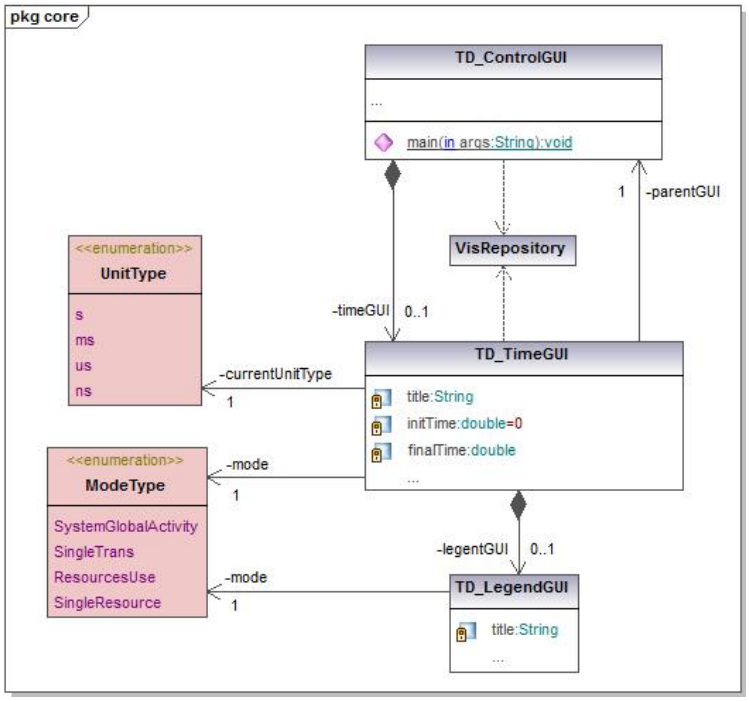

Figura 18. Relaciones entre las clases del paquete core.

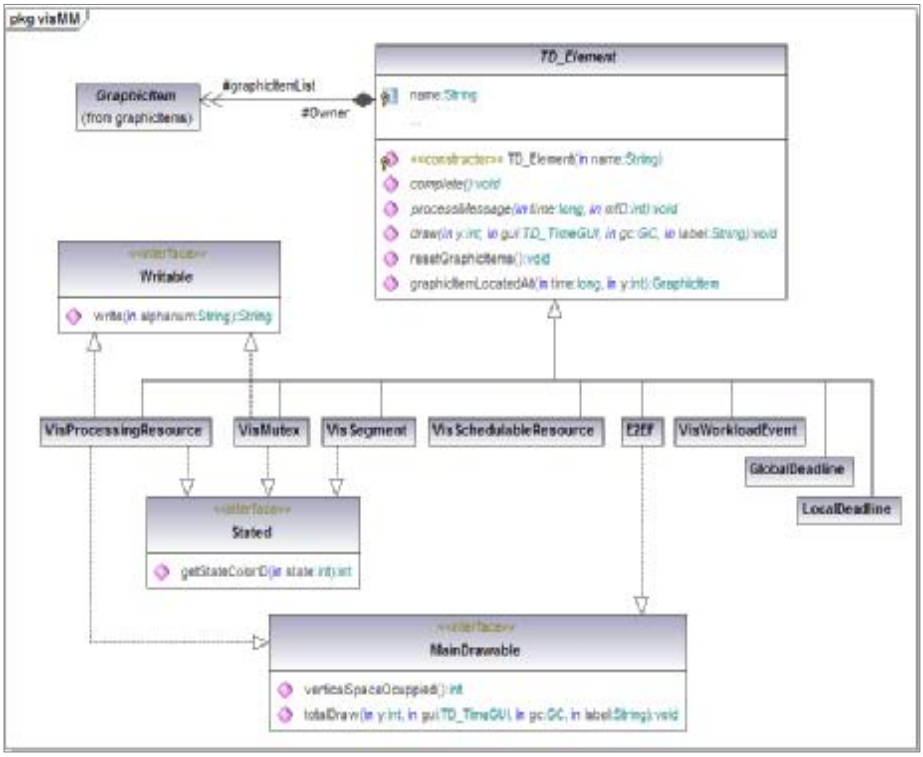

Figura 20. Jerarquía de clases del paquete visualization.

A continuación se exponen más en detalle las clases de este paquete.

3.2.2.1. Clase *TD_Element*

Esta clase abstracta es la clase raíz de la que se derivan todas las subclases correspondientes a la visualización.

- Campos
  - Nombre: name
  - Lista de elementos gráficos: graphicItemList
- Métodos
  - complete()
  - processMessage()
  - draw()
  - reset

3.2.2.2. Clase *VisProcessingResource*

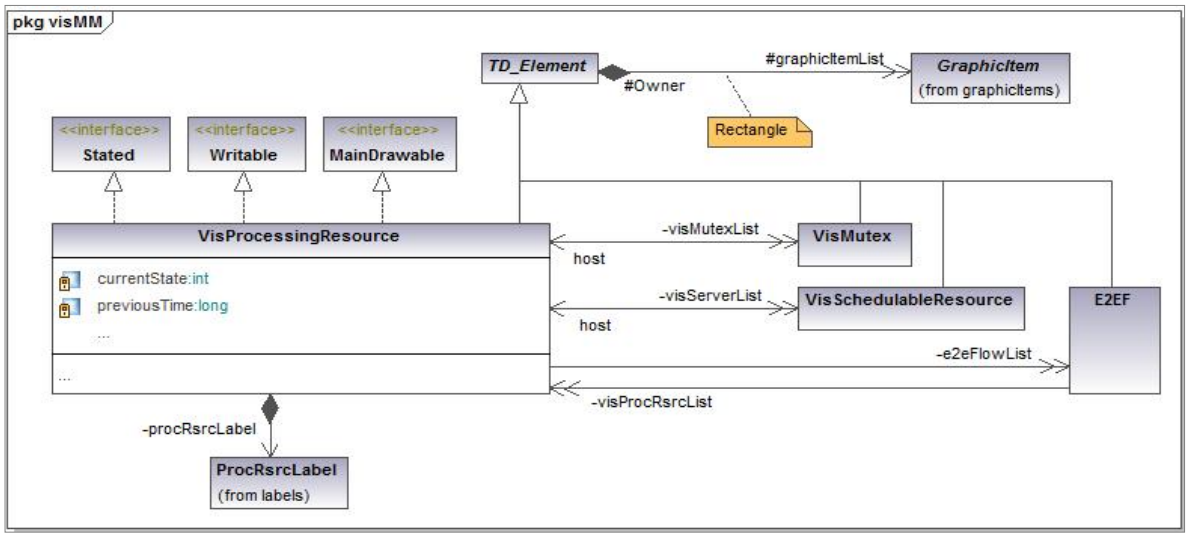

Figura 21. VisProcessingResource

Esta subclase concreta de TD_Element es la clase del meta-modelo de visualización homóloga a la clase Processing_Resource de MAST.

A los atributos heredados de TD_Element, añade las siguientes referencias a otros elementos de visualización:

- visServerList: recursos de planificación asociados a un recurso de procesamiento, esto es, aquellos cuyo host sea ese recurso de procesamiento.
- e2eFlowList: transacciones asociadas a un recurso de procesamiento, esto es, aquellas que poseen al menos una actividad cuyo ejecutor sea un recurso de planificación asociado a ese recurso de procesamiento.
- visMutexList**:** mútexes asociados a un recurso de procesamiento, esto es, aquellos tomados por operaciones ejecutadas por recursos de planificación asociados a ese recurso de procesamiento.

Así como los siguientes atributos:

- currentState: identificador del estado actual
- previousTime: instante en que sucedió el último cambio de estado.

La clase implementa tanto los métodos abstractos definidos en su superclase TD_Element como los definidos en las interfaces que implementa. La implementación de los primeros merece ser detallada brevemente a continuación:

- **public void** complete()
  Establece las siguientes referencias:
  - e2eFlowList
  - visServerList
  - visMutexList
- **public void** processMessage(**long** time, **int** mID)
  Añade a la lista de elementos gráficos uno que represente al recurso de procesamiento en su **estado previo** a la llegada del mensaje, desde el instante en que sucedió el último cambio de estado hasta el instante en que llega el mensaje.
- **public void** draw(**int** y, TD_TimeGUI gui, GC gc)
  Dibuja en un *canvas* de gui el recurso de procesamiento como una banda (compuesta de una sucesión de rectángulos) a partir de la altura y pasada como parámetro más otras bandas por debajo correspondientes a sus mútexes asociados.

3.2.2.3. Clase *VisMutex*

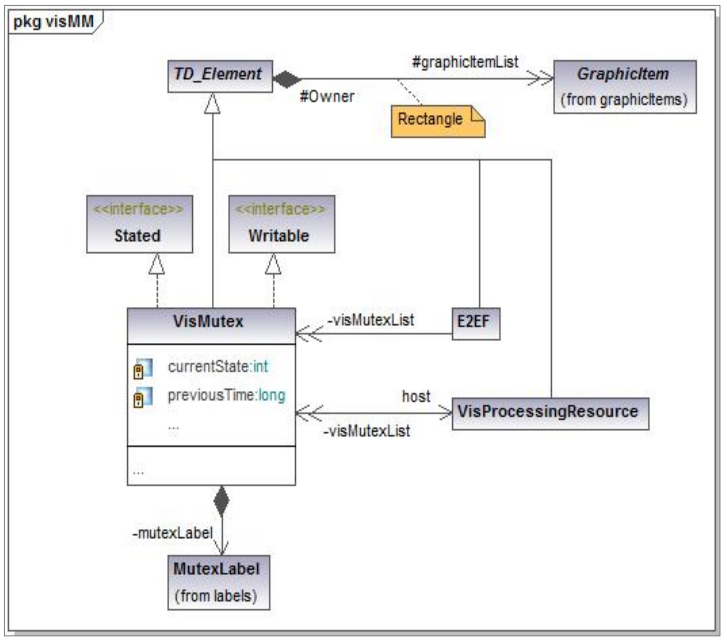

Figura 22. VisMutex

Esta subclase concreta de TD_Element es la clase del meta-modelo de visualización homóloga a la clase Mutual_Exclusion_Resource de MAST.

A los atributos heredados de TD_Element, añade los siguientes atributos:

- currentState: identificador del estado actual.
- previousTime: instante en que sucedió el último cambio de estado.

La clase implementa tanto los métodos abstractos definidos en su superclase TD_Element como los definidos en las interfaces que implementa. La implementación de los primeros merece ser brevemente detallada a continuación.

- **public void** complete()
  Establece la referencia host
- **public void** processMessage(**long** time, **int** mID)
  Añade a la lista de elementos gráficos uno que represente al mútex en su estado previo a la llegada del mensaje, desde el instante en que sucedió el último cambio de estado hasta el instante en que llega el mensaje.
- **public void** draw(**int** y, TD_TimeGUI gui, GC gc)
  Dibuja en un *canvas* de gui el mútex como una banda (compuesta de una sucesión de rectángulos) a partir de la altura y pasada como parámetro.

3.2.2.4. Clase *VisSchedulableResource*

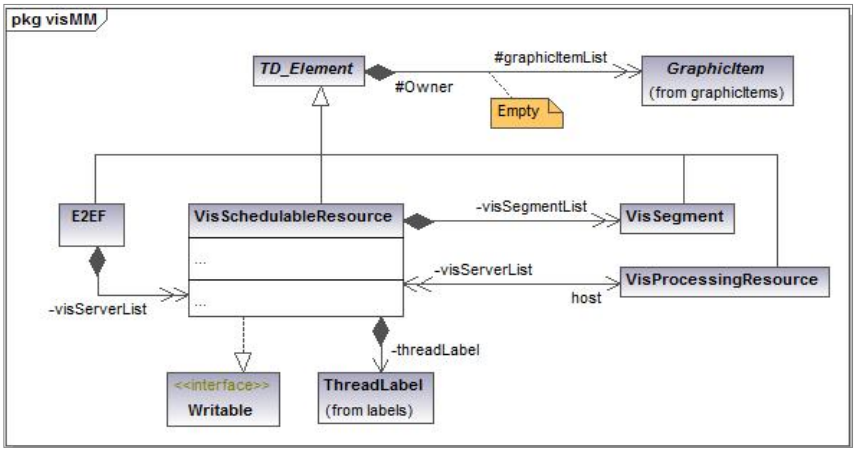

Figura 23. VisSchedulableResource

Esta subclase concreta de TD_Element es la clase del meta-modelo de visualización homóloga a la clase Schedulable_Resource de MAST.

