Sistemas Informáticos
Curso 2003-2004

EDROOM TRACER: Trazador a nivel de diseño gráfico para SISTEMAS DE TIEMPO REAL desarrollados con EDROOM

Lucas Gut Galán
Luis Herrero Buitrago
Tomás Morales Mendoza

Dirigido por:
Prof. Jesús Manuel de la Cruz García
Oscar Rodríguez Polo

Dpto. Arquitectura de Computadoras y Automática

Facultad de Informática
Universidad Complutense de Madrid

Índice

Autorización

Se autoriza a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Firmado:

Lucas Gut Galán Luis Herrero Buitrago Tomás Morales Mendoza

Objetivo

Crear una herramienta que soporte la traza de la ejecución de un sistema software de tiempo real diseñado y generado automáticamente con EDROOM [1] y [2] .

Para ello se quiere crear una aplicación que, empleando los mismos elementos gráficos de diseño que utiliza EDROOM , permita seguir la evolución del sistema durante su ejecución, resaltando tanto el estado en el que se encuentra, como los eventos que disparan un cambio de estado. Para facilitar esta tarea la nueva herramienta permitirá establecer puntos de ruptura (breakpoints) asociados a cualquier tipo de evento y proporcionara facilidades como el trazado en modo continuo, en modo paso a paso, el retroceso a estados previos, etc.

Resumen

En este proyecto se ha diseñado un trazador, implementado en JAVA, para sistemas de tiempo real desarrollados con EDROOM. La aportación original consiste en que el trazado se va a realizar a un nivel gráfico y no a un nivel de código fuente. El trazador será capaz de trabajar a partir de una descripción de un modelo de sistema diseñado con EDROOM y mostrar gráficamente cómo es la evolución de su estado en función de los eventos disparados durante su ejecución. La solución se basa en que una secuencia de eventos puede ser generado automáticamente, guardándolo en un archivo de traza, y que posteriormente pueda ser usado por el trazador. Por lo tanto, el propósito del trazador consiste en simular un modelo EDROOM según la secuencia de eventos definida en el archivo de traza.

El trazado es una técnica para la depuración de sistemas en tiempo real ya que los depuradores tradicionales no son capaces de congelar el estado de los dispositivos que interrumpen y a la frecuencia a la que interrumpen. Esta técnica se usa frecuentemente en los emuladores hardware.

In this project, a tracer for real-time systems developed with EDROOM, and implemented in JAVA, is being presented. The main advantages are that the tracing process will be done at a graphical design level and not at a source code level. The tracer will be able to work from a description of a system model designed with EDROOM and show graphically the state evolution determined by the ordering of execution events. The main idea is that such sequences of events may be generated automatically, stored in trace files, and then processed by the tracer. Thus, the purpose of the tracer is to simulate an EDROOM model using such a trace file as an input.

Tracing is a technique used to debug real time systems, as traditional debuggers are usually not able to freeze the system's state when operating at higher frequencies. This technique is often used on hardware emulators.

Palabras Clave

Lista de palabras clave para su búsqueda bibliográfica:

ROOM

EDROOM

Trazador

Sistemas de Tiempo Real

Introducción

EDROOM es una herramienta libre de diseño gráfico y generación automática de código C++ para Sistemas de Tiempo Real basados en ROOM [3]. EDROOM corre bajo Windows pero el código que genera permite trabajar con diferentes plataformas soportadas por varios Kernels de Tiempo Real como RTKernel [4], CMX [5], VxWorks [6], RTAI [7]. EDROOM resulta una herramienta ideal para crear aplicaciones multiplataforma. Tanto ROOM y EDROOM se estudiarán con más detalle en los capítulos 1 y 2 respectivamente.

Como todo entorno de desarrollo, EDROOM debe facilitar una manera de depurar los programas desarrollados. Si bien siempre se pueden emplear depuradores tradicionales que den soporte al lenguaje C++, para los sistemas de tiempo real éstos tienen el problema de que generalmente no resulta factible congelar el estado de los dispositivos que interrumpen y la frecuencia a la que interrumpen (generalmente mayor que 1 Hz) hace complicado otro tipo de control. No existe, por lo tanto, una forma sencilla de efectuar su depuración. Una técnica empleada con éxito para la depuración de los sistemas de tiempo real es la del **Trazado.** Esta técnica se utiliza frecuentemente en los llamados emuladores hardware ya que permiten almacenar toda la información de los buses del sistema y reconstruir después todos los detalles de la ejecución, incluso en el lenguaje fuente.

En este proyecto se ha diseñado un trazador implementado en JAVA para los sistemas desarrollados con EDROOM. La aportación original consiste en que el trazado se va a realizar a nivel de diseño gráfico y no a nivel de código fuente. El trazador será capaz de trabajar a partir de una descripción de un modelo de sistema diseñado con EDROOM y mostrar gráficamente cómo es la evolución de su estado en función de los eventos disparados durante su ejecución. La solución se basa por tanto en el almacenamiento de la información relevante a estos eventos durante la ejecución del sistema que permita después trazar su ejecución a nivel de diseño.

Esta solución presenta un problema de intrusión puesto que se debe insertar código extra en el sistema de tiempo real para el guardado de la información. Sin embargo, a pesar de eso, resulta una herramienta de gran utilidad en las etapas iniciales del proyecto, pudiéndose después refinar la depuración del sistema con otras herramientas tales como los mencionados trazadores integrados en los emuladores hardware.

JAVA es un lenguaje de programación orientado a objetos desarrollado por Sun [8]. Una de sus principales características es su portabilidad entre plataformas, funciona tanto en sistemas Windows como en sistemas basados en Unix (Linux, Solaris, ...).

Todas las herramientas utilizadas en este proyecto son gratuitas y los resultados conseguidos están bajo licencia GPL [11], tanto el programa EDROOM TRACER como el portal web desarrollado para contener todo el material desarrollado.

Descripción del Proyecto

Crear una herramienta para trazar la ejecución de un sistema de tiempo real diseñado con EDROOM y generado automáticamente.

Para ello se quiere crear una aplicación que represente gráficamente los estados de cada actor del sistema de tiempo real (comportamiento) y que vaya mostrando las transiciones producidas al recibir mensajes cada uno de estos actores. Estos mensajes estarán contenidos en un fichero de trazas que también se genera automáticamente.

También se quiere mostrar gráficamente la estructura de cada actor, así como sus conexiones con otros actores.

Para facilitar la labor de desarrollo se permitirá establecer puntos de ruptura (breakpoints) en cualquier estado o transición del comportamiento del actor así como en cualquier puerto o conexión de su estructura.

FUNCIONES

- Carga dinámica de modelos: Compilación de cualquier modelo ya sea desde el disco duro o de una url. Dado un modelo para trazar, se puede compilar y cargar. En el programa se ve el árbol del modelo y luego se puede crear/asignar grupos para trazar.

- Representación gráfica del comportamiento de los actores: Correspondiente al modelo de ROOM. Desarrollo de la representación de componentes, puertos y conexiones.

- Representación gráfica de la estructura de los actores: Correspondiente al modelo de ROOM. Desarrollo de la representación de estados, transiciones, etc.

- Creación de grupos que engloben a diversos actores: Para organizar el proceso de trazado del modelo en cuestión.

- Ejecución hacia delante paso a paso: Lectura de un mensaje de traza en cada ejecución.

- Ejecución continua: Lectura consecutiva de mensajes de traza en la ejecución.

- Ejecución hacia atrás paso a paso: Muestra el estado del sistema hasta procesar el mensaje anterior al actualmente leído. Esto es otra novedad respecto a los depuradores tradicionales ya que éstos solo permiten controlar la ejecución hacia delante. Con este trazador se pretende retroceder un paso al actualmente dado para ver el estado en que se

encontraba el sistema, y así evitar volver a ejecutar otra vez todos los mensajes.

- Ejecución hasta siguiente breakpoints: Permite situar breakpoints tanto en estados y transiciones del comportamiento como en puertos y conexiones de la estructura, para poder ver el estado en que se queda el sistema después del procesamiento de los mensajes leídos y así poder estudiarlo.

- Ejecución hacia atrás, buscando el breakpoint anterior: Muestra el estado del sistema hasta alcanzar el breakpoint previo más cercano al estado en el que está el sistema con la lectura del último mensaje de traza. Es decir, se presente regresar a un estado anterior al actual localizado con un breakpoint en algún elemento del modelo. Esto no es normal que se pueda realizar en la depuración de un lenguaje de programación, ya que los depuradores que se tienen solo permiten poner puntos de ruptura que se alcanzan con la ejecución hacia delante del programa. Por tanto, esto es una novedad que ayuda al desarrollo y depuración de sistemas en tiempo real.

- Reinicialización del trazado de un modelo: Se comienza a leer el fichero de trazas desde el inicio y el estado de todos los actores se reinicia. De este modo se puede volver a trabajar desde el principio con el sistema cargado reiniciado.

- Configuración de la velocidad de ejecución: Cambiar el valor del tiempo de visualización de la transición entre estados. Contra mayor sea el tiempo de delay, más tiempo tarda en ejecutarse la lectura de un mensaje, es decir, las transiciones entre las iluminaciones de los elementos es más lenta. Si el tiempo disminuye, ocurre todo lo contrario, las transiciones de las iluminaciones es más rápido. Esto se configura según la máquina o según el gusto de detalle que se desee ver.