A los atributos heredados de TD_Element, añade las siguientes referencias a otros elementos de visualización:

- visSegmentList: segmentos[1] agregados a un recurso de planificación, esto es, aquellos que son ejecutados por él (pertenecientes a la transacción en la que aparece el recurso de planificación). Se crea y se inicializa en el constructor de la clase.
- host

La clase implementa tanto los métodos abstractos definidos en su superclase TD_Element como los definidos en las interfaces que implementa. La implementación de los primeros merece ser brevemente detallada a continuación.

- **public void** complete()
  Establece la referencias host
- **public void** processMessage(**long** time, **int** mID)
- Implementación vacía, pues un recurso de planificación no es una fuente de trazas.
- **public void** draw(**int** y, TD_TimeGUI gui, GC gc)
  Dibuja un *thread* en el *canvas* de gui a partir de la altura y pasada como parámetro, lo que supone dibujar la línea de *thread* como punteada y dibujar los segmentos.

3.2.2.5. Clase *E2EF*

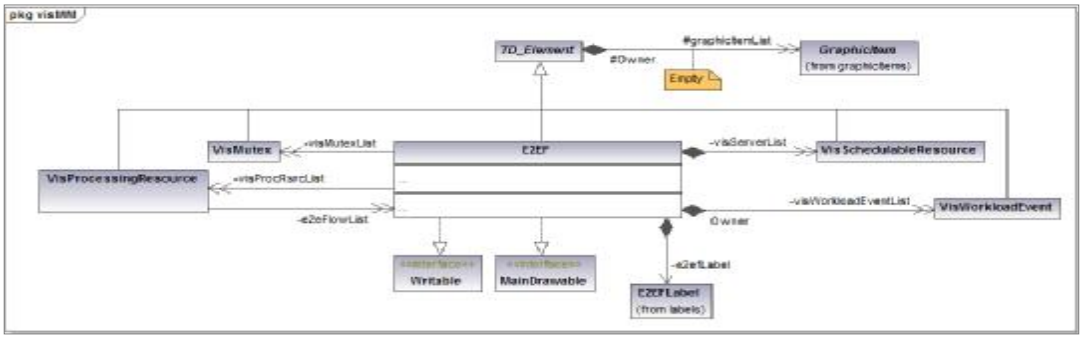

Figura 24. E2EF

[1] Un segmento es una sucesión de actividades ejecutadas por el mismo recurso de planificación

Esta subclase concreta de TD_Element es la clase del meta-modelo de visualización homóloga a la clase Regular_End_To_End_Flow de MAST.

A los atributos heredados de TD_Element, añade las siguientes referencias a otros elementos de visualización:

- visWorkloadEventList: eventos externos agregados a una transacción, esto es, aquellos que desencadenan la transacción. Se crea y se inicializa en el constructor de la clase.
- visServerList: recursos de planificación agregados a una transacción, esto es, aquellos que ejecutan las operaciones de sus actividades. Se crea y se inicializa en el constructor de la clase, aunque cada uno de los recursos de planificación habrán de ser posteriormente completados.
- visProcRsrcList: recursos de procesamiento asociados a una transacción, esto es, aquellos que son *host* de los recursos de planificación agregados a la transacción.
- visMutexList: mútexes asociados a una transacción, esto es, aquellos tomados por operaciones de sus actividades.

La clase implementa tanto los métodos abstractos definidos en su superclase TD_Element como los definidos en las interfaces que implementa. La implementación de los primeros merece ser brevemente detallada a continuación.

- **public void** complete()
  Completa cada recurso de planificación agregado y establece las siguientes referencias:
  - visProcRsrcList
  - visMutexList
- **public void** processMessage(**long** time, **int** mID)
- Implementación vacía, pues una transacción no es una fuente de trazas.
- **public void** draw(**int** y, TD_TimeGUI gui, GC gc)
  Dibuja una transacción en el *canvas* de gui a partir de la altura **y** pasada como parámetro, lo que supone dibujar:
  - la línea de la transacción.
  - los eventos externos.
  - las líneas de *threads.*

3.2.2.6. Clase *VisSegment*

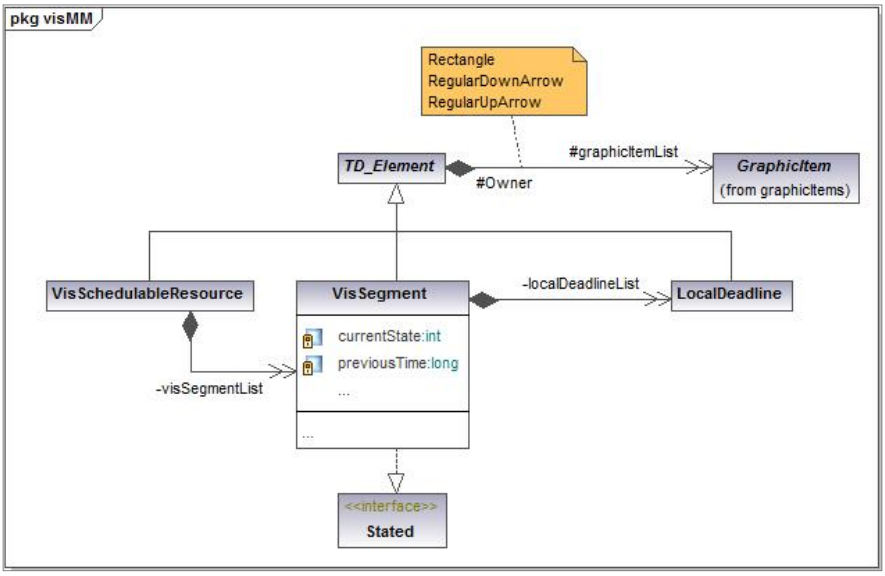

Figura 25. VisSegment

Esta subclase concreta de TD_Element no posee clase homóloga en MAST. Representa, desde el punto de vista de la visualización de un escenario de trazas, a una sucesión de actividades ejecutadas por el

mismo recurso de planificación.

A los atributos heredados de TD_Element, añade las siguientes referencias a otros elementos de visualización:

- localDeadlineList: deadlines locales agregados a un segmento. Se crea y se inicializa en el constructor de la clase.

Así como los siguientes atributos:

- currentState: identificador del estado actual.
- previousTime: instante en que sucedió el último cambio de estado.

La clase implementa tanto los métodos abstractos definidos en su superclase TD_Element como los definidos en las interfaces que implementa. La implementación de los primeros merece ser brevemente detallada a continuación.

- **public void** complete()
  Implementación vacía, pues no existen referencias cruzadas que necesiten ser establecidas.
- **public void** processMessage(**long** time, **int** mID)
  Añade a la lista de elementos gráficos uno que represente al ***segmento*** en su estado previo a la llegada del mensaje, desde el instante en que sucedió el último cambio de estado hasta el instante en que llega el mensaje.
- **public void** draw(**int** y, TD_TimeGUI gui, GC gc)
  Dibuja en un *canvas* de gui al segmento como una banda (compuesta de una sucesión de rectángulos) a partir de la altura **y** pasada como parámetro.

3.2.2.7. Clase *VisWorkloadEvent*

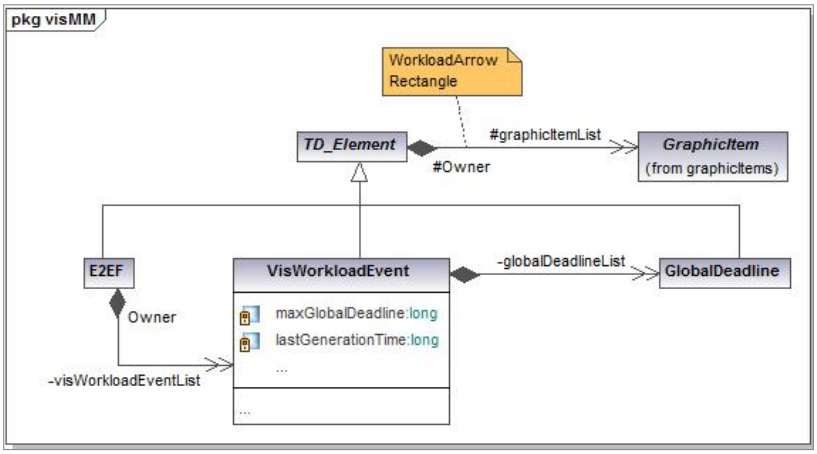

Figura 26. VisWorkloadEvent

Esta subclase concreta de TD_Element es la clase del meta-modelo de visualización homóloga a la clase Workload_Event de MAST.

A los atributos heredados de TD_Element, añade las siguientes referencias a otros elementos de visualización:

- globalDeadlineList: deadlines globales asociados a un evento externo**.** Se crea y se inicializa en el constructor de la clase.
- owner: transacción propietaria.

Así como los siguientes atributos:

- maxGlobalDeadline
- lastGenerationTime

La clase implementa los métodos abstractos definidos en su superclase TD_Element. Tal implementación merece ser brevemente detallada a continuación:

- **public void** complete()

- Implementación vacía, pues no existen referencias cruzadas que necesiten ser establecidas.
- **public void** processMessage(**long** time, **int** mID)
- **public void** draw(**int** y, TD_TimeGUI gui, GC gc)
  Dibuja en un *canvas* de gui a todo lo relacionado con un evento externo a partir de la altura y pasada como parámetro, lo que supone dibujar:
  - la flecha de aparición del evento.
  - los *deadlines* globales agregados.

3.2.3. Paquete *graphicItems*

Clase abstracta de la que heredan todas aquellas clases concretas que implementan la funcionalidad de dibujar en el canvas de la GUI los elementos gráficos necesarios para representar las trazas.

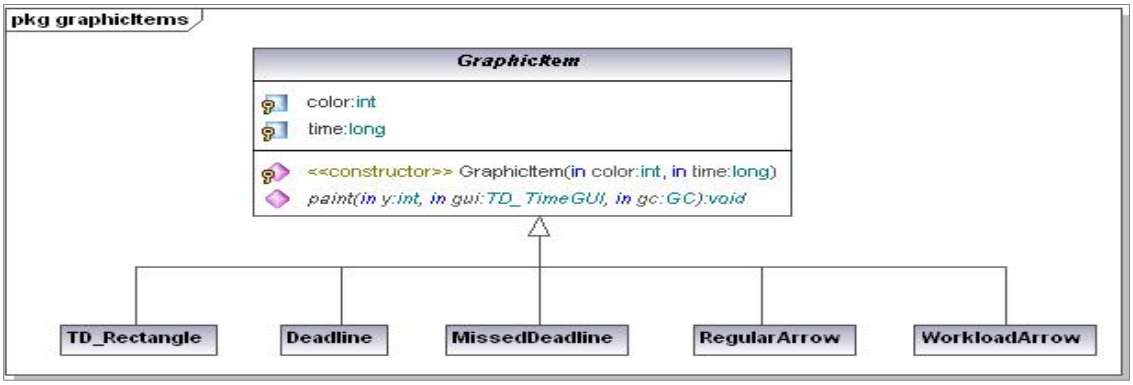

Figura 27. Jerarquía de clases del paquete graphicItems

- Campos:
  - color
  - time
- Métodos
  - **public abstract void** paint(**int** y, TD_TimeGUI gui, GC gc)
    Permite dibujar el elemento gráfico en el *canvas* asociado a gc, estableciendo la coordenada Y que se toma como referencia para comenzar el dibujo.
    La instancia TD_TimeGUI es necesario pasarla explícitamente para poder acceder a sus métodos de conversión de tiempo a coordenada X en función de los valores establecidos en sus cajas de texto *Init time* y *Final time*.

3.3. Casos de uso

A continuación se describen los casos de uso más relevantes en la aplicación, acompañados de diagramas de actividad ilustrativos.

3.3.1. *Especificación del fichero de trazas.*

El usuario de la aplicación escribirá el nombre del fichero de trazas *.trc.xml* en el campo de texto *Traces file.* Una vez rellenado el campo de texto con el nombre del fichero, deberá pulsar la tecla *ENTER.* Al pulsar la tecla *ENTER*, la aplicación será capaz de construir el modelo, inicializar el repositorio y ***parsear*** el fichero de trazas.

Al inicializar el repositorio, se instanciarán y guardarán los elementos de visualización, se pasarán a almacenar los *mutexes*, los *processing resources* y las e*nd to end flow.*

La aplicación ***parseará*** el fichero de trazas. A partir del árbol DOM se obtendrán los *Process messages* y se crearán los elementos gráficos que se irán añadiendo a su correspondiente lista de elementos gráficos.

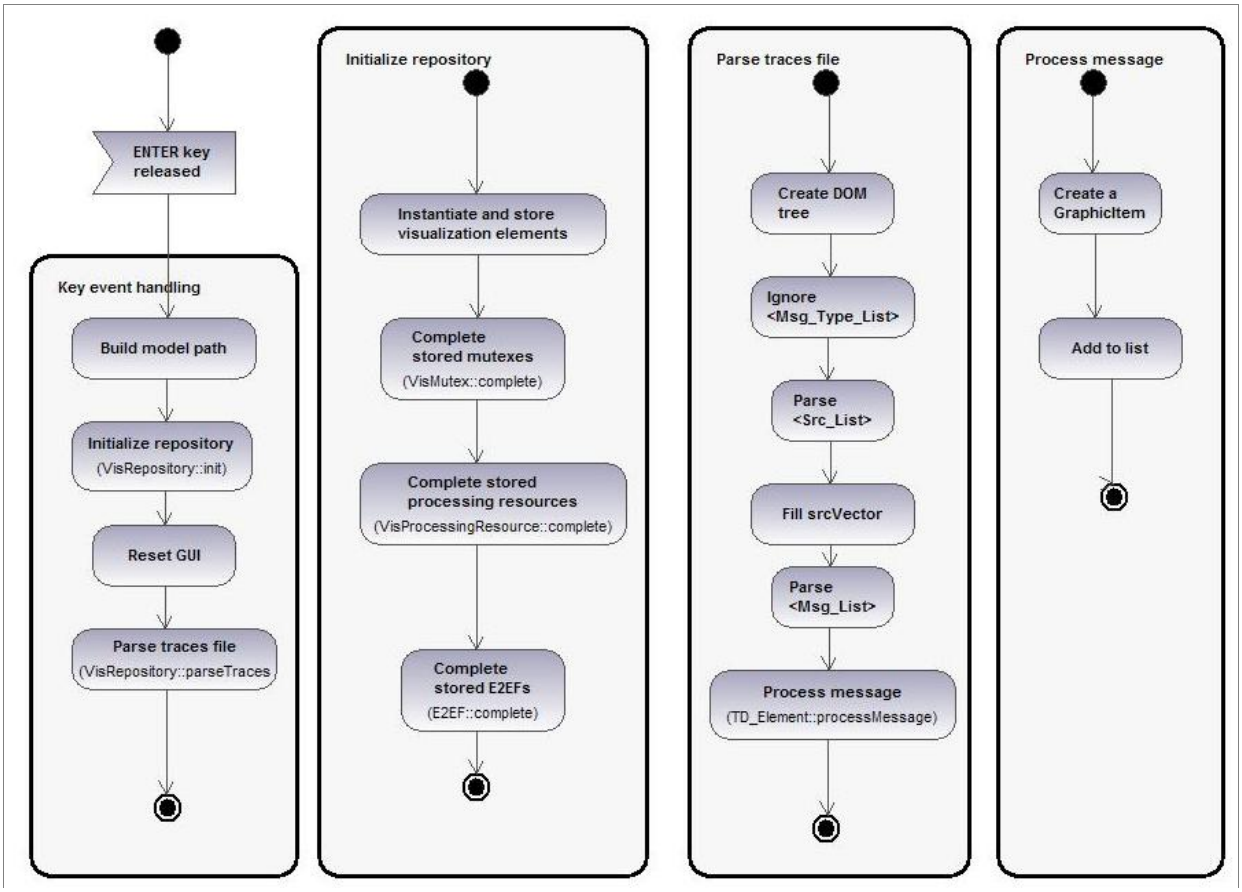

Figura 28. Diagrama de actividad para Enter trace file.

3.3.2. *Representación de la actividad global del sistema*

La aplicación ha sido arrancada y se ha especificado un fichero de trazas, esto es, se ejecutado el caso de uso descrito anteriormente en 3.3.1. A continuación el usuario selecciona la pestaña de transacciones *end-to-end flow* y se encuentra con la lista desplegable donde aparecen todas las transacciones del sistema ofrecidas para escoger una. El usuario puede elegir una de ellas seleccionándola, ante lo cual el botón *Display Selected E2EF* se habilitará, y optar por ver su representación gráfica temporal pulsando dicho botón o puede decidir no elegir ninguna transacción y optar por visualizar la actividad global del sistema pulsando el botón *DisplayAllE2EFs.* En este último caso, una vez pulsado el botón, se crea una nueva instancia de *TD_TimeGUI* bajo modo *System global activity,* que será la ventana donde se visualice la actividad del sistema. La creación de una ventana implica crear todos los elementos GUI involucrados, empezando por una instancia de la clase Shell de la librería SWT (la ventana propiamente dicha) y en particular una instancia de la clase Canvas, que será el lienzo donde se represente la visualización gráfica según la información contenida en el fichero de trazas y el modelo del sistema.

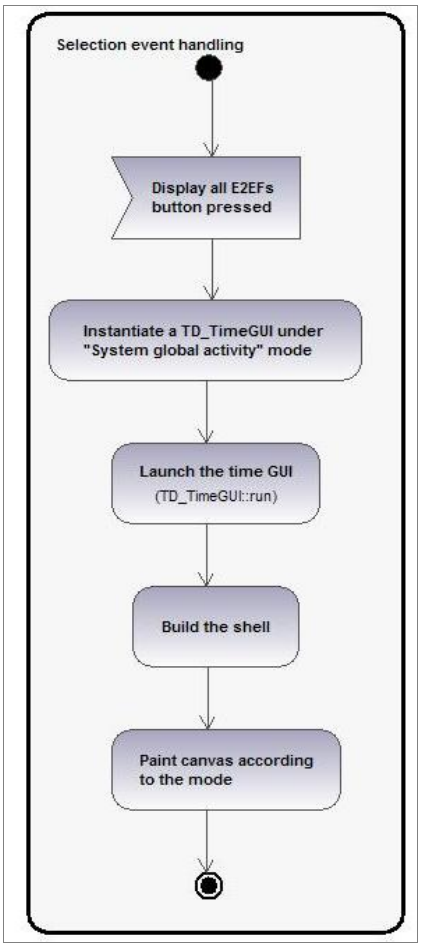

Figura 29. Diagrama de actividad para Display All E2EFs

3.3.3. Manipulación del intervalo temporal.

La GUI le permite al usuario tener un control del tiempo sobre lo que se está representando. Se podrá ampliar o reducir el rango de la representación gráfica, a modo de zoom, con los botones */2* y *x2*. También se le ofrece la posibilidad de volver al punto de partida con el botón *initTime*.

El funcionamiento interno de los botones */2* y *x2* será muy similar. Una vez el usuario pulsa cualquiera de estos botones, se comprueba si el botón no pulsado está habilitado, y en caso de no estarlo se habilita. A continuación se refrescan los campos *initTime* y *finalTime* y se vuelve a dibujar el *canvas* con los nuevos valores de tiempo. Por último, se hace una comprobación para saber si una nueva acción sobre el botón es posible. En caso de no ser posible, el botón quedará deshabilitado.

Al presionar sobre el botón *initTime*, se resetearán los campos de texto *initTime* y *finalTime* con los valores iniciales, el botón *x2* pasará a estar deshabilitado (ya que inicialmente *initTime* tiene un valor igual a cero y este nunca podrá ser inferior) y se volverá a dibujar el *canvas* con los valores iniciales de *initTime* y de *finalTime*.

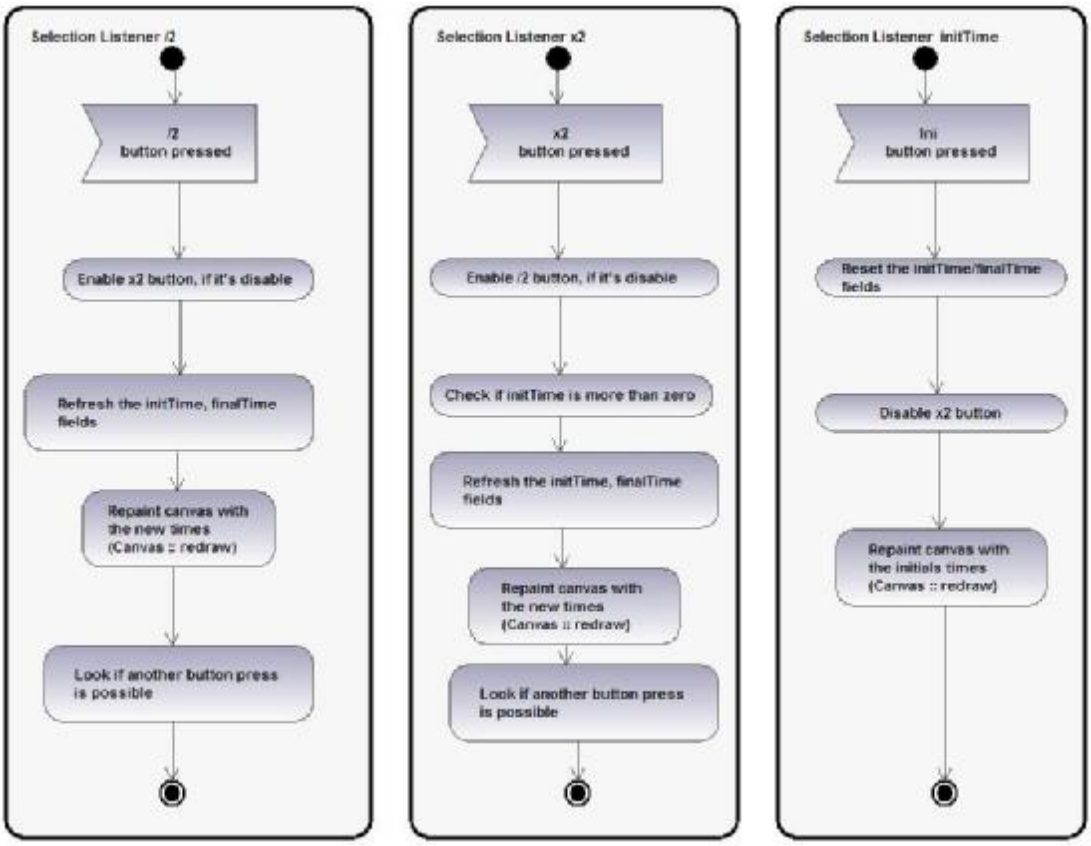

Figura 30. Diagrama de actividad para Time Control GUI

4. Ejemplo de monitorización de un sistema de tiempo real

En este apartado se muestra a través de un ejemplo de sistema de tiempo real cuyo comportamiento temporal está descrito por un modelo MAST y que a través del simulador se ha generado un fichero de trazas. Con el ejemplo se muestra la operatividad y capacidad del visualizador “Traces_Displayer” para mostrar la evolución de un sistema.

4.1. PowerDistortion: Ejemplo de sistema de tiempo real

La aplicación *PowerDistortion* tiene como objetivo la monitorización de la calidad de servicio de una subestación eléctrica y la generación de señales de alarma cuando esta alcanza un cierto umbral. Se monitoriza la distorsión de las formas de onda de las intensidades de las tres fases R, S y T de un conjunto de líneas trifásicas de la subestación, en base a la medida del tanto por ciento de tercer armónico que presentan las señales, y la alarma se lanza cuando en alguna de las líneas la distorsión alcanza el 10%.

La evaluación de la distorsión de tercer armónico de cada línea trifásica se basa en el muestreo de la misma a razón de 32 muestra por cada ciclo de 50 Hz (esto es cada 625 μs). El muestreo de cada 15 señales de intensidad (las 3 fases de un grupo de 5 líneas trifásicas) se realiza mediante el hardware de una tarjeta de adquisición DAC. Dotada de un conversor A/D de 12 bits y 10 μs de tiempo de conversión.

En la figura 31 se muestra un esquema del sistema distribuido que ejecuta esta aplicación. Su modo de operación es el siguiente:

- El reloj interno de la tarjeta hace que cada 625 μs la tarjeta DAC muestrea las 15 señales y las almacena en su pequeña memoria interna de 2048 bytes.
- Cuando la ocupación de la memoria alcanza el 50% genera la interrupción *HalfMemoryFull.*
- La aplicación a través de la tarea *DACHandler* atiende la interrupción, leyendo las 512 muestra (1024 bytes) almacenada en la mitad de la memoria ya ocupada. Conforma la forma de onda relativa a un ciclo de las fases de una línea trifásicas y la transfiere como un paquete UDP hacia el computador que tiene el servicio de evaluación de la distorsión y la generación de alarmas..
- La aplicación tiene tres requisitos de tiempo real:
    - No puede haber pérdida de muestras, por lo que tras la interrupción *HalfMemoryFull*, la información debe ser leída y procesada antes de que sea leída la siguiente mitad de memoria.
    - El fallo *DistortionAlarm* que representa que una línea tenga una distorsión superior al 10% debe ser detectado antes de que transcurra 1 segundo desde que se ha producido.
    - Una vez que se detecta una alarma *DistortionAlarm, el sistema genera* una acción de corrección que se ejecuta en la configuración de la red. Esta alarma esporádica y se produce muy rara vez. Pero cuando ocurre debe ejecutarse con un retraso inferior a 500 ms.