- Información sobre la memoria utilizada en cada momento por la aplicación: Mostrar el consumo de memoria por parte de la máquina virtual de JAVA para el trazado del modelo cargado. Esto ayuda a conocer el consumo de memoria que acarrea la carga del modelo.

- Salvar/Cargar proyectos de traza: Almacenar y recuperar proyectos de traza que contienen la asignación de grupos de los actores en formato XML. Esto ayuda a su transmisión y control ya que XML es un estandar y

- Multiplataforma: La aplicación se ejecuta sobre cualquier sistema operativo con el software de JAVA instalado (Windows XX, Linux/Unix, ...). De este modo se puede trabajar con toda tranquilidad, ya que el trabajo realizado en una máquina (cambios en el código del trazador, modelo, etc..) seguirán funcionando en cualquier otro puesto.

PASOS SEGUIDOS

Se ha seguido un ciclo de vida de ingeniería del software iterativo e incremental que se ha desarrollado en las siguientes etapas:

1ª Iteración

- Realizar la representación gráfica del comportamiento de los actores.
- Realizar la creación de grupos y la asignación de actores a grupos.
- Realizar la ejecución paso a paso.

2ª Iteración

- Depuración de los pasos anteriores.
- Realizar la ejecución mediante breakpoints.

3ª Iteración

- Depuración de los pasos anteriores.
- Realizar la carga dinámica de modelos.
- Realizar la ejecución hacia atrás, paso a paso y hacia el breakpoint anterior.

4ª Iteración

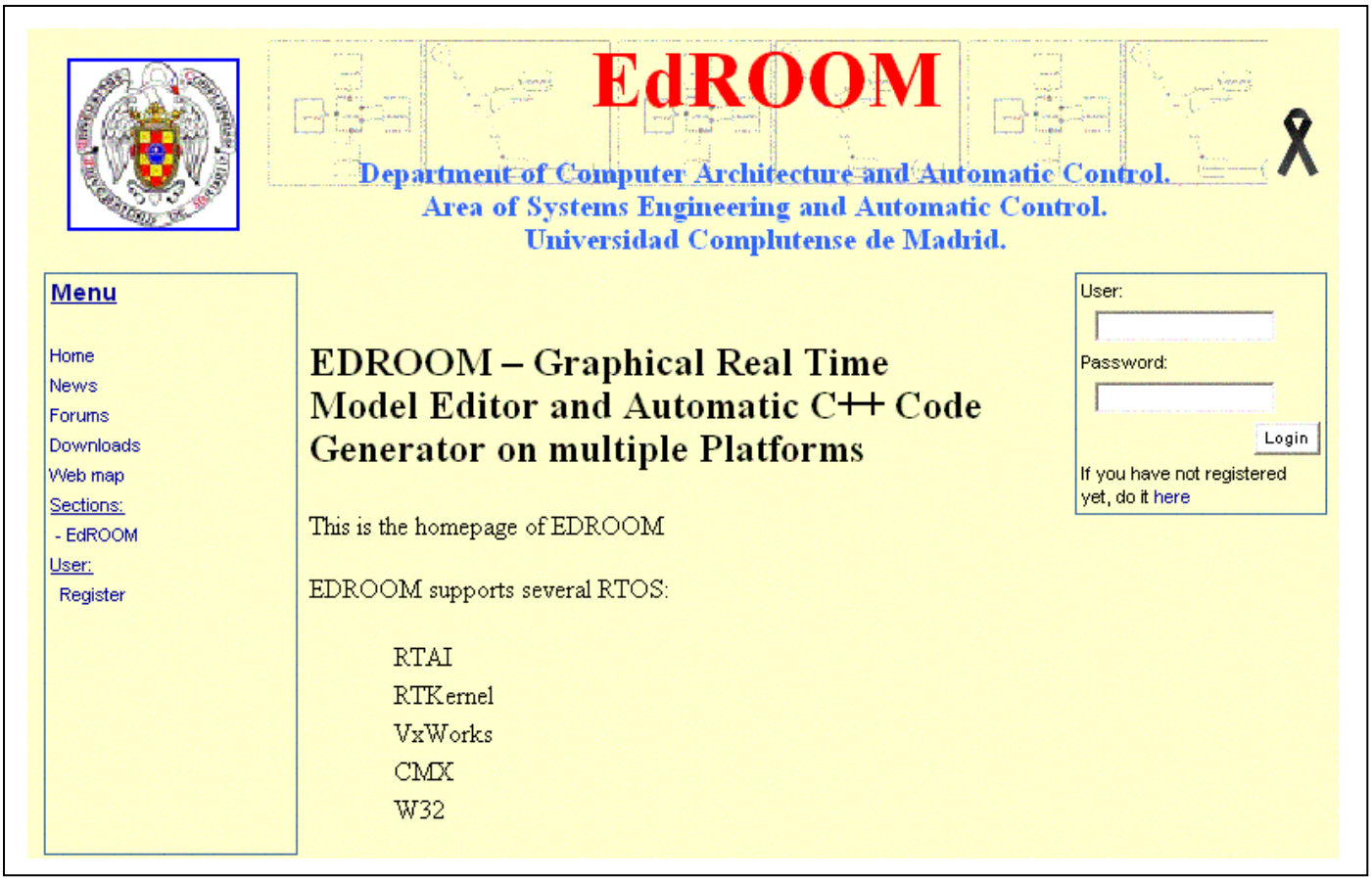
- Depuración de los pasos anteriores.
- Realizar la ejecución continua.
- Realizar la representación gráfica de la estructura de los actores.

5ª Iteración

- Depuración de los pasos anteriores.
- Optimizaciones.
- Fase de pruebas.

PORTAL WEB

Además para recopilar todo el desarrollo del proyecto se ha creado un portal web para dar acceso a EDROOM y los Sistemas en Tiempo Real a toda la comunidad académica. En la página web se colgará el resultado del presente proyecto y futuros trabajos. Además se ha añadido secciones de foro y noticias para los miembros muestren sus resultados o hagan sus preguntas.



El portal web se ha desarrollado con la tecnología JSP desarrollado por Sun [8] para la creación de contenidos web dinámicos mezclando HTML y JAVA. Para el almacenaje de información se ha empleado la base de datos MySQL [10] que es de libre distribución.

El portal contiene todos los servicios que cualquier otro portal. El acceso a archivos y documentación sobre EDROOM TRACER es libre para cualquier persona que entre. Se ha desarrollado un sistema de gestión de usuarios para las secciones de foro y noticias. El portal también da servicio al webmaster para que pueda subir cualquier material a la web. Asimismo, el webmaster podrá crear nuevas secciones dentro del portal de una forma intuitiva mediante formularios. De la misma forma, el webmaster podrá gestionar las distintas secciones, pudiendo eliminar noticias, foros, descargas de una sección o secciones completas a través de formularios.

Este portal está contenido en un servidor de la Facultad de Informática y se accede desde la siguiente dirección:

http://isawin.dacya.ucm.es/edroom

El código del portal se encuentra bajo licencia GPL [11] para su libre distribución y de igual modo, las herramientas empleadas para su desarrollo: Apache Tomcat [13] y MySQL [10].

Capítulo 1

ROOM: Modelado de Sistemas en Tiempo Real

En este capítulo se describen los elementos empleados en la construcción de los modelos de los sistemas en tiempo real con los cuales trabaja la herramienta EDROOM. Dicho modelado está basado en ROOM y permite definir gráficamente, y bajo el paradigma de objetos, los elementos de especificación del sistema tales como las tareas que comprende, topologías de comunicación que se emplean y la definición de su comportamiento reactivo. ROOM permite definir en varios niveles y de una manera jerárquica tanto la estructura de tareas como el comportamiento de las mismas. La definición del comportamiento de forma reactiva hace que el modelo sea orientado a eventos mientras que la comunicación exclusiva a través de puertos proporciona una encapsulación que facilita el desarrollo basado en componentes.

1 ROOM: Modelado de sistemas en tiempo real

ROOM (Real-Time Object Oriented Modelling), presentado por Selic, Gulleckson y Ward en 1994 [3], define un lenguaje de modelado que intrínsecamente lleva asociada una metodología orientada a objetos para el desarrollo de sistemas en tiempo real. La herramienta EDROOM incorpora un editor gráfico de modelos y un generador de código para obtener de forma automática su implementación en C++.

1.1 Requisitos del diseño de sistemas de tiempo real

Definimos un sistema de tiempo real como un conjunto de elementos que interactúan permanentemente con su entorno. Muchos de los eventos que reciben no son de aparición periódica y su respuesta es solo válida si se realiza en un intervalo de tiempo (fuera de él, la respuesta no es válida o es ignorada). Además es muy normal que la atención a los distintos eventos deba realizarse según un orden de prioridad, lo que hace necesario trabajar en un entorno concurrente.

Según estos requisitos la consecución del diseño de los sistemas de tiempo real debe dar como resultado los siguientes elementos:

- La determinación del conjunto de tareas concurrentes que colaboran para realizar el sistema.
- La definición de la comunicación entre ellas, incluyendo su topología y los protocolos válidos.
- La especificación del comportamiento de cada una de las tareas.
- La política de planificación de las tareas.

El lenguaje de modelado ROOM proporciona una manera de especificar gráficamente las tareas, la comunicación que se establece entre ellas y su comportamiento. En cuanto a la política de planificación, ROOM integra su especificación en el diseño detallado del comportamiento.

1.2 Estructura de actores del modelo

En ROOM cada tarea está asociada a un actor. Un modelo de un sistema define el conjunto de actores que colaboran para realizarlo. La tarea correspondiente a cada actor se ejecuta concurrentemente con el resto.

Los actores de un modelo se organizan de manera estructurada en varios niveles (la organización de cada nivel permanece oculta a los niveles superiores). Para ello ROOM posibilita que un actor pueda contener en su interior otros actores. En todos los sistemas modelados con EDROOM va a existir siempre un actor principal que contiene, en varios niveles, al resto de los actores del sistema. La estructura de los modelos es, por tanto, jerárquica, facilitando así la encapsulación y la claridad del diseño. También se consigue modularidad y escalabilidad, al poder cambiar elementos sin transformar el modelo y añadir elementos nuevos para conseguir más funcionalidad o mayor realismo hacia el sistema modelado.

En ROOM los actores, basándose en el paradigma de objetos, son siempre instancia de una clase actor. De esta manera es la declaración de la clase actor principal la que define el primer nivel de la estructura del modelo, como se puede ver en la figura 1-1.

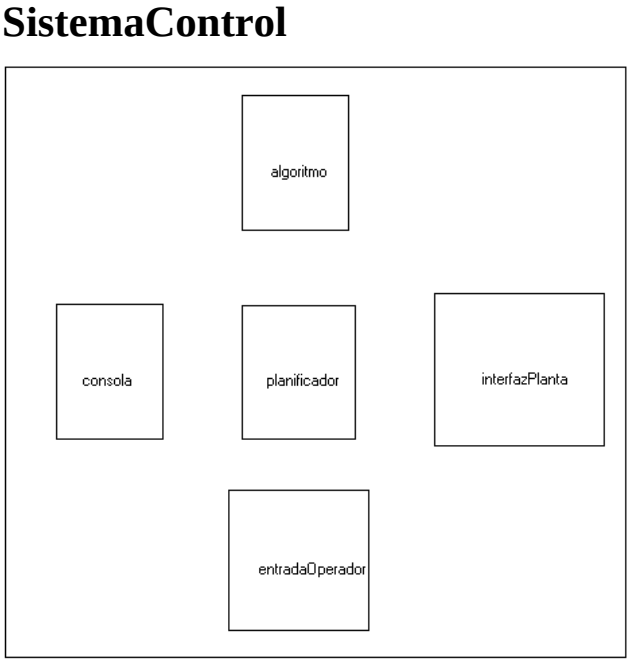


Figura 1-1: Ejemplo de estructura del primer nivel definida por la clase actor principal.

ROOM potencia el rediseño de la estructura debido a su naturaleza gráfica. La representación del sistema mediante una jerarquía de actores facilita la compresión del modelo y simplifica la tarea de añadir, reorganizar o eliminar funciones a la gestión de actores.

1.3 Comunicación. Mensajes, puertos y conexiones

1.3.1 Mensajes

En ROOM los actores se comunican mediante mensajes. Cada mensaje incluye una señal que lo identifica, un dato opcional, y es enviado con una prioridad determinada. Cada actor mantiene una cola con los mensajes recibidos ordenada por prioridad de forma que el actor atiende primeramente aquellos de mayor prioridad. Si dos mensajes tienen la misma prioridad el más antiguo estará primero en la cola. En un entorno multitarea hay un solo procesador y por lo tanto sólo será posible ejecutar un actor en cada momento. ROOM determina que las tareas que implementan los actores deben ajustar su prioridad a la del mensaje de mayor prioridad de los recibidos.

Los mensajes son la base para la posterior generación de archivos de traza, ya que es de aquí donde se consigue la información de lo que ha ocurrido en el sistema para su posterior estudio en el trazador, que reconstruirá la ejecución del sistema. En la traza solo aparecerán los mensajes procesados, porque un actor puede recibir muchos mensajes pero que ignora porque tiene otros de mayor prioridad.

1.3.2 Tipos de puertos

Cada puerto, independientemente de ser conjugado o no, puede ser de uno de estos tres tipos: externo, interno o "relay". Cada uno de los tipos representa una forma de comunicación distinta entre actores. Los puertos externos permiten establecer comunicación entre actores componentes situados al mismo nivel. Los puertos internos permiten comunicar un actor componente con el actor que lo contiene. Los puertos "relay" permiten exportar el puerto de un actor componente a la interfaz del actor que lo contiene. Los puertos externos y los "relay" forman la interfaz externa del actor mientras que los puertos internos forman la interfaz interna.

Los actores no pueden enviar mensajes a través de sus puertos "relay" ya que en realidad su gestión corresponde al actor componente. Además, los mensajes recibidos a través de los puertos "relay" son automáticamente redirigidos al actor componente a través del puerto correspondiente sin que puedan ser tratados en el actor contenedor de ninguna manera.

Los puertos externos sí son accesibles directamente por los actores y a través de ellos los actores pueden enviar y recibir mensajes.
Los puertos internos se distinguen fácilmente del resto porque se localizan fuera del borde del actor. No forman parte de la interfaz externa del actor por lo que la comunicación a través de ellos queda totalmente oculta a los clientes de la clase actor.

1.3.3 Conexiones

Las conexiones hacen efectiva la comunicación entre los actores a través de sus puertos. Además también se emplean para exportar un puerto de un actor componente hacia un puerto "relay". En ROOM cada puerto sólo puede establecer una conexión.

En resumen, la comunicación entre los actores se determina mediante la definición de las clases protocolo y la inclusión en las clases actor de puertos externos, internos o "relay" que determinan estrictamente su interfaz. Las conexiones entre los puertos determinan cuál es el actor destino de cada mensaje enviado desde el actor origen.

1.4 Comportamiento. Diagrama de estados

Para definir el comportamiento de un modelo ROOM cada clase actor del modelo cuenta con un diagrama de estados de uno o varios niveles, cuyas transiciones están disparadas por la llegada de mensajes. El comportamiento global del modelo esta determinado por tanto por la cooperación entre los actores que se comunican mediante paso de mensajes a través de sus puertos.

Su representación gráfica determina, a partir del estado actual, cual es la transición que se dispara por la llegada de cada posible mensaje. Las entradas y salidas de los estados, así como las transiciones, pueden tener acciones asociadas que son ejecutadas en los instantes correspondientes. De esta forma el diagrama sirve de marco para determina el tratamiento de cada mensaje según la situación actual del actor. La figura 1-2 muestra un ejemplo de diagrama de estados.

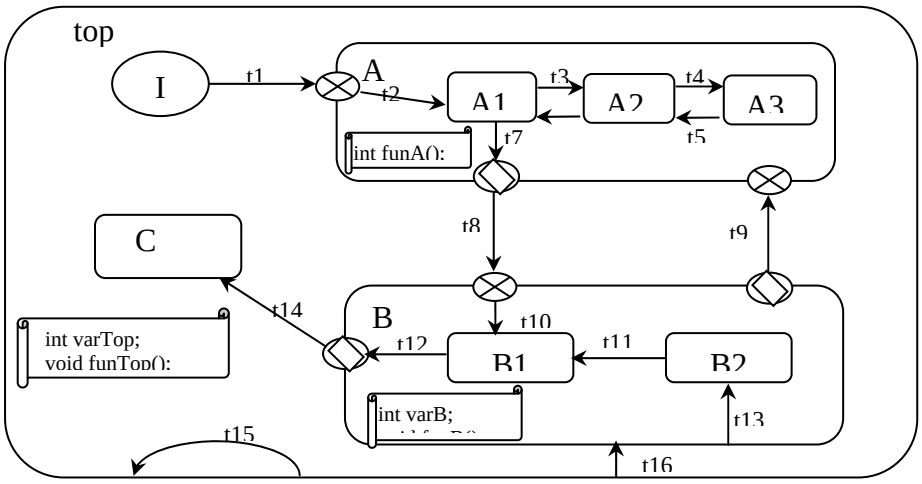


Figura 1-2: Ejemplo de Diagrama de Comportamiento de ROOM

1.4.1 Contextos y estados hoja

Los diagramas incluyen la posibilidad de definir varios niveles dentro del comportamiento. El nivel superior, denominado contexto top, está definido por un conjunto de estados entre los cuales se encuentra el estado inicial I. Todos ellos son subestados del contexto top y, exceptuando el estado inicial, cada uno puede contar a su vez con otros subestados. En ese caso decimos que definen un contexto propio situado un nivel por debajo del contexto top. Los estados que no definen un contexto propio se denominan estados hoja; el estado inicial es por lo tanto siempre un estado hoja. La figura 1-4 muestra dos niveles. En él los estados I, A1, A2, A3, B1, B2 y C son estados hoja, mientras que A y B son estados que definen un contexto propio.

El número de niveles al que se puede extender el comportamiento no está limitado. Este hecho proporciona gran flexibilidad ya que siempre es posible refinar el diseño del comportamiento de un actor incluyendo dentro de un estado nuevos subestados y definiendo así un nuevo contexto. Esto es muy útil en el desarrollo de sistemas basados en la evolución de prototipos donde cada prototipo amplía la funcionalidad del anterior.