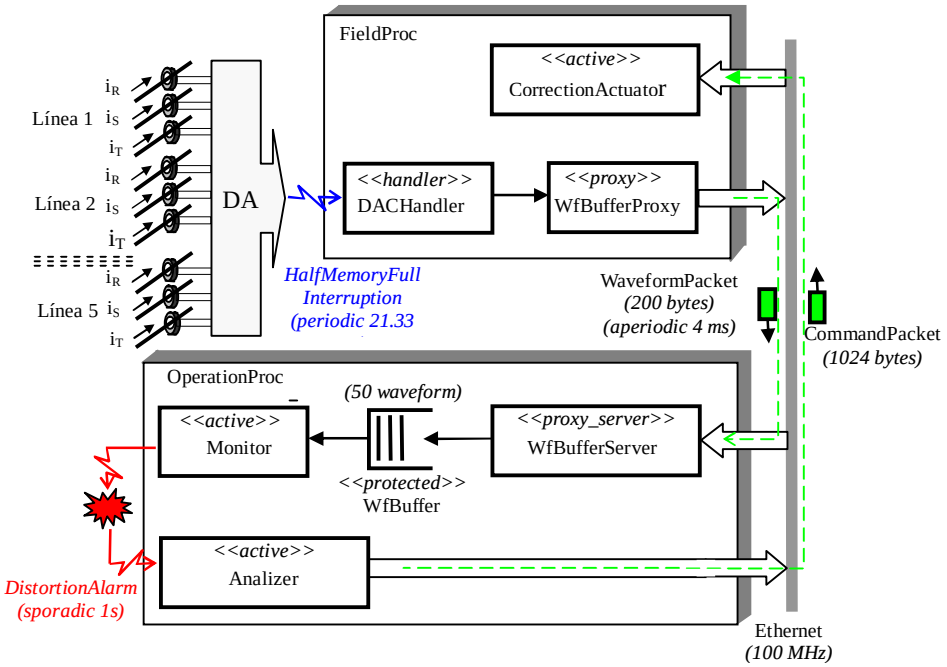

Figura 31.Plataforma y módulos lógicos de la aplicación PowerDistortion.

- La aplicación tiene naturaleza distribuida. Se ejecuta en dos procesadores: *FieldProc* localizado en la subestación y *ControlProc* ubicado en la sala de control de la red.
- En el procesador *FieldProc* está ubicada la tarjeta de adquisición de señales (DAC) y el manejador (*DACHandler*) de los eventos (*HalfMemoryFull*) que se generan cuando su memoria está ocupada en un 50% de su capacidad (2096 bytes = 1024 muestras). La tarjeta muestrea las 15 señales a razón de 32 muestras por ciclo de la red (20 ms), por lo que el tiempo entre generación de los eventos *HalfMemoryFull* es de (1024/2)/(50*32* 15) = 21.33 ms.
- El *DACHandler* lee de la tarjeta las 512 muestras capturadas, las conforma en formas de ondas de 32*3 muestras por línea(192 bytes), y las data con el instante de captura (8 bytes), resultando un paquete con 200 bytes de conversión. Los cuales los pasa al módulo *WfBufferProxy* para que los transmita como un paquete UDP (*WfPacket*) a través de la red *Ethernet.* El tiempo medio de transferencia de paquetes con formas de onda es de (1/(50*5)= 4ms), sin embargo la transmisión no es periódicas sino de tipo ráfaga, con grupos de 5 paquetes (uno por línea) cada 20 ms, con otros 5 extras cada 15*20=320 ms, como consecuencia de que la capacidad de la memoria no es un múltiplo exacto del tamaño de la forma de onda. (512/480=1+ 1/15). Sin embargo, por los tiempo que requiere la elaboración de los paquetes en *WfBufferProxy* entre transmisión de paquetes hay un tiempo mínimo de 0.256 ms.
- En el procesador *OperationProc* los paquetes UDP *WfPacket* son recibidos por el módulo *WfBufferServer* y los encola en *WfBuffer* para que sean procesados cuando se disponga capacidad para ello. Cada paquete que se reciba debe ser procesado antes de que se reciba el siguiente paquete, a fin de que no se pierda, por lo que entre la recepción y el encolado debe cumplirse un plazo máximo de 0,256 ms (tiempo mínimo de transmisión entre paquetes).
- En el procesador *OperationProc* está instalado módulo *Monitor* que lee las formas de ondas disponibles en *WfBuffer* las procesa evaluando el tanto por ciento de tercer armónico de cada fase, y en el caso de que este supere el umbral establecido del 10%, genera la alarma *DistortionAlarm*.
- También en el procesador *OperationProc*, está instalado el módulo *DistortionAnalyzer* que de acuerdo con el estado global de la red, determina la corrección que debe llevarse en su topología para eliminar la

situación de alarma. La corrección se transfiere al procesador *FieldProc* como un paquete UDP *CorrectionCommand* con un tamaño de 1024 bytes.

- Por último, en el procesador *FieldProc*, el módulo *CorrectionActuator* se recibe el *CorrectionCommand* y se ejecuta la acción que se ordena en él.

En esta aplicación existen dos transacciones (*End_To_End_Flow*), ambas con requisitos de tiempo real:

- En la figura 32 se muestran la secuencia de invocaciones de métodos entre objetos que constituyen la transacción *WfProcessing*, que es disparada con la interrupción hardware *HalfMemoryFull* que se genera en la tarjeta hardware DAC y que finaliza cuando se han procesado todas las forma de ondas de señales que se tienen adquiridas con la información leída de la memoria Habitualmente 15 formas de onda, pero a veces, hasta 30 formas de onda sucesivas. Esta transacción tiene varios requisitos temporales:
  - El método *handle()* debe haber finalizado antes de que se genere la siguiente interrupción hardware. *GolbalDeadline* con *deadline* = 21.33 ms
  - El método *receive()* debe haber finalizado antes de que se reciba el siguiente paquete. *LocalDeadline* con *deadline* = 0.256 ms
  - El método *analize()* debe haber finalizado antes de que transcurra un segundo desde el inicio de la transacción. *GlobalDeadline* con *deadline*=1 s.
- En la figura 33 se muestra la secuencia de invocaciones de métodos que constituyen la transacción *DistortionAdjust*. Esta transacción se genera cuando en la ejecución del método *analize()* se genera una *DistortionAlarm*. Este es un fenómeno muy esporádico, y por ello se desacopla su flujo del de la transacción *WfProcess.* Esta transacción tiene un único requisito temporal:
  - El método *adjust()* de be finalizar antes de que transcurra un plazo de 1 segundo desde que se inicio la transacción. *GlobalDeadline* con *deadline*= 1s.

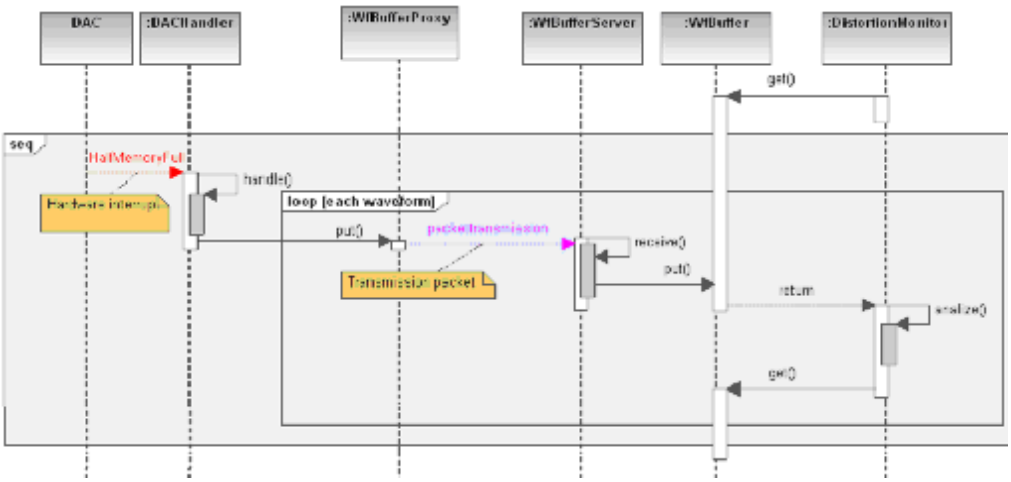

Figura 32. Actividad de la transacción WfProcessing.

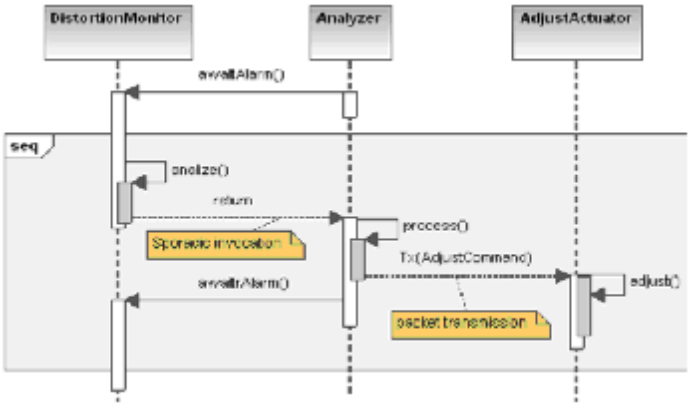

Figura 33. Actividad de la transacción DistortionAdjust.

4.2. Modelo MAST del ejemplo

En la tabla 2, se muestra el modelo de tiempo real de la aplicación. Desde el punto de vista de la aplicación de visualización de trazas que se ha desarrollado en este trabajo, lo relevante del modelo son los aspectos estructurales, esto es, los elementos que constituyen el modelo y sus relaciones de agregación, a fin de poder seleccionar las trazas de interés y los elementos que son fuentes de ellas.