1.4.2 Acciones que se ejecutan al tratar un mensaje recibido

La dinámica del comportamiento de un actor definida por su diagrama es la siguiente: en todo momento el actor se encuentra en uno de sus estados hoja; un mensaje recibido constituye un evento capaz de disparar una transición; la transición disparada determina cual es el nuevo estado hoja del actor. Las acciones que se ejecutan debido al disparo de una transición son las asociadas a la salida del estado origen, a la transición y a la llegada al estado destino. Si la transición, además, provoca la salida o entrada a uno o más contextos, a las acciones anteriormente citadas habrá que añadir la ejecución de las acciones asociadas a las salidas y entradas de los estados que definen esos contextos. En todo caso el orden de ejecución de las acciones se corresponde con el recorrido que sigue la transición.

1.4.3 Disparo de una transición

Para determinar cuando se dispara una transición cada una de ellas tiene definida la tríada: señal, puerto y guarda. Un mensaje tiene siempre una señal que lo identifica y es recibido a través de un puerto concreto. La guarda es una función booleana que debe devolver "true" para que la transición se dispare. Cuando un mensaje llega a un actor, la implementación de su comportamiento debe de gestionar cual es la transición que se dispara en función del estado hoja actual, la señal del mensaje, el puerto por el que ha sido recibido y la evaluación de las correspondientes guardas. Si la gestión encuentra una transición válida la lleva a cabo, ejecuta las acciones pertinentes y actualiza el nuevo estado hoja del actor. Si no encuentra ninguna, el actor permanece en el estado hoja actual.

1.4.5 Puntos de entrada y puntos de salida

Los puntos de entrada, representados por el símbolo ⊗, marcan la llegada a un contexto desde una transición definida en el contexto superior. En el ejemplo de la figura 1-2 el contexto A contiene dos puntos de entradas correspondientes a las transiciones t1 y t9 definidas en el contexto top.

Por otro lado, los puntos de salida, representados por el símbolo ◇, se generan debido a la existencia de una transición de salida definida igualmente en el contexto superior. En la figura 1-2 la transición t8 del contexto top genera el correspondiente punto de salida en el contexto A. La figura 1-3 muestra con más detalle la localización de estos dos puntos de entrada y del punto de salida en el borde del contexto A.

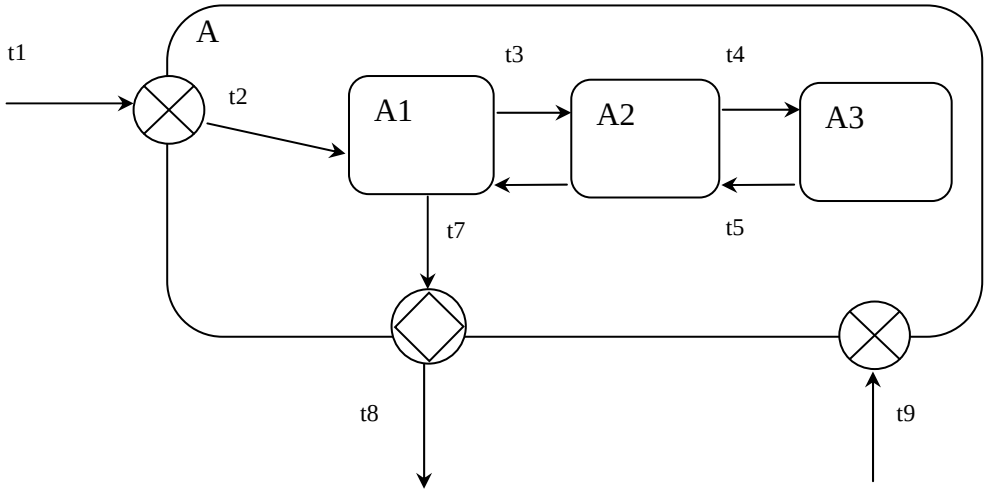


Figura 1-3: Puntos de entrada y salida definidos en un contexto distinto del top.

El punto de entrada correspondiente a t1 es además el origen de la transición t2 definida en A. En realidad t2 es el nombre con el que se identifica a la transición t1 dentro del contexto A y ambas forman una única transición disparada por un sólo evento. El hecho de dividirla en dos transiciones separadas por el punto de entrada permite encapsular completamente cada contexto de forma que se puede asociar una acción a t1, definida en el contexto top, y otra a t2, definida en el contexto A. De igual manera la transición t8 del contexto top y la transición t7 del contexto A son en realidad una única transición con dos etapas enlazadas por el punto de salida.

1.4.6 Transiciones de contexto

Una transición de contexto es una transición que se dispara para todos los subestados de dicho contexto. En la figura 1-2 son transiciones de contexto t9, t13, t15 y t16.

Hay dos formas distintas de definir transiciones de contexto. La primera es mediante un punto de salida que no es destino de una transición. Este es el caso de la transición t9 respecto al contexto B. Esta transición se dispara para cualquiera de sus subestados (B1 y B2) y es por tanto una transición de contexto. La segunda manera es mediante una transición que parte del borde del contexto. Este es el caso de las transiciones t13, t15 y t16 definidas para los contextos B, la primera, y top las dos siguientes. Las transiciones de este tipo pueden tener como destino un estado concreto (t13 y t16) o el propio borde del contexto (t15). Cuando el destino es el borde el estado al que se llega después de la transición es el mismo del que se parte.

La diferencia entre los dos tipos reside en que las que tienen como origen el borde no implican la salida del contexto en el que están definidas mientras que las otras sí.

La dinámica que define el disparo de las transiciones de contexto es la siguiente: cuando el actor se encuentra en un estado hoja y llega un mensaje, primero se busca entre las transiciones que salen directamente de él si hay alguna disparada por ese mensaje, de no encontrarla se busca entre las transiciones de contexto definidas en el propio contexto, si la búsqueda tampoco tiene éxito se busca en las transiciones de contexto del contexto superior y así sucesivamente. La búsqueda termina, bien cuando se encuentra una transición que sea disparada por el mensaje, o bien cuando después de buscar en el contexto top no se ha encontrado ninguna transición. En ese caso el mensaje no dispara ninguna transición y el estado actual sigue siendo el mismo. El uso de las transiciones de contexto reduce notablemente el número de transiciones y constituyen un aspecto de la sintaxis gráfica muy potente que permite la creación de diagramas de estados claros, fáciles de manejar y de gran capacidad expresiva.

Capítulo 2

EDROOM: Editor gráfico de modelos y generador automático de código

En este capítulo se presenta la herramienta EDROOM [1] y [2] en la que se integra un editor gráfico de modelos basados en ROOM y un generador automático de código C++. Todo esto permite obtener la implementación de un sistema de tiempo real a partir de su modelo ROOM.

2 EDROOM: Editor gráfico de modelos y generador automático de código

En el capítulo 1 se ha descrito cómo capturar, empleando una sintaxis gráfica, todas las ideas propias del diseño de un sistema software de tiempo real. Este capítulo se va a centrar en describir las características principales de la herramienta EDROOM que son la integración de un editor de modelos basados en ROOM y un generador automático de código. Todo esto permite obtener la implementación de un sistema de tiempo real a partir de su modelo ROOM.

2.1 Edición del modelo gráfico

EDROOM facilita la construcción de la estructura de las clases actor utilizando técnicas basadas en el uso del ratón. La figura 2-1 muestra la pantalla de edición de la estructura de una clase actor.

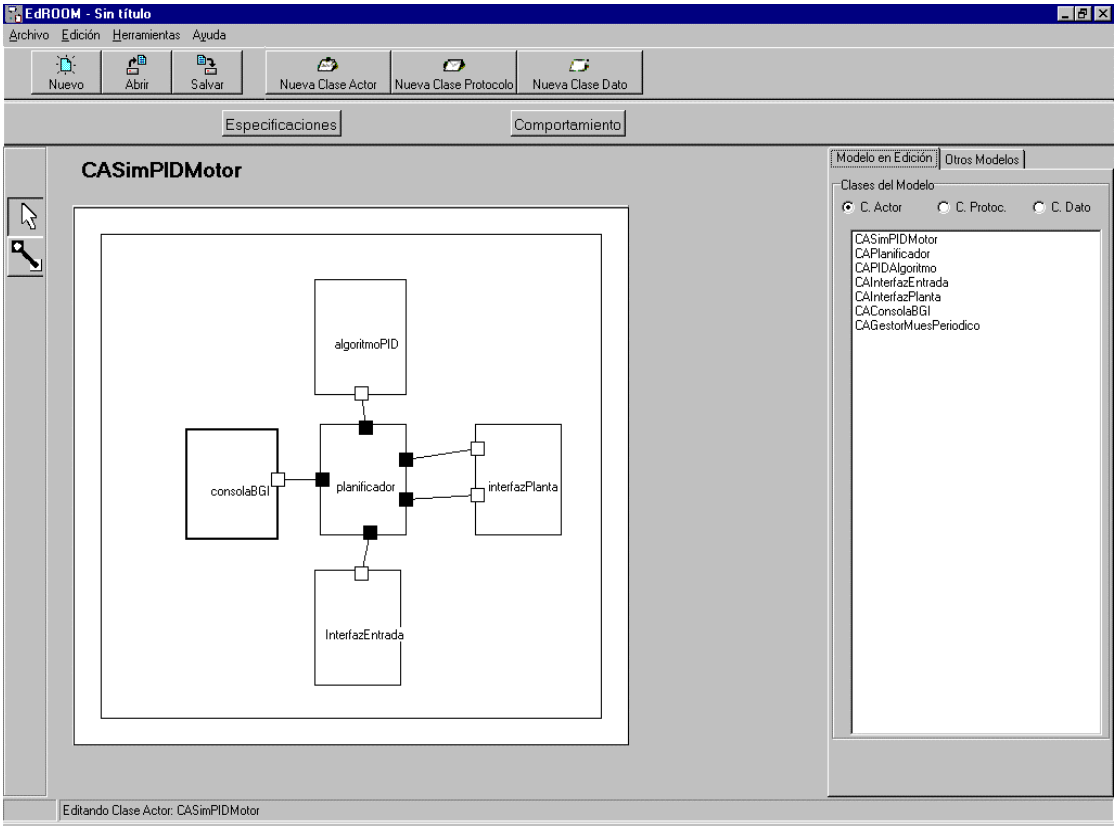
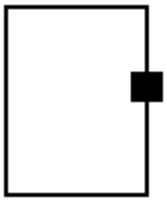


Figura 2-1: Ventana de edición de la estructura de una clase Actor en EdROOM.