Tabla 2. Modelo de tiempo real de la situación de tiempo real

```
<?xml version="1.0" encoding="UTF-8"?>
<!--**************************************************************************************************************
                    Proyecto MAST
      Grupo de Computadores y tiempo real (CTR)
                Universidad de Cantabria
      Model: PowerDistortion
      Authors:: Emilio Ruíz, C. Cuevas y J.M. Drake
      Version: 16-12-2011
**************************************************************************************************************-->
<mast_mdl:MAST_MODEL
[1]           xmlns:mast_mdl="http://mast.unican.es/xmlmast/model"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://mast.unican.es/xmlmast/model ../JSimMast_V_Schemas/Mast2_Model_V.xsd"
              Model_Name="PowerDistortion"
              Model_Date="2011-12-16T00:00:00">
      <!-- ************************************** EXECUTION PLATFORM MODEL*********************************-->
      <!-- ********* Processor FieldProc ***-->
      <mast_mdl:Regular_Processor Name="FieldProc"
                  Speed_Factor="1.0"
[2]               System_Timer="FieldProc_Timer">
                  <mast_mdl:Timer Name="FieldProc_Timer"/>
      </mast_mdl:Regular_Processor>
      <mast_mdl:Ticker Name="FieldProc_Timer"
                  Min_Overhead="7.5E-6" Avg_Overhead="7.5E-6" Max_Overhead="7.5E-6"/>
[3]   <mast_mdl:Primary_Scheduler Name="FieldProc_Scheduler" Host="FieldProc">
            <mast_mdl:Fixed_Priority_Policy
[4]               Min_Priority="1" Max_Priority="31"
                  Worst_Context_Switch="15.5E-6" Avg_Context_Switch="14.2E-6"
                  Best_Context_Switch="14.0E-6"/>
      </mast_mdl:Primary_Scheduler>
      <!-- ********* Processor OperationProc ***-->
      <mast_mdl:Regular_Processor Name="OperationProc"
                  Speed_Factor="1.0"
[5]               System_Timer="Operation_Clock">
            <mast_mdl:Timer Name="Operation_Clock"/>
      </mast_mdl:Regular_Processor>
```

```
        <mast_mdl:Alarm_Clock  Name="Operation_Clock"
                    Min_Overhead="0.75E-6" Avg_Overhead="0.75E-6" Max_Overhead="0.75E-6"/>
[6]     <mast_mdl:Primary_Scheduler Name="OperationProc_Scheduler" Host="OperationProc">
            <mast_mdl:Fixed_Priority_Policy
[7]                     Min_Priority="11" Max_Priority="59"
                        Worst_Context_Switch="12.5E-6" Avg_Context_Switch="10.5E-6"
                        Best_Context_Switch="9.8E-6"/>
        </mast_mdl:Primary_Scheduler>
        <!-- ********* Network Ethernet ***-->
        <mast_mdl:Packet_Based_Network Name="Ethernet"
                    Max_Blocking="1500" Max_Packet_Size="1500" Min_Packet_Size="40"
[8]                 Speed_Factor="1.0" Throughput="100000000" Transmission_Kind="HALF_DUPLEX"/>
        <mast_mdl:Primary_Scheduler Name="Ethernet_Scheduler" Host="Ethernet">
            <mast_mdl:Fixed_Priority_Policy  Min_Priority="0" Max_Priority="7"/>
[9]     </mast_mdl:Primary_Scheduler>
        <!-- ************************************** LOGIC MODEL ********************************************************-->
        <!--  ******** DACHandler  module ***-->
        <mast_mdl:Simple_Operation Name="DACHandler.handle"
                    Worst_Case_Execution_Time="3.84E-3" Avg_Case_Execution_Time="3.84E-3"
[10]                Best_Case_Execution_Time="3.84E-3"/>
        <mast_mdl:Thread Name="DacHandler_Thr" Scheduler="FieldProc_Scheduler">
            <mast_mdl:Fixed_Priority_Params Priority="20"/>
[11]        </mast_mdl:Thread>
        <!--  ********* WfBufferProxy  module ***-->
        <mast_mdl:Simple_Operation Name="WfBufferProxy.put"
                    Worst_Case_Execution_Time="0.256E-3"  Avg_Case_Execution_Time="0.2560E-3"
[12]                Best_Case_Execution_Time="0.2560E-3"/>
        <mast_mdl:Message Name="WaveformPacket"
                    Max_Message_Size="244" Min_Message_Size="244" Avg_Message_Size="244"/>
[13]    <mast_mdl:Communication_Channel  Name="WaveformPacket_Cch" Scheduler="Ethernet_Scheduler">
            <mast_mdl:Fixed_Priority_Params Priority="7"/>
[14]    </mast_mdl:Communication_Channel>
        <!--  ********* WfBufferServer  module ***-->
        <mast_mdl:Simple_Operation Name="WfBufferServer.receive"
                    Worst_Case_Execution_Time="0.80E-3"  Avg_Case_Execution_Time="0.80E-3"
[15]                Best_Case_Execution_Time="0.80E-3"/>
        <mast_mdl:Thread Name="WfBufferServer_Thr" Scheduler="OperationProc_Scheduler">
            <mast_mdl:Fixed_Priority_Params Priority="20"/>
[16]    </mast_mdl:Thread>
        <!-- ******** WfBuffer class ***-->
        <mast_mdl:Immediate_Ceiling_Mutex Name="WfBuffer.Mtx" Ceiling="20"/>
        <mast_mdl:Simple_Operation Name="WfBuffer.put"
[17]                Worst_Case_Execution_Time="1.150E-3" Avg_Case_Execution_Time="1.150E-3"
[18]                Best_Case_Execution_Time="1.150E-3">
            <mast_mdl:Mutex Name="WfBuffer.Mtx"/>
        </mast_mdl:Simple_Operation>
        <mast_mdl:Simple_Operation Name="WfBuffer.get"
                    Worst_Case_Execution_Time="0.90E-3" Avg_Case_Execution_Time="0.90E-3"
[19]                Best_Case_Execution_Time="0.90E-3">
            <mast_mdl:Mutex Name="WfBuffer.Mtx"/>
        </mast_mdl:Simple_Operation>
        <!-- ******** DistortionMonitor class ***-->
        <mast_mdl:Simple_Operation Name="Analizer.analize"
                    Worst_Case_Execution_Time="5.3E-3" Avg_Case_Execution_Time="5.1E-3"
[20]                Best_Case_Execution_Time="4.8E-3"/>
        <mast_mdl:Thread Name="DistortionMonitor_Thr" Scheduler="OperationProc_Scheduler">
            <mast_mdl:Fixed_Priority_Params Priority="10"/>
[21]    </mast_mdl:Thread>
        <!-- ******** Analizer class ***-->
        <mast_mdl:Simple_Operation Name="Analizer.process"
                    Worst_Case_Execution_Time="7.3E-3" Avg_Case_Execution_Time="7.1E-3"
[22]                Best_Case_Execution_Time="6.8E-3"/>
        <mast_mdl:Message Name="CommandPacket"
                    Max_Message_Size="1064" Min_Message_Size="1064" Avg_Message_Size="1064"/>
[23]    <mast_mdl:Thread Name="Analyzer_Thr" Scheduler="OperationProc_Scheduler">
            <mast_mdl:Fixed_Priority_Params Priority="15"/>
[24]    </mast_mdl:Thread>
        <mast_mdl:Communication_Channel Name="CommadPacket_Cch" Scheduler="Ethernet_Scheduler">
            <mast_mdl:Fixed_Priority_Params Priority="5"/>
[25]    </mast_mdl:Communication_Channel>
        <!-- ******** AdjustmentActuator class ***-->
        <mast_mdl:Simple_Operation Name="AdjustmentActuator.adjust"
                    Worst_Case_Execution_Time="45.7E-3" Avg_Case_Execution_Time="45.7E-3"
[26]                Best_Case_Execution_Time="45.7E-3"/>
        <mast_mdl:Thread Name="AdjustmentActuator_Thr" Scheduler="FieldProc_Scheduler">
```

```
                <mast_mdl:Fixed_Priority_Params Priority="15"/>
[27]        </mast_mdl:Thread>
            <!-- ****************************** REAL-TIME SITUATION MODEL ********************************************-->
            <!-- ********** WfProcessing_e2ef ***-->
            <mast_mdl:Regular_End_To_End_Flow Name="WfProcessing_e2ef">
                <mast_mdl:Periodic_Event Name="HalfMemoryFull" Period="21.33E-3"/>
[28]            <mast_mdl:Internal_Event Name="memRec">
[28a]               <mast_mdl:Hard_Global_Deadline Referenced_Event="HalfMemoryFull" Deadline="21.33E-3"/>
[28b]           </mast_mdl:Internal_Event>
                <mast_mdl:Internal_Event Name="wfGenO"/>
                <mast_mdl:Internal_Event Name="wfGenC"/>
                <mast_mdl:Internal_Event Name="wfGenE"/>
                <mast_mdl:Internal_Event Name="wfGenP"/>
                <mast_mdl:Internal_Event Name="wfGen0"/>
                <mast_mdl:Internal_Event Name="wfGen1"/>
                <mast_mdl:Internal_Event Name="wfGen2"/>
                <mast_mdl:Internal_Event Name="wfGen3"/>
                <mast_mdl:Internal_Event Name="wfGen4"/>
                <mast_mdl:Internal_Event Name="wfGen5"/>
                <mast_mdl:Internal_Event Name="wfGen6"/>
                <mast_mdl:Internal_Event Name="wfGen7"/>
                <mast_mdl:Internal_Event Name="wfGen8"/>
                <mast_mdl:Internal_Event Name="wfGen9"/>
                <mast_mdl:Internal_Event Name="wfGen10"/>
                <mast_mdl:Internal_Event Name="wfGen11"/>
                <mast_mdl:Internal_Event Name="wfGen12"/>
                <mast_mdl:Internal_Event Name="wfGen13"/>
                <mast_mdl:Internal_Event Name="wfGen14"/>
                <mast_mdl:Internal_Event Name="wfGen"/>
                <mast_mdl:Internal_Event Name="wfPacketTxt"/>
                <mast_mdl:Internal_Event Name="wfPacketRec"/>
                <mast_mdl:Internal_Event Name="wfPacketAttended">
                    <mast_mdl:Hard_Local_Deadline Deadline="0.256E-3"/>
[28c]           </mast_mdl:Internal_Event>
                <mast_mdl:Internal_Event Name="wfPacketQueued"/>
                <mast_mdl:Internal_Event Name="wfPacketDequeued"/>
                <mast_mdl:Internal_Event Name="wfProcessed">
                    <mast_mdl:Hard_Global_Deadline Referenced_Event="HalfMemoryFull" Deadline="1.0"/>
[28d]           </mast_mdl:Internal_Event>
                <mast_mdl:Step Input_Event="HalfMemoryFull" Output_Event="memRec"
                        Step_Schedulable_Resource="DacHandler_Thr" Step_Operation="DACHandler.handle"/>
[28e]           <mast_mdl:Fork Input_Event="memRec" Output_Events_List="wfGenO wfGenC"/>
                <mast_mdl:Rate_Divisor Input_Event="wfGenC" Output_Event="wfGenE" Rate_Factor="15"/>
[28f]           <mast_mdl:Merge Input_Events_List="wfGenO wfGenE" Output_Event="wfGenP"/>
                <mast_mdl:Fork Input_Event="wfGenP" Output_Events_List="wfGen0 wfGen1 wfGen2 wfGen3 wfGen4
[28g]           wfGen5 wfGen6 wfGen7 wfGen8 wfGen9 wfGen10 wfGen11 wfGen12 wfGen13 wfGen14"/>
[28h]           <mast_mdl:Merge Output_Event="wfGen" Input_Events_List="wfGen0 wfGen1 wfGen2 wfGen3 wfGen4
                wfGen5 wfGen6 wfGen7 wfGen8 wfGen9 wfGen10 wfGen11 wfGen12 wfGen13 wfGen14"/>
[28i]           <mast_mdl:Step Input_Event="wfGen" Output_Event="wfPacketTxt"
                        Step_Schedulable_Resource="DacHandler_Thr" Step_Operation="WfBufferProxy.put"/>
[28j]           <mast_mdl:Step Input_Event="wfPacketTxt" Output_Event="wfPacketRec"
                        Step_Schedulable_Resource="WaveformPacket_Cch" Step_Operation="WaveformPacket"/>
[28k]           <mast_mdl:Step Input_Event="wfPacketRec" Output_Event="wfPacketAttended"
                        Step_Schedulable_Resource="WfBufferServer_Thr" Step_Operation="WfBufferServer.receive"/>
[28l]           <mast_mdl:Step Input_Event="wfPacketAttended" Output_Event="wfPacketQueued"
                        Step_Schedulable_Resource="WfBufferServer_Thr" Step_Operation="WfBuffer.put"/>
                <mast_mdl:Step Input_Event="wfPacketQueued" Output_Event="wfPacketDequeued"
[28m                    Step_Schedulable_Resource="DistortionMonitor_Thr" Step_Operation="WfBuffer.get"/>
]               <mast_mdl:Step Input_Event="wfPacketDequeued" Output_Event="wfProcessed"
                        Step_Schedulable_Resource="DistortionMonitor_Thr" Step_Operation="Analizer.analize"/>
[28n]           </mast_mdl:Regular_End_To_End_Flow>
                <!-- ********* End-to-end-flow DistortionAdjust_e2ef ***-->
[28o]           <mast_mdl:Regular_End_To_End_Flow Name="DistortionAdjust_e2ef">
                    <mast_mdl:Periodic_Event Name="DistortionAlarm" Period="1.0" Phase="1.0"/>
                    <mast_mdl:Internal_Event Name="comndTxt"/>
                    <mast_mdl:Internal_Event Name="comndRec"/>
[29]                <mast_mdl:Internal_Event Name="corrected">
[29a]                   <mast_mdl:Hard_Global_Deadline Referenced_Event="DistortionAlarm" Deadline="0.5"/>
                    </mast_mdl:Internal_Event>
                    <mast_mdl:Step Input_Event="DistortionAlarm" Output_Event="comndTxt"
[29b]                       Step_Schedulable_Resource="Analyzer_Thr" Step_Operation="Analizer.process"/>
                    <mast_mdl:Step Input_Event="comndTxt" Output_Event="comndRec"
                        Step_Schedulable_Resource="CommadPacket_Cch" Step_Operation="CommandPacket"/>
[29c]               <mast_mdl:Step Input_Event="comndRec" Output_Event="corrected"
                        Step_Schedulable_Resource="AdjustmentActuator_Thr"
```

```
[29d]                    Step_Operation="AdjustmentActuator.adjust"/>
                    </mast_mdl:Regular_End_To_End_Flow>
[29e]               <!-- ************************************************************-->
</mast_mdl:MAST_MODEL>
```

Notas:

[2][3][4] Definen las características del procesador *FieldProc*, de su temporizador tipo ticker *FieldProc_Timer* y de su planificador con política de prioridades fijas *FieldProc_Scheduler*.

[5][6][7] Definen las características del procesador Operation*Proc*, de su temporizador tipo alarm clock Operation_*Clock* y de su planificador con política de prioridades fijas Operation*Proc_Scheduler*.