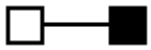
La estructura de un actor puede estar formada por:

Actores: que se representan como elementos rectangulares y se comportan como subactor del cual están contenidos:



Puertos: que se representan como cuadrados que se colocan en los laterales de los actores (ya sea de subactor o actor contenedor).

Conexiones: que se representan como una línea entre dos puertos.



En cuanto al comportamiento, EDROOM también dispone de un interfaz gráfico para editar el diagrama de estados correspondiente a cada clase actor. La figura 2-2 muestra esta ventana de edición. Desde ella se puede añadir los estados, transiciones, puntos de entrada y salida correspondientes al diagrama. Para añadir varios niveles al comportamiento simplemente hay que hacer doble clic sobre el estado al que se le quieren añadir nuevos subestados.

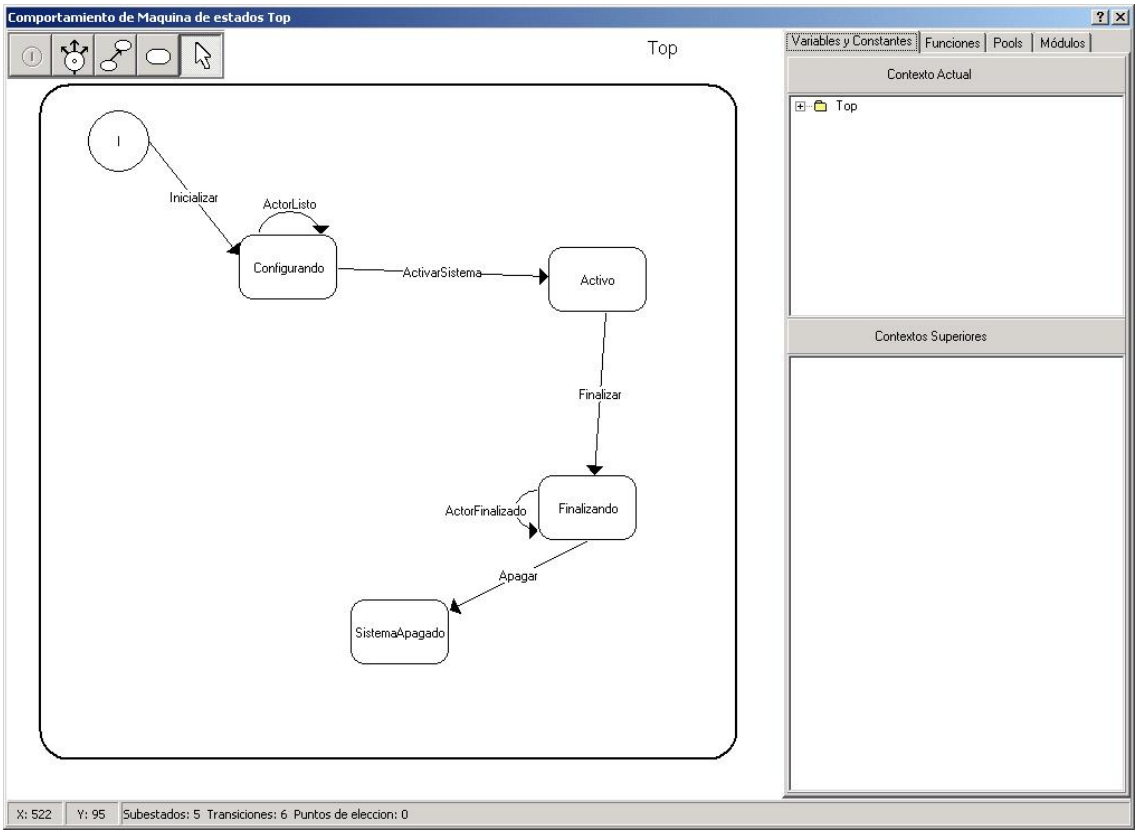
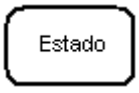


Figura 2-2: Ventana de edición del comportamiento de una clase Actor en EdROOM.

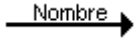
El comportamiento de un actor puede estar formada por:

Subestado Inicial: Representa el estado en el que se encuentra el actor antes de recibir ningún mensaje.

Estado: Representa una situación en el comportamiento del actor



Transición: Representa el paso de un estado a otro (puede ser el mismo) en el comportamiento.



Punto de entrada/salida: Representan las zonas de paso a otro contexto de comportamiento.

2.2 Generación de código

El código generado por EDROOM para cada modelo lo forman un conjunto de ficheros fuente C++. Este código debe compilarse y enlazarse junto con la librería de servicios ROOM para formar el ejecutable que realiza el sistema. Además de generar el código ejecutable, nos proporciona un conjunto de clases JAVA que va a posibilitar el Trazado sin tener que realizar cambio alguno.

Parte de la complejidad de la generación de código viene dado por los elementos gráficos del diseño que deben ser interpretados e implementados correctamente. El cuerpo de las funciones definidas en el comportamiento debe integrarse con el resto del código generado para formar una única entidad. El resultado de este proceso es la generación de una única clase para cada clase actor definida en el modelo.

2.3 Generación de archivo de traza

La ejecución del modelo nos debe dar información sobre lo que está ocurriendo para luego poder comprobar los posibles fallos en el trazador. Esto archivo implica un elemento intrusivo en el sistema porque debe de dar un nuevo servicio para la generación del fichero. Esto es un "mal" necesario para el proceso de trazado, pero que no es una carga pesada para el sistema.

2.4 Prestaciones

EDROOM ha sido empleado en la implementación experimental de la construcción de un sistema anti-balanceo para ferrys [12].

Capítulo 3

EDROOM Tracer: Trazador a nivel de diseño gráfico para Sistemas de Tiempo Real desarrollados con EDROOM

En este capítulo se presenta la herramienta EDROOM Tracer en la que se integra un trazador a nivel gráfico para sistemas de tiempo real desarrollados con EDROOM. Esta herramienta permite seguir la evolución del sistema durante su ejecución, empleando los mismos elementos gráficos de diseño que utiliza EDROOM, resaltando tanto el estado en el que se encuentra, como los eventos que disparan un cambio de estado. Para facilitar esta tarea la nueva herramienta permitirá establecer puntos de ruptura (breakpoints) asociados a cualquier tipo de evento y proporcionara facilidades como el trazado en modo continuo, en modo paso a paso, el retroceso a estados previos, etc.

3 EDROOM Tracer: Trazador a nivel gráfico para Sistemas de Tiempo Real desarrollados con EDROOM

Con EDROOM Tracer se va a depurar un modelo ROOM implementado con EDROOM y en JAVA.

3.1 Entrada de Datos

Los elementos principales de entrada de EDROOM Tracer son el archivo de traza generado automáticamente por el código ejecutable que crea EDROOM y modelo del sistema en JAVA. Además se pueden añadir otro elemento de entrada que es la carga de un proyecto (fig 3-1). Este proyecto almacena los grupos que conforman la traza del modelo y los actores que contiene cada grupo. Este proyecto ha sido creado anteriormente para el mismo modelo.

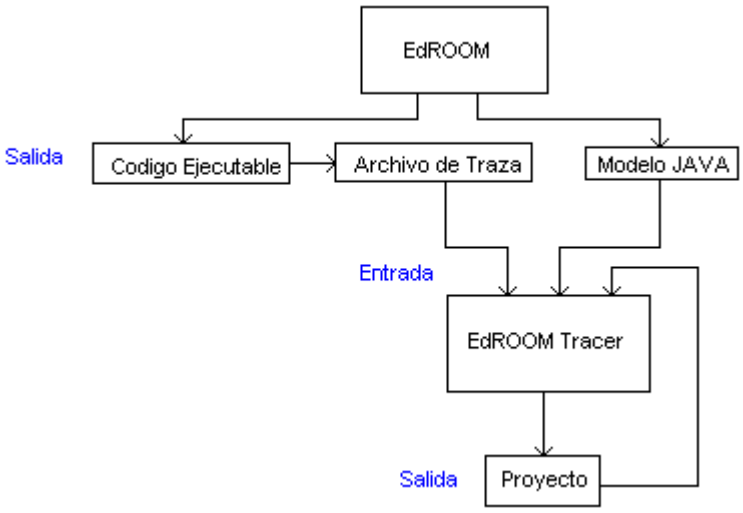


Fig 3-1

Una vez compilado y cargado el modelo, la estructura del mismo se visualiza en pantalla (fig 3-2). En el árbol de actores se visualiza el nombre de cada actor y la composición de actores del modelo. Después de la creación de los grupos, se pueden asignar los grupos a los actores sobre el mismo árbol. También se puede cargar un proyecto que contiene los grupos y la asignación de actores.

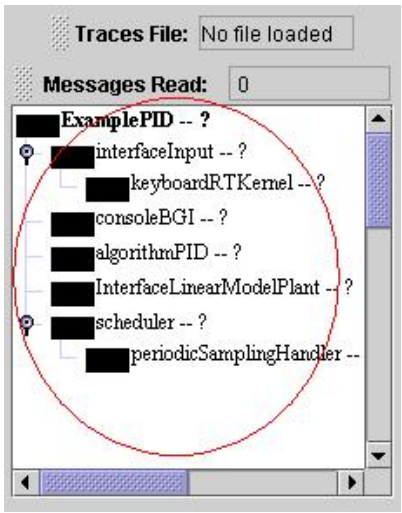


Fig. 3-2

Después de cargar el fichero de traza ya se pueden empezar el proceso de trazado (fig 3-3) sobre el modelo cargado y con la configuración de grupos que se ha dado a los actores.

```
MsgIn  keyboardRTKernel                  0      0      0
MsgIn  interfaceInput                    0      0      0
MsgOut      interfaceInput               sendOrder     actorReady
MsgIn  consoleBGI                        0      0      0
MsgIn  algorithmPID                      0      0      0
MsgIn  InterfaceLinearModelPlant         0      0      0
MsgOut      InterfaceLinearModelPlant    outputSampling      actorReady
MsgIn  periodicSamplingHandler           0      0      0
MsgIn  scheduler                         0      0      0
MsgOut      scheduler          PIDmanagement              initialConfiguration
MsgOut      scheduler          outputSamplingManagement        periodValue
MsgOut      scheduler          inputOperatorManagement    initialConfiguration
MsgOut      scheduler          consoleManagement          periodValue
MsgOut      scheduler          consoleManagement          initialConfiguration
MsgIn  scheduler                         6      2      0
MsgIn  scheduler                         6      2      0
MsgIn  algorithmPID                      5      4      0
MsgOut      algorithmPID       actorManagement    actorReady
MsgIn  periodicSamplingHandler           4      3      0
....
```

Fig 3-3

El formato del fichero de traza es el siguiente:

- MsgIn: Este tipo de mensajes se refieren a la parte de comportamiento. Los parámetros de este mensaje son el nombre del actor a que se refiere y tres dígitos que representan el número de la señal, el nombre de la transición y el nivel de la transición. Por ejemplo:

```
MsgIn  algorithmPID                      5      4      0
```

- MsgOut: Este tipo de mensajes se refieren a la parte de estructura. Los parámetros de este mensaje son el nombre del actor a que se refieren, el nombre del puerto y el nombre de la señal. Por ejemplo:

```
MsgOut      scheduler    PIDmanagement        initialConfiguration
```

Este archivo es generado automáticamente por EDROOM durante la ejecución del modelo.

Para la depuración hay que crear un grupo como mínimo en el cual se añadirán los actores que queramos visualizar para estudiar su comportamiento en el proceso de trazado. Se pueden crear tantos grupos como se quiera y añadir los actores que se desee. Así se consigue un significado gráfico porque podemos juntar elementos con igual o distinto comportamiento (según se quiera) para tener una mejor visión en la depuración (fig 3-2).

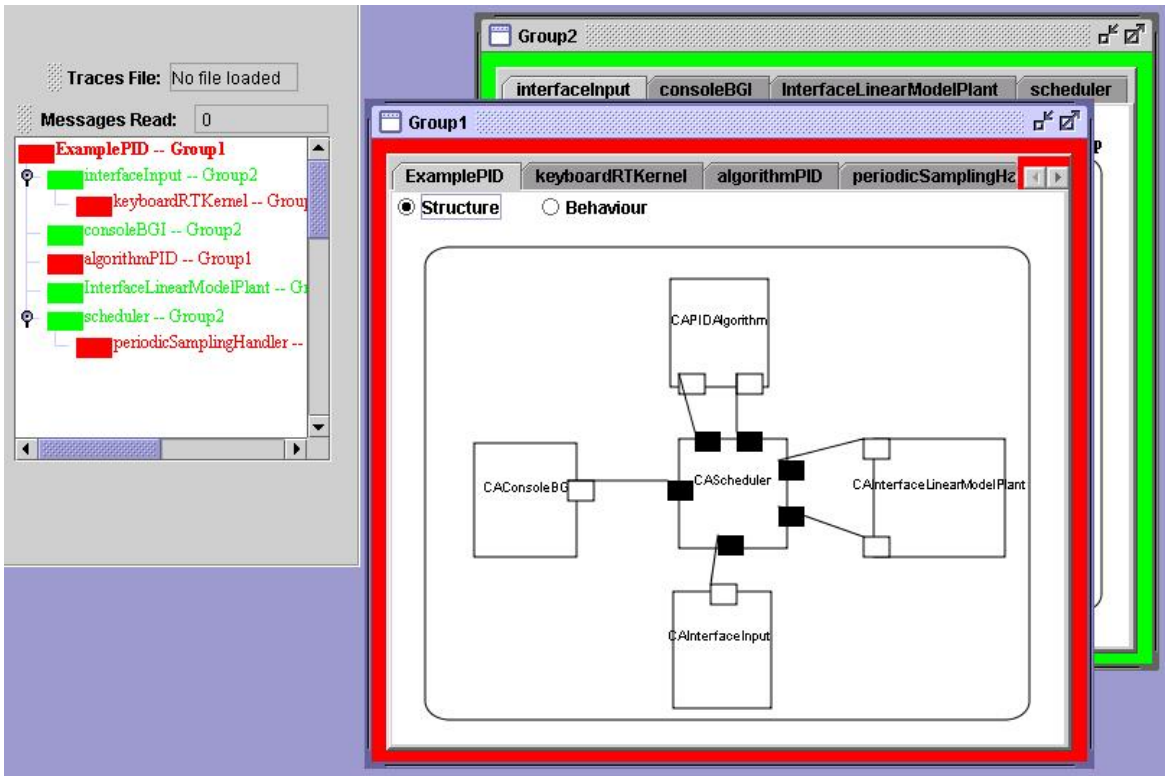


Fig 3-2

Se puede comprobar que la estructura y comportamiento que se muestra es la misma que con EDROOM.

3.2 Resultado del Software desarrollado

Una vez compilado y cargado el modelo, con los grupos ya creados y un fichero de traza abierto con el programa, ya se está en condiciones de empezar a depurar. En este momento ya se puede procesar el fichero de traza a través de los botones disponibles e introducir/quitar puntos de ruptura en los elementos de estructura y comportamiento de los actores (fig 3-3).