[8][9] Define las características de la red de comunicaciones *Ethernet* y de su planificador *Ethernet_Scheduler*.

[10][11] Define los elementos del módulo DACHandler: el tiempo de ejecución de la operación *handle*, y el thread que como módulo activo introduce para planificar su actividad.

[12][13][14] Define los elementos que introduce el módulo *WFBufferProxy*. La operación *put* que corresponde a la gestión de la transmisión de una forma de onda leida como un paquete UDP por la red Ethrenet, las características del mensaje *WaveformPacket* con el que se transmite una forma de onda y el canal de comunicación WaveformPacket_Cch con el que se planifica la transmisión del mensaje.

[15][16] Define los elementos que introduce el módulo *WFBufferServer* con el que se reciben los mensajes con las formas de onda que deben ser transferidos para su análisis: La operación *receive* de recepción del mensaje y el thread *WFBufferServer_Thr* que queda a la espera de la recepción de un paquete.

[17][18][19] Define los elementos que introduce el módulo *WFBuffer* con el que se desacopla la recepción de las formas de onda y su procesamiento. Es un objeto protegido ya que su información debe ser introducida *put* y leída *get*, por lo que se introduce un mutex *WfBuffer.Mtx* que debe ser accedido por ambas operaciones.

[20][21] Define los elementos que introduce el módulo *DistortionMonitor* con el que se monitorizan en la consola a una prioridad más baja las formas de onda capturadas: La operación *analize* que modela el tiempo de procesamiento de cada forma de onda y su monitorización y el thread *DistortionMonitor_Thr* que planifica su ejecución.

[22][23][24][25] Define los elementos que introduce el módulo *Analyzer_Thr*: La operación *process* que describe el tiempo de ejecución con la que se evalúa el ajuste de la red eléctrica que se requiere, las características del mensaje CommandPacket que se transmite y el thread *Analyzer_Thr* y el canal de comunicación *CommandPacket_Cch* en los que se planifica su ejecución y su transmisión.

[26][27] Define los elementos que introduce el módulo *AdjustmentActuator* con el que se realiza la reconfiguración de la red eléctrica: La operación *adjust* que modela el tiempo de procesamiento de la reconfiguración y el thread *AdjustmentActuator_Thr* que planifica su ejecución.

[28] Modela la transacción *EFProcessing_e2ef* con la que se procesan las formas de onda:

- [28a] Describe la frecuencia con la que se ejecuta.
- [28b] Requerimiento temporal que información leida debe ser procesada ante de que se reescriba la memoria de la targeta.
- [28c] Requiere que el thread que recibe cada paquete UDP quede libre antes de que se reciba el paquete siguiente.
- [28d] Requiere que cada forma de onda sea procesada con un plazo de 1 segundo contado desde que se capturó la forma de onda.
- [28e] Describe la invocación del metodo DACHandler.handle que lee y conforma las formas de ondas.
- [28f,g,h,i] Modela las formas de onda que se generan por cada media memoria leída.
- [28j] Modela la construcción del paquete de cada forma de onda para su transmisoión por la red.
- [28k] Modela la transmisión de cada paquete de forma de onda por la red.
- [28l] Modela la recepción de un paquete con una forma de onda desde la red.
- [28m] Modela la introducción del buffer de una forma de onda,
- [28n] Modela la lectura del buffer de una forma de onda.
- [28o] Modela el procesado y monitorización de la distorsión de la forma de onda.

[29] Modela la transacción *DistortionAdjust_e2ef* con la que se actualiza la red si hay una alarma por distorsión alta:

- [29a] Describe la frecuencia con la que se ejecuta.
- [29b] Requerimiento temporal de que el ajuste de la red eléctrica se reliza antes 500 ms.
- [29c] Describe la invocación del método *process* del módulo *Analyzer* para procesar el cambio de la red que se requiere y elabora el mensaje que lo transmite.
- [28d] Describe el envío por la red Ethrnet del paquete que requiere la reconfiguración de la red.
- [28e] Describe la recepción del paquete y la ejecución del la reconfiguración.

4.3 Generación del fichero de trazas

El entorno MAST proporciona un conjunto de herramientas para su análisis. Estas herramientas se lanzan desde la interfaz gráfica que se muestra en la figura 34. Dentro de estas herramientas se encuentran:

- Análisis de planificabilidad.
- Satisfacción de los requerimientos temporales.
- Cálculo de holguras.
- Asignación de prioridades.
- Análisis por simulación.

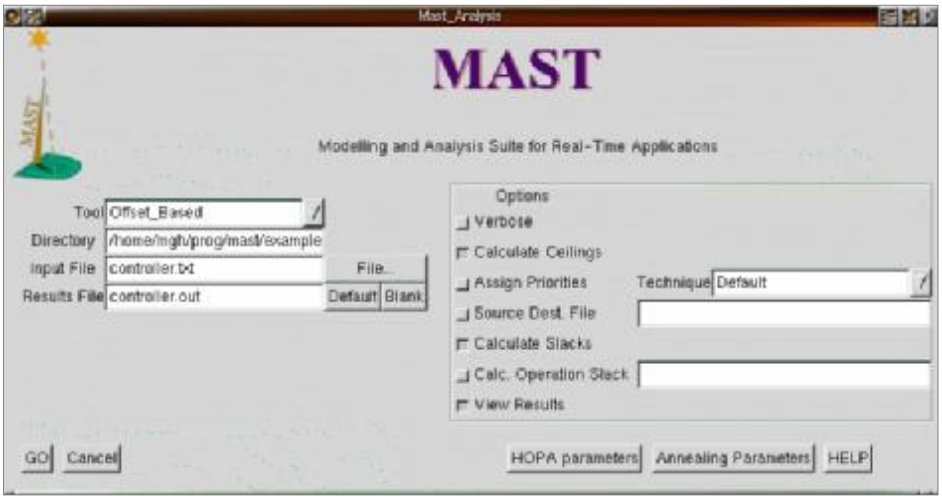

Figura 34. Ventana de lanzamiento de las herramientas del entorno MAST.

El fichero de trazas para el que se ha diseñado el visualizador que se presenta en este trabajo se obtiene a partir del simulador Sim_MAST [9]. La ventana de control del simulador se muestra en la figura 35.

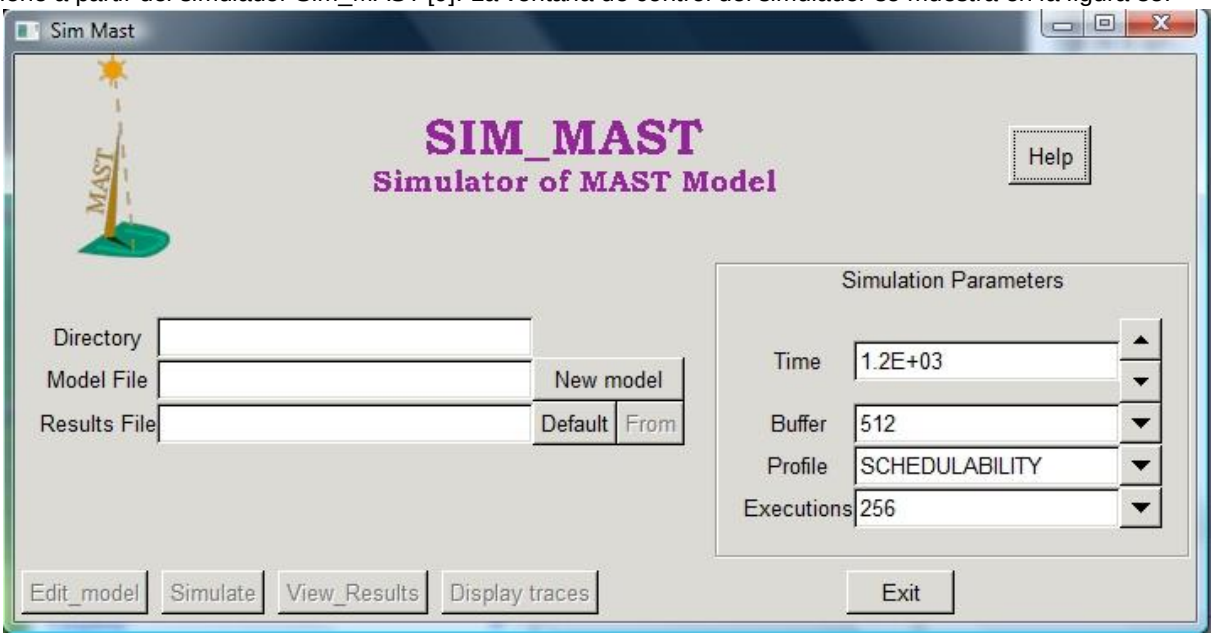

Figura 35. Ventana de control del simulador. JSimMast.

El simulador genera un fichero de traza que es visualizable por la herramienta TracesDisplayer desarrollada.

4.4 Visualización del fichero de trazas

La herramienta de visualización va a operar sobre los ficheros de trazas *PowerDistortion.trc.xml* y del

modelo *PowerDistortion.mdl.xml* del sistema bajo estudio.

Una vez lanzada la GUI se inicia la visualización del contenido de los ficheros de trazas.

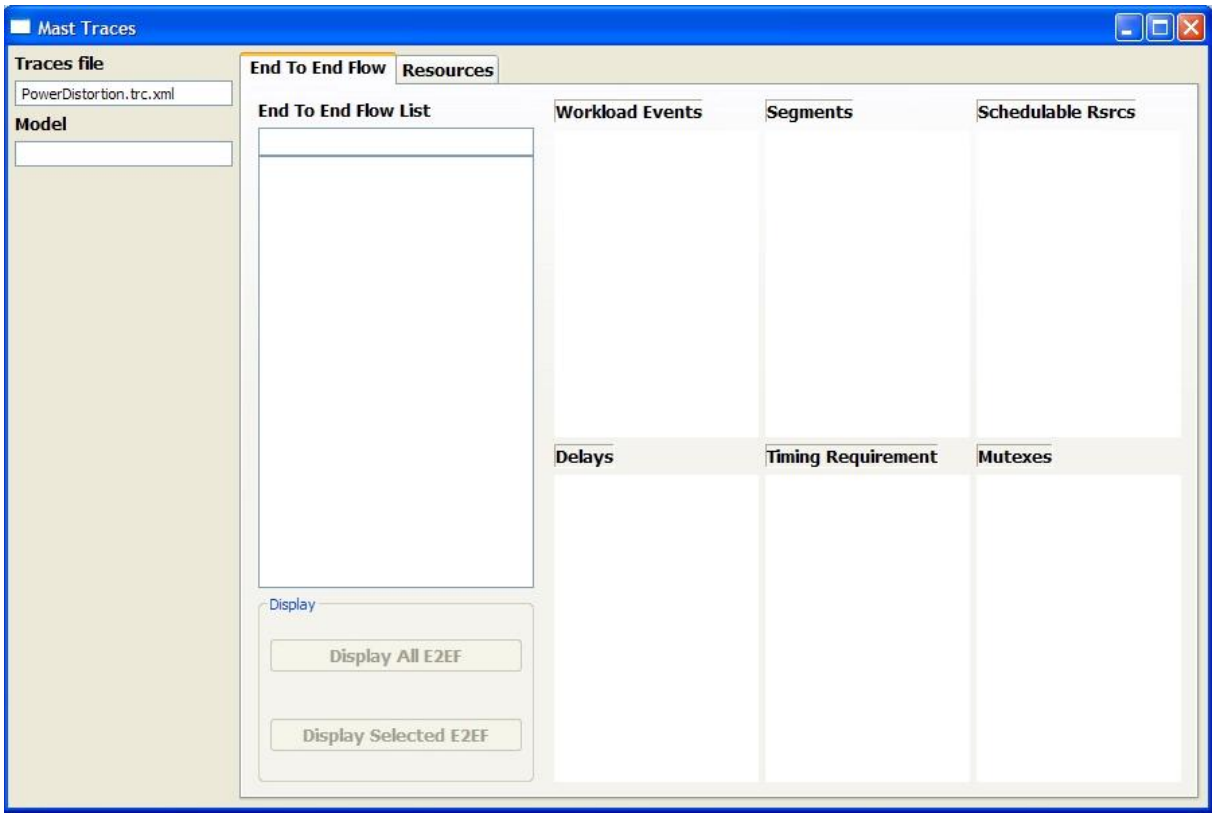

Figura 36. Ventana de control de selección de las trazas a visualizar (GUI).

En el apartado Traces File se cargan los datos del fichero de trazas. En este ejemplo, el directorio donde se encuentran los ficheros de trazas y el modelo `.mdl.xml` será: `C:/Mast2_Project/Mast2_Examples/`.

Se escribe el nombre de un fichero de trazas válido que se encuentre en la carpeta de trabajo, con la terminación `.trc.xml` (en este caso *PowerDistortion.trc.xml*). Una vez rellenemos el campo Traces File, se pulsa la tecla *enter* y la interfaz rellenará los datos Model (con el nombre del fichero de trazas terminado en `.mdl.xml`, en este caso *PowerDistortion.mdl.xml*), la lista `End To End Flow List` y la lista de resursos de procesamiento `Resources List`.También se habilitará la opción `Display All E2EF` y `Display All Resources` que permite visualizar todas las transacciones y todos los recursos, respectivamente, que contiene el fichero de trazas.

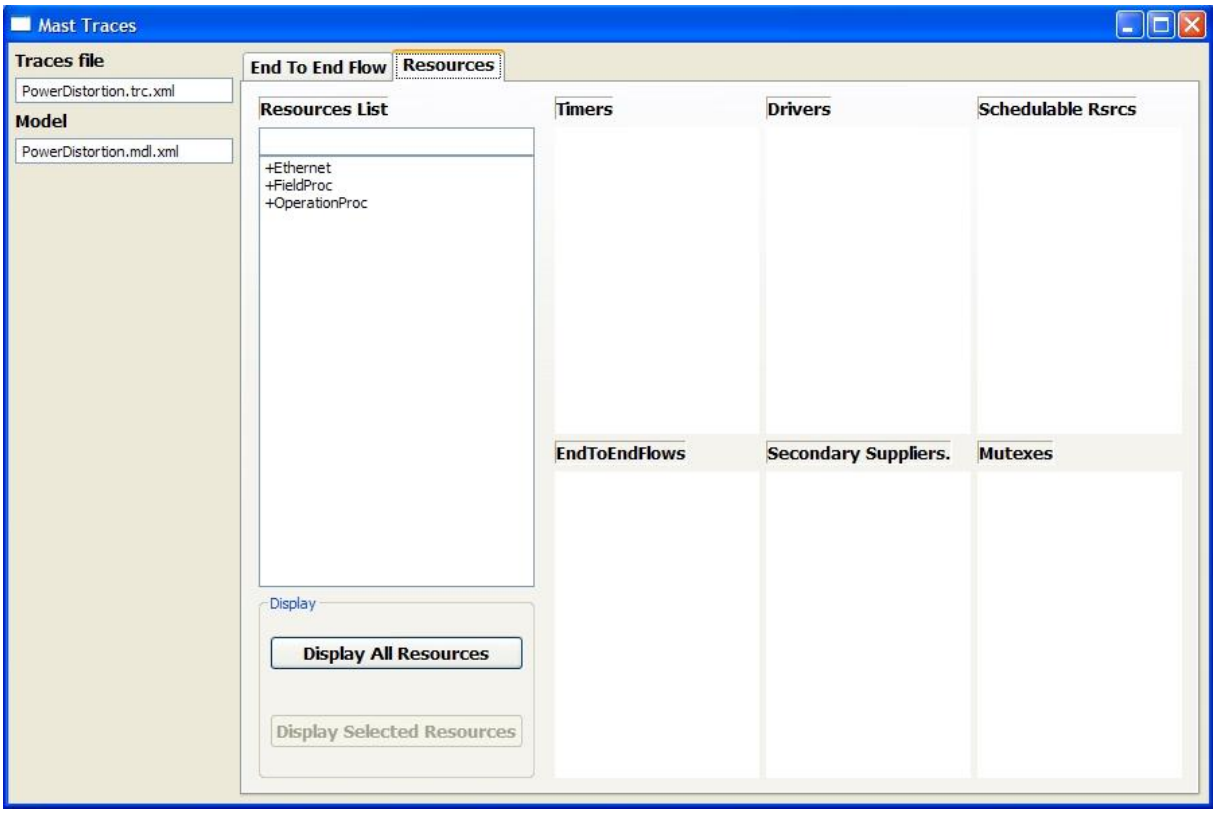

Figura 37. Ventana que muestra la lista de recursos de procesamiento de las trazas a visualizar.

Una vez se ha cargado la lista de transacciones y la lista de recursos de procesamiento, podremos pasar a seleccionar cualquiera de las opciones de ambas listas. Pulsando sobre cualquiera de ellas se completarán las listas de elementos asociados que contiene, y quedará habilitado el botón *Display Selected E2EF* para las transacciones y el botón *Display Selected Resources* para los recursos. Este botón nos permitirá visualizar únicamente la transacción o el recurso de procesamiento seleccionado con todos sus elementos asociados.

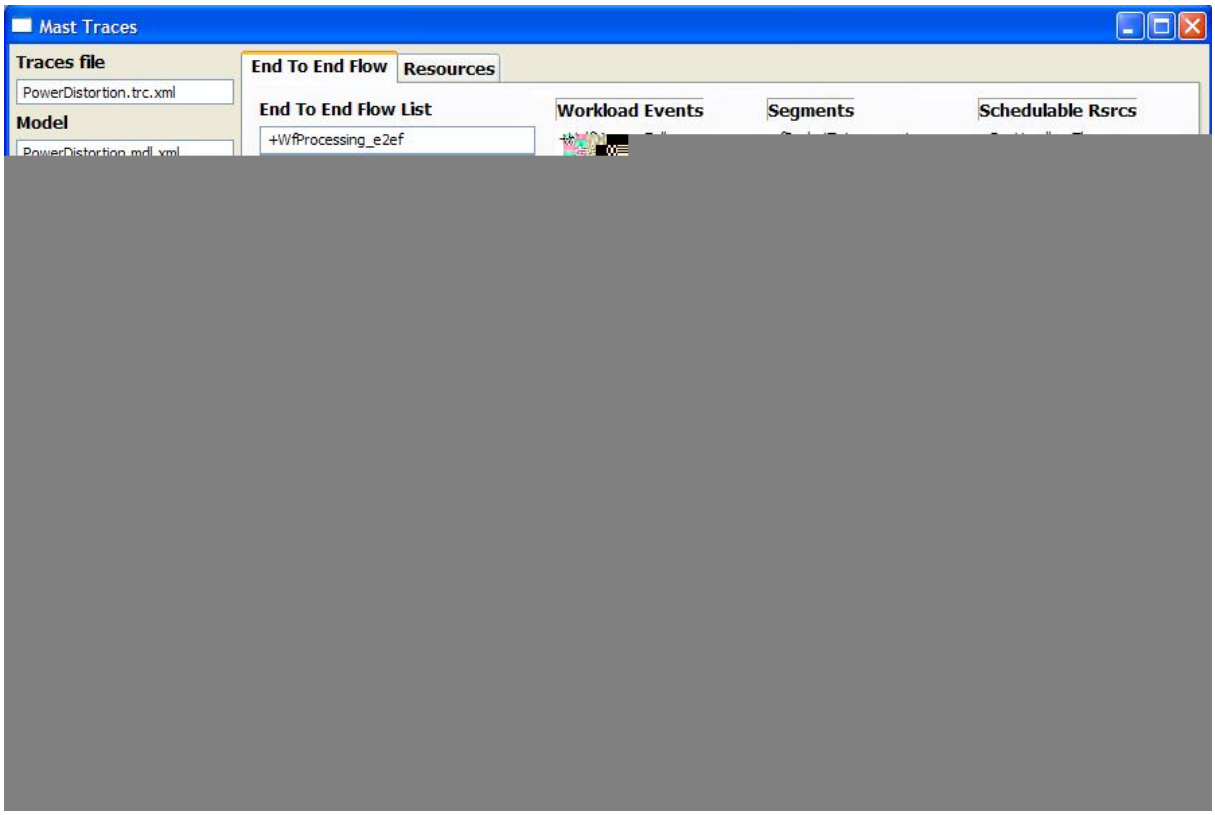

Figura 38. Selección de la transacción, WfProcessing_2e2f, y sus elementos asociados

En este ejemplo, vamos a seleccionar la transacción `WfProcessing_e2ef` de la lista de transacciones y visualizarla. Se mostrarán la transacción `WfProcessing_e2ef` junto con sus *schedulable resources* (`DacHandler_Thr`, `WfBufferServer_Thr`, `WaveformPacket_Cch`, `DistortionMonitor_Thr`) y la lista de recursos de procesamiento (`Ethernet`, `FieldProc`, `OperationProc`).

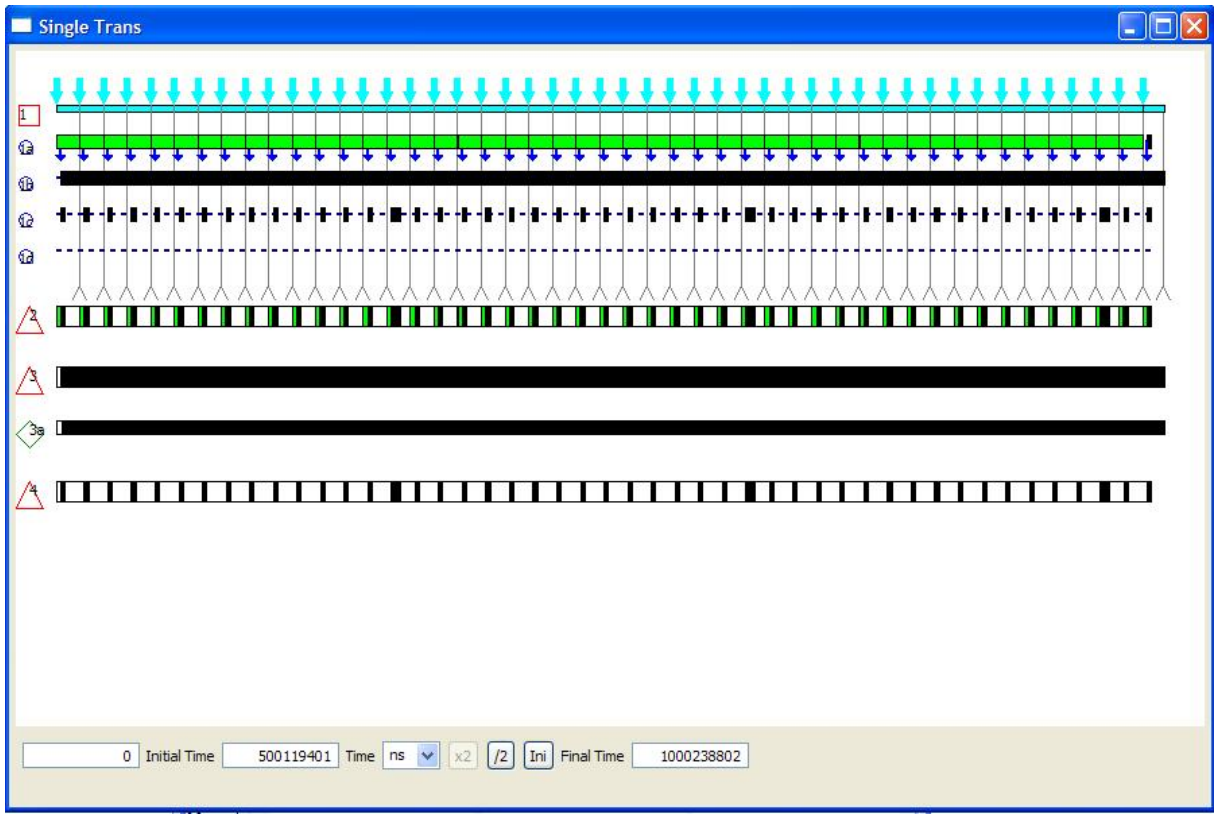

Figura 39. Visualización de la transacción WfProcessing_e2ef junto a los recursos de la traza.

Inicialmente, mostraremos la traza desde el tiempo inicial (`0.0ns`) hasta el final (`1000238802ns`). Para un fichero con una gran cantidad de datos, como este, inicialmente no nos será de gran ayuda por lo que debemos de hacer uso del control de tiempo.