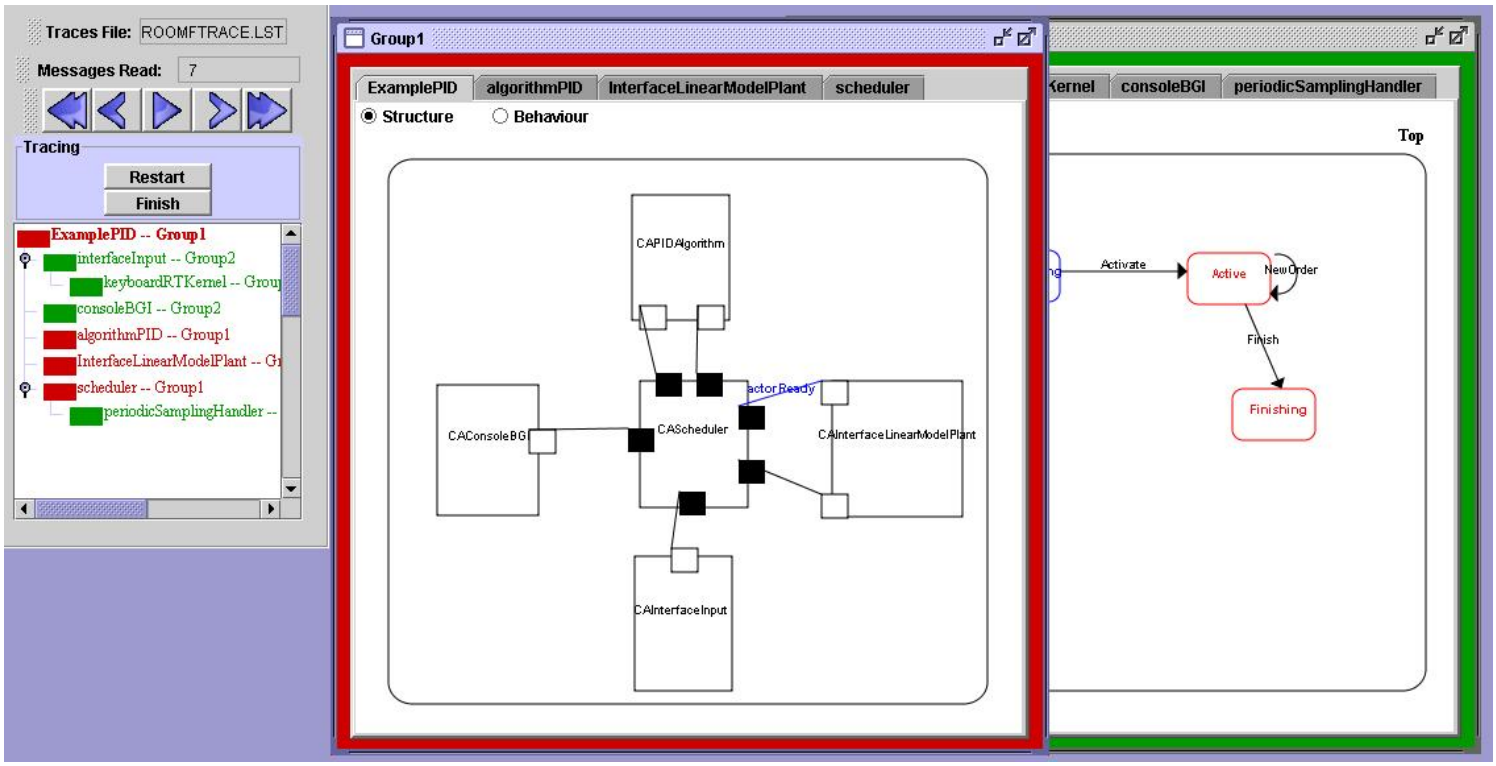


Fig 3-3

El resto de acciones que se pueden realizar con el programa se detallan en el "Manual de EDROOM TRACER" que sigue a este capítulo.

3.3 Estructura Básica

La estructura del modelo de ejemplo visto en UML se puede ver en la figura 3-4:

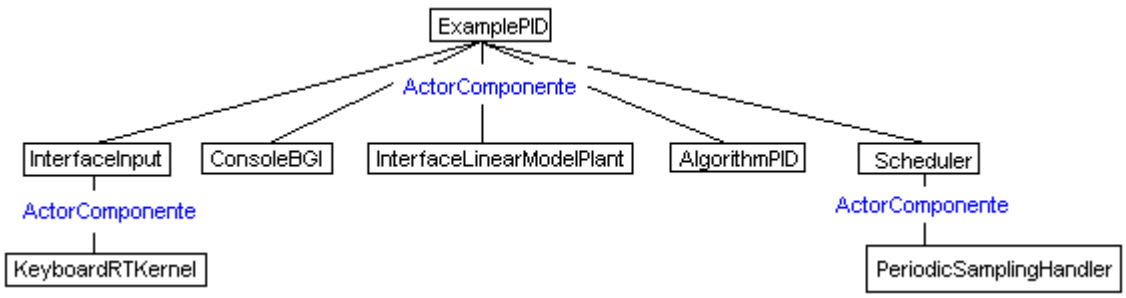


Fig 3-4

Esta estructura (a primer nivel) se puede ver cargada en EDROOM Tracer en la figura 3-3 en la parte de estructura. El actor principal es el nombre del modelo que contiene al resto de actores. Los actores pueden tener subactores (estructura). Dentro del comportamiento también puede haber contextos distintos al top, ya que un mismo subestado puede representar otro contexto.

Las clases que forman la base de la aplicación se dividen en los siguientes grupos:

DTraceRoom*:

Group: Crea un grupo de depuración formado por actores. Se caracteriza por un nombre y por un color.

GroupList: Es una lista que contiene grupos de la clase anterior y da métodos para su manipulación.

MsgStruct: Define un hilo encargado de los eventos de dibujado en la parte de estructura.

Threads: Define una clase hilo padre de la cual heredarán los hilos de estructura y comportamiento.

MTraceRoom*:

Actor: Es la clase que modela un actor de ROOM, contiene todo el control sobre las acciones gráficas, tanto en comportamiento como en estructura.

ActorComp: Representa la parte de comportamiento de un actor ROOM y métodos para su gestión. Tiene acceso a su actor y la parte gráfica que representa al actor comportamiento.

ActorStruct: Representa la parte de estructura de un actor ROOM y métodos para su gestión. Puede tener asociado listas de "ActorComp", "Connection" e "Interface".

Connection: Maneja la lógica de una conexión y sus puertos en la parte de estructura.

Interface: Maneja la lógica de los puertos en la parte de estructura y está relacionado con la parte de estructura.

Message: Representa el modelo de un mensaje.

CTX: Representa el contexto de un actor. Todo actor tiene un contexto que es el top. Los estados en la parte de comportamiento también pueden representar un nuevo contexto cuando se alcancen.

CurrentMessage: Indica el mensaje actual en ejecución.

GTraceRoom*:

ActorComp: Representa el modelo gráfico de un actor de comportamiento.

Connection: Representa el modelo gráfico de una conexión. Es el objeto donde se dibuja y hereda de "Gobject".

EntryPoint: Representa el modelo gráfico de un punto de entrada. Es el objeto donde se dibuja y hereda de "Gobject".

ExitPoint: Representa el modelo gráfico de un punto de salida. Es el objeto donde se dibuja y hereda de "Gobject".

InitSubState: Representa el modelo gráfico de un estado inicial. Es el objeto donde se dibuja y hereda de "Gobject".

SubState: Representa el modelo gráfico de un subestado. Es el objeto donde se dibuja y hereda de "Gobject".

Interface: Representa el modelo gráfico de un puerto. Es el objeto donde se dibuja y hereda de "Gobject".

Transition: Representa el modelo gráfico de una transición. Es el objeto donde se dibuja y hereda de "Gobject".

GObject: Es la clase que representa un objeto gráfico de EdROOM. Todas las clases gráficas heredan de ésta.

La clases encargadas de la visualización son las siguientes:

TFormMain: Clase principal, que presenta el formulario inicial. Es la ventana donde se realizan todas las funciones esenciales de la aplicación.

TFromTraceWindow: Genera las ventanas de traza con todos sus componentes. Equivale a un grupo y muestra los actores asociados. Cada actor está asociado a un tab que muestra sus atributos de estructura y comportamiento.

TFormGroups: Se crea los grupos para la depuración. En él se introduce el nombre y el color del grupo. Se pueden eliminar grupos.

Otras clases auxiliares para el entorno gráfico son:

TabbedPane: Componente gráfico que va a estar contenido en un formulario TFormTraceWindow, formado por un JTabbedPane especializado. Va a contener un TabRoom en cada tab.

TabRoom: Componente gráfico que está contenido en los tabs del TabbedPane del TFormTraceWindow. En el se muestra los botones para

seleccionar lo que se quiere ver y los dos zonas de dibujo para comportamiento y estructura.

CustomRenderer: Gestiona las eventos que se producen al pulsar sobre el JTree y que luego generan acciones de presentación.

FilterRoom: Gestiona la apertura de archivos de traza, permitiendo que el usuario pueda abrir solo los archivos de extensión ".lst".

FilterRoomXML: Gestiona la apertura de archivos de proyectos, permitiendo que el usuario pueda abrir solo los archivos de extensión ".xml".

Para más información sobre el código y su documentación, se pueden obtener los JAVADOCS de las clases de la aplicación en formato HTML en la página web de EDROOM.

La siguiente figura 3-5 muestra en modelo de clases de la estructura del programa:

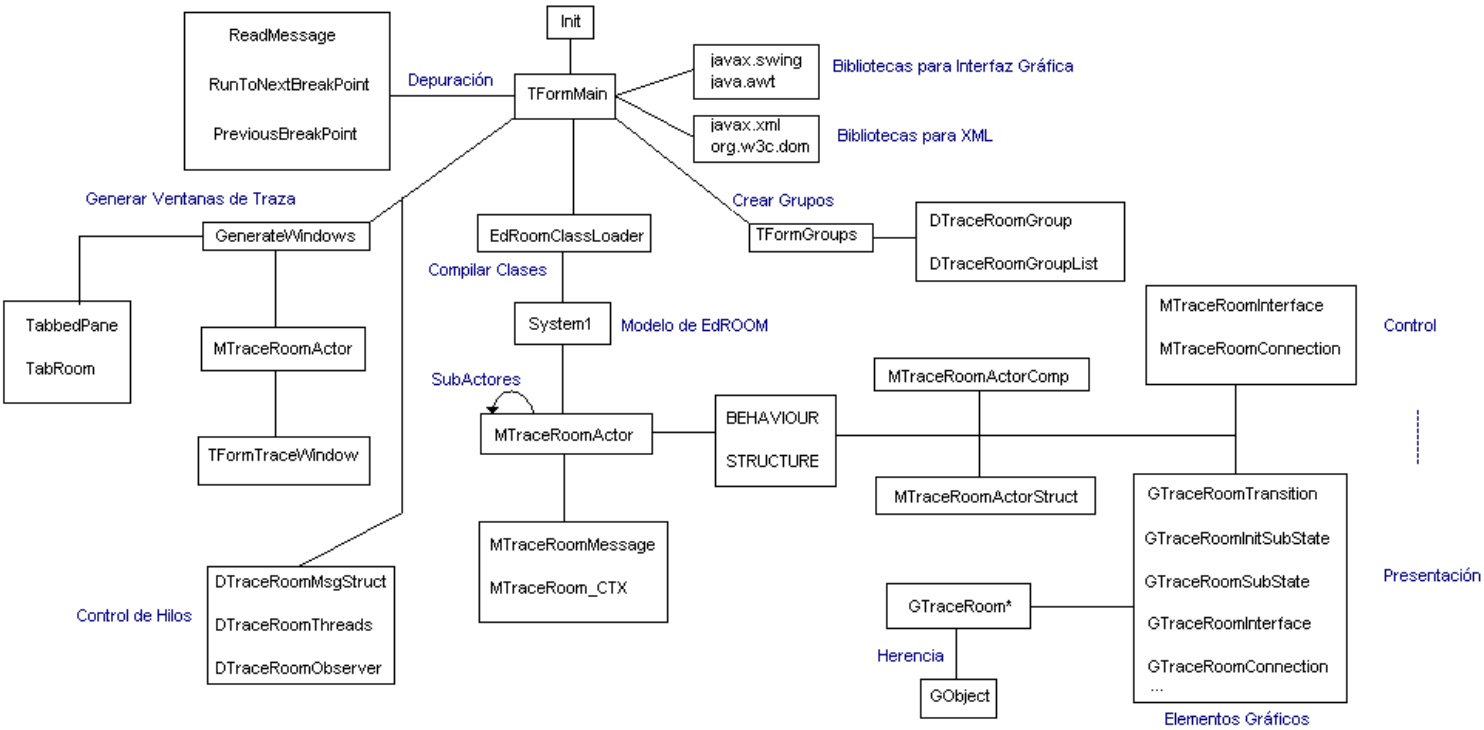


Fig 3-5

La aplicación comienza con la clase Init.java que llama al formulario principal TFormMain, que es la clase que gestiona todo la aplicación.

La clase EdRoomClassLoader se encarga de compilar (si es necesario) y cargar los modelos. Los modelos están formados por actores en los cuales está codificado su lógica de comportamiento y estructura. Los modelos poseen independencia total del trazador, ya que tan solo necesita que cumpla el

modelo EDROOM para poder compilarse y poder llegar a ser depurados. Además los modelos pueden ser cargados desde una URL.

En las clases encontradas dentro del modelo se definen los contextos de comportamiento y los elementos de estructura. Además de las acciones a tomas cuando se reciban ciertos estímulos.

Las clases GTraceRoom* son las encargadas de dibujar los elementos en los formularios. Tiene asociados hilos, para que la ejecución de las acciones gráficas se realice de forma concurrente.

3.4 Tecnología y Técnicas

En la aplicación se ha usado el estándar XML [9] para guardar la información que concierne a la configuración de la aplicación: ruta del compilador de JAVA y el delay de los mensajes. Además con este mismo formato se guardan los proyectos de depuración: grupos y actores por grupo. Tanto para la escritura como para la lectura de archivos XML desde JAVA se ha utilizado el API DOM, que permite manejar los ficheros XML como árboles y además incluye funciones para realizar diversas búsquedas dentro de dichos árboles.

La aplicación es fácilmente escalable y está implementada de forma estructurada. Todos los objetos gráficos heredan de GTraceRoom, permitiendo asociar un estado diferente a cada objeto gráfico.

Se ha usado la técnica de doble buffering que permite mayor eficiencia en la ejecución gráfica, aprovechando así la capacidad de procesamiento de los procesadores actuales y permitiendo ampliar la información gráfica de forma eficiente.

El programa se ha desarrollado usando hilos separados para cada actor, simulando así la ejecución concurrente de un sistema de tiempo real. El trazado se realiza usando mecanismos de concurrencia, de exclusión mutua y señalización, basándose en el archivo de trazas que se generó durante la ejecución real.

Como se ha mencionado se pueden insertar puntos de ruptura a lo largo del trazado pulsando sobre el objeto gráfico que servirá como punto de ruptura. Cualquier objeto gráfico tiene asociado un estado de seleccionado y de iluminado, definido en la clase *GTraceRoom* por lo que es fácilmente escalable en caso de que las especificaciones ROOM sean extendidas con nuevos objetos. Al declarar la clase del nuevo objeto basta con heredar *GTraceRoom*, haciendo así que el nuevo objeto sea igualmente un objeto gráfico.

El motor de trazado se ha optimizado al completo para que las ejecuciones hasta el siguiente breakpoint se realicen de forma eficiente, incluso para archivos de traza grandes. Toda la información necesaria para optimizar el trazado está incluida dentro del propio motor de trazado, teniendo así la ventaja de que los modelos no necesitan absolutamente ninguna información de control de trazado. Esto también nos conduce a que los modelos sean completamente independientes del trazador.

La estructura y el diseño de EDRoomTracer se ha hecho con especial cuidado para que sea fácilmente escalable y siempre independiente de los modelos generados por EDRoom. Especialmente para este último aspecto, Java es la plataforma ideal debido que no solamente permite ejecutar el trazador sobre cualquier sistema operativo, sino que permite enlazar clases dinámicamente durante su ejecución. Esto nos ha permitido construir un cargador de clases (*EdRoomClassLoader*) que permite cargar los modelos compilados tanto desde el sistema de ficheros local como remotamente desde cualquier URL.

Todo el código ha sido suficientemente documentado para que el trazador pueda ser ampliado y adaptado a las futuras necesidades que pueda haber. Toda la descripción de métodos y clases se ha realizado para soportar *Javadoc*, permitiendo de esta manera actualizar y publicar fácilmente cualquier cambio que se realice sobre la documentación.

Manual de EdROOM TRACER

Toda esta documentación está colgada de la página web desarrollada para contener toda la información sobre EdROOM.

INSTALACIÓN DE EDROOM TRACER JAVA

REQUERIMIENTOS

-JDK 1.4 o versiones superiores.

INSTALACIÓN

- 1- Copiar la carpeta "EdRoomTraceJava" en el disco duro.
- 2- Compilar los ficheros java mediante la orden

  *javac *.java*

  desde la línea de comandos.

- 3- Ejecutar el método Init mediante la orden

  java Init

  desde la línea de comandos.

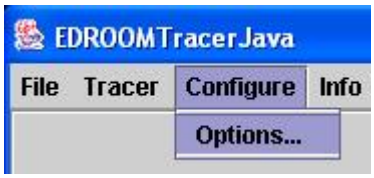
- 4- Aparecerá el menú principal de EdRoom Trace Java.

PARTE 1

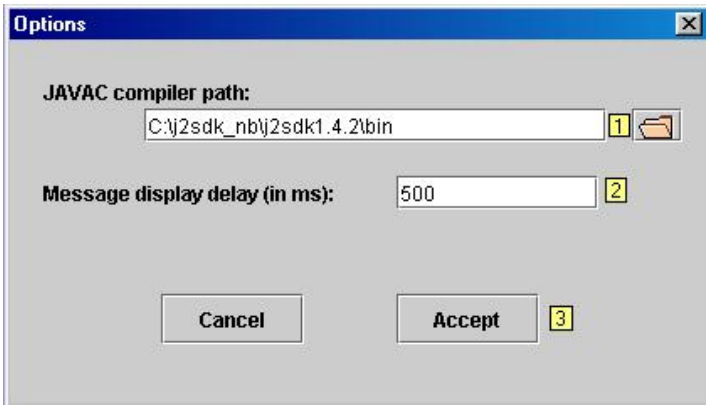
CONFIGURAR OPCIONES

Aquí se configuran los dos únicos parámetros de EdROOM Tracer:

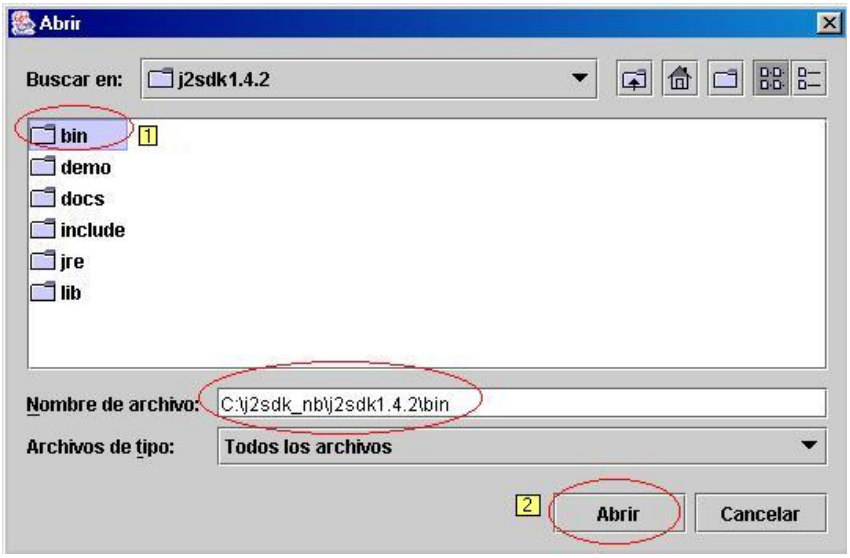
- 1 - El path del compilador de JAVA (javac) que se encuentra en la carpeta “bin” donde esté instalado Java Development Kit.
- 2 - El tiempo de retardo para la muestra de mensajes por pantalla. Contra mayor sea el valor, más rápido se procesarán los mensajes.



- **1- Click en "Configure" --> "Options..."**
- **2- Opciones**
  - [1]Buscar el path del compilador de JAVA.
  - [2]Escribir el tiempo de transición entre estados.
  - [3]Click en "Accept" para actualizar .



- **3- El path del compilador JAVAC**
  - [1]Buscar la carpeta "bin" en