Si deseamos obtener un intervalo con un *zoom* mayor, por ejemplo entre `Initial Time = 490170318ns` y `Final Time = 492123910ns` el resultado obtenido será el mostrado en la siguiente figura:

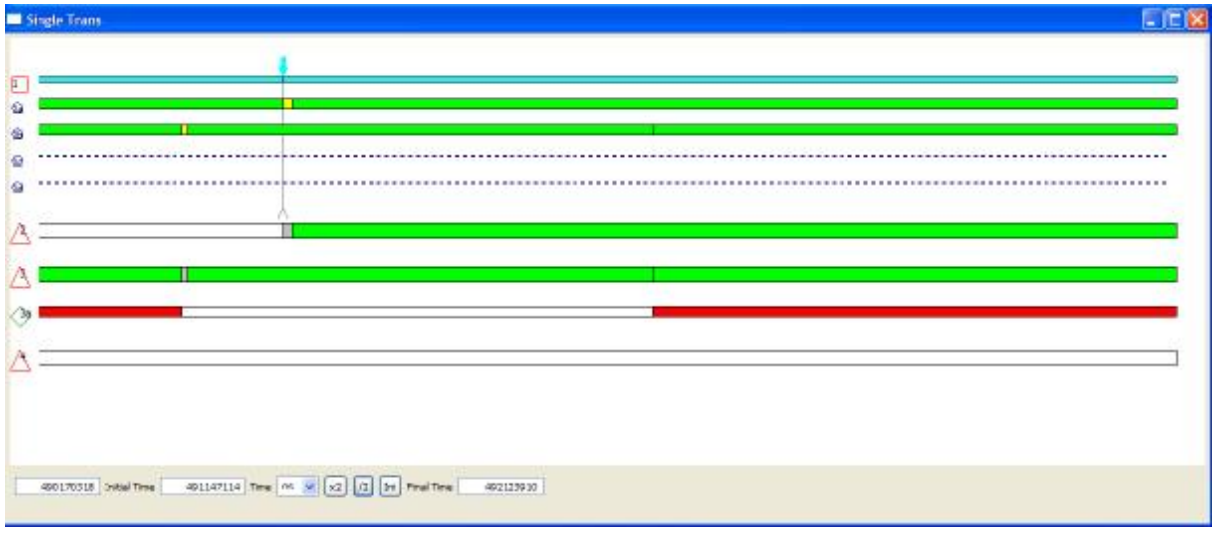

Figura 40. Visualización de la transacción WfProcessing_2e2f, junto a los recursos de la traza en el intervalo de tiempo Initial Time = 490170318ns y Final Time = 492123910ns

5 Conclusiones

En el momento actual, la interfaz gráfica de visualización de trazas está totalmente operativa con toda la funcionalidad que fue prevista en su especificación verificada. Con su uso por los investigadores que desarrollan y diseñan sistemas de tiempo real con el entorno MAST, se validará su diseño y se propondrán los cambios que lo hagan más útil.

La herramienta proporciona una visión gráfica del funcionamiento de cualquier sistema de tiempo real simulado mediante MAST, lo que lo complementa en gran medida. La herramienta permite visualizar las trazar y con ello, poder interpretarlas en base al modelo de tiempo real al que pertenecen.

La validación de su funcionalidad está probada con ficheros de trazas de los cuales conocemos su comportamiento. Comparando su evolución temporal con sus correspondientes ficheros de trazas podemos verificar que lo que visualizamos se corresponde con lo que deberíamos visualizar.

Gracias a la interfaz gráfica podemos acotar la visualización de las trazas en un intervalo de tiempo determinado, ayudándonos a visualizar segmentos aislados que nos permitirán estudiar el sistema en el punto concreto de la traza.

Debido a la complejidad de los ficheros de trazas y que contienen millones de eventos relativos a la actividad del sistema, creemos que esta herramienta, proporciona al operador un herramienta con la que poder interpretarlas.

En esta tesis no sólo hemos desarrollado una herramienta de visualización, sino que también hemos podido explorar la utilización de los recursos XML desde el lenguaje de programación Java y la facilidad para su integración en el entorno MAST desarrollado en Ada'95. Desde este punto de vista, se ha llegado a las siguientes conclusiones:

- El manejo de los recursos XML desde el lenguaje Java es muy intuitivo, y en un tiempo relativamente breve y sin experiencia previa se puede utilizar de forma efectiva.
- La herramienta DOM es muy eficiente para gestionar directamente estructuras de datos con muchos enlaces internos, y por tanto con muchas posibilidades para navegar por ellas. Quizás se le pueda poner como defecto que por su generalidad da lugar a un código poco legible por sí, que requiere un enriquecimiento con comentarios para que pueda ser seguido por terceros.
- La herramienta SAX ha resultado muy eficiente para el manejo de ficheros extensos aún dentro de un entorno interactivo. Los retrasos que introduce en la exploración de los ficheros son razonables.
- La librería gráfica SWT de JAVA nos ha permitido con creces cumplir nuestro objetivo de representar la evolución temporal de las trazas.

Cómo hemos comentado en el apartado 1.3.2, debido a que los ficheros XML son portables por definición y que son los ficheros que se intercambian entre el entorno MAST y la herramienta JAVA no se ha presentado ningún problema en la integración de ambas partes.

Para el futuro, uno de los puntos mejorables con el fin de añadir más funcionalidades a nuestra interfaz gráfica y dotarla de mayor usabilidad, sería poder obtener mayor información de cada elemento que estamos visualizando, como ejemplo, una forma sencilla que no hemos podido abordar debido a la falta de tiempo, podría ser que al pulsar con el ratón sobre un elemento gráfico se muestre la información más relevante de dicho elemento, como puede ser el instante en el que se está dibujando, nombre del elemento,... con el fin de tener toda la información disponible en la ventana de visualización. Actualmente, únicamente mostramos el nombre del elemento sobre el que se pulsa y hemos dejado las bases y la forma de trabajar para implementar esta mejor.

6 Referencias

[1] A. Burns and A. Wellings: “Real-Time Systems and Programming Languages”.Addison Wesley, 1997.
[2] Manuel Collado: “Informática industrial”. Universidad de Castilla-La Mancha, 1997
[3] J. Liu: “Real-Time Systems”. Prentice Hall, 2000.
[4] J. M. Drake, M.González Harbour, J.J Gutierrez y J.C Palencia: “Description of the MAST Model”. http://mast.unican.es/
[5] Java & XML 2nd Edition 2001, por Brett McLaughlin, Editorial O’Reilly & Associates, Inc.
[6] World Wide Web Consortium. Página web: http://www.w3.org/standards/xml/
[7] Comunidad java. Página web: http://www.comunidadjava.org
[8] P. Pacheco: "Interfaz gráfica para la visualización de trazas de sistemas de tiempo real". Tesina de Ingeniería de Telecomunicaciones, Universidad de Cantabria.
[9] P. López Martínez: “Simulador Sim_Mast” página web “http://mast.unican.es/simmast”.
[10] C. Cuevas, P. López Martínez y J. M. Drake: JSimMAST: Simulador de sistemas de tiempo real diseñados con paradigmas avanzados. Página web “ http://www.ctr.unican.es/publications”

7 Anexos

7.3 Mast_Trace.xsd

Plantilla W3C-Schema para el almacenamiento de trazas en forma de documento XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!--*****************************************************************************
       Schema templates for XML Mast Traces File  (Version MAST 1.3.7 )
                              (MAST Project)
        By:
            Patricia Lopez Martinez (lopezpa@unican.es)
            Jose M. Drake Moyano    (drakej@unican.es)
        Grupo de Computadores y Tiempo Real (CTR)
        Universidad de Cantabria
        Santander, 8-Feb-05
*****************************************************************************-->
xs:schema targetNamespace="http://mast.unican.es/xmlmast/trace"
elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:mast_trace="http://mast.unican.es/xmlmast/trace">
    <!-- SIMPLE TYPES -->
   <xs:simpleType name="Date_Time">
       <xs:restriction base="xs:dateTime"/>
       <!-- YYYY-MM-DDThh:mm:ss format -->
   </xs:simpleType>
   <xs:simpleType name="Time">
       <xs:restriction base="xs:float">
           <xs:minInclusive value="0.0"/>
       </xs:restriction>
       <!-- Time interval quantity type -->
   </xs:simpleType>
   <xs:simpleType name="Identifier">
       <xs:restriction base="xs:NCName">
           <xs:pattern value="([a-z]|[A-Z])([a-z]|[A-Z]|[0-9]|.|_)*"/>
       </xs:restriction>
       <!-- Identify a object univocally:begin with a letter, followed by
letters, digits, underscores ('_') or                    periods ('.') -->
   </xs:simpleType>
    <xs:simpleType name="External_Reference">
         <xs:restriction base="xs:NCName">
              <xs:pattern value="([a-z]|[A-Z])([a-z]|[A-Z]|[0-9]|.|_)*"/>
         </xs:restriction>
         <!-- References an external object: begin with a letter, followed by
letters, digits, underscores ('_') or              periods ('.') -->
    </xs:simpleType>
    <xs:simpleType name="Source_Identifier">
         <xs:restriction base="xs:int"/>
         <!-- Key integer (positive, negative,o zero) identifier of a model
object that is able to generate trace messages. -->
    </xs:simpleType>
    <xs:simpleType name="Message_Identifier">
         <xs:restriction base="xs:nonNegativeInteger"/>
         <!-- Non negative integer (positive o zero) thas is an univoval
identifier of a message type -->
    </xs:simpleType>
    <!-- MESSAGE TYPE LIST -->
    <xs:complexType name="Message_Type">
         <xs:attribute name="Mid" type="mast_trace:Message_Identifier"
use="required"/>
         <xs:attribute name="Type" type="xs:string" use="optional"/>
         <!-- Link an explanatory text (type)with a message type (Mid).-->
    </xs:complexType>
    <xs:complexType name="Msg_Type_List">
         <xs:sequence>
            <xs:element name="Msg_Type" type="mast_trace:Message_Type"
                                          maxOccurs="unbounded"/>
         </xs:sequence>
         <!-- List of message types that are referenced in the traces file-->
    </xs:complexType>
```

```
    <!-- MESSAGE LIST -->
    <xs:complexType name="Source">
         <xs:attribute name="Sid" type="mast_trace:Source_Identifier"
use="required"/>
         <xs:attribute name="Name" type="mast_trace:External_Reference"
use="optional"/>
         <xs:attribute name="Type" type="xs:string" use="optional"/>
         <!-- Data linked to a model object that is able to generate messages:
              (Sid)Source key number, (Name) Source identifier in the model and
            (Type)
              type of object model -->
    </xs:complexType>
    <xs:complexType name="Src_List">
         <xs:sequence>
            <xs:element name="Src" type="mast_trace:Source"
            maxOccurs="unbounded"/>
         </xs:sequence>
         <!-- List of model object referenced as a source of messages -->
    </xs:complexType>
    <!-- MESSAGE LIST -->
    <xs:complexType name="Message">
         <xs:attribute name="T" type="mast_trace:Time" use="required"/>
         <xs:attribute name="S" type="mast_trace:Source_Identifier"
use="required"/>
         <xs:attribute name="M" type="mast_trace:Message_Identifier"
use="required"/>
         <!-- Data linked to a message instance:(Time) Ocurrence time, (Sid)
Source of message,(Mid) Mesage type -->
    </xs:complexType>
    <xs:complexType name="Msg_List">
         <xs:sequence>
            <xs:element name="Msg" type="mast_trace:Message"
            maxOccurs="unbounded"/>
         </xs:sequence>
         <!-- List of messages included in a traces file -->
    </xs:complexType>
    <!-- TRACE FILE (root element) -->
    <xs:element name="TRACE_FILE">
         <!-- Data included in a traces file: -->
         <xs:complexType>
            <xs:sequence>
               <!-- Declared message type list -->
               <xs:element name="Msg_Type_List" type="mast_trace:Msg_Type_List"
                                minOccurs="0"/>
               <!-- Declared model object as message source -->
               <xs:element name="Src_List" type="mast_trace:Src_List"
            minOccurs="0"/>
               <!-- Message list of the traces register -->
               <xs:element name="Msg_List" type="mast_trace:Msg_List"/>
            </xs:sequence>
            <xs:attribute name="Model_Name" type="mast_trace:Identifier"
                     use="required"/>
               <xs:attribute name="Model_Date" type="mast_trace:Date_Time"
                     use="required"/>
               <xs:attribute name="Generation_Tool" type="xs:string"
                     use="required"/>
               <xs:attribute name="Generation_Profile" type="xs:string"
                     use="optional"/>
               <xs:attribute name="Generation_Date" type="mast_trace:Date_Time"
                     use="required"/>
               <xs:attribute name="Start_Time" type="mast_trace:Time"
            use="required"/>
               <xs:attribute name="End_Time" type="mast_trace:Time"
            use="required"/>
               <!-- Real-time situation description -->
               <!-- Name of the of the real-time situation model that has
            generated the trace -->
               <!-- Date of last modification of the analyses real-time
            situation model -->
               <!-- Text representing the name of the tool that generated the
            traces -->
               <!-- Text representing the command and options used to invoke
            the tool for the generation of the traces -->
               <!-- Date of generation of traces -->
               <!-- Init time of the trace generation -->
               <!-- End time of the trace generation -->
```

```
        </xs:complexType>
    </xs:element>
</xs:schema>
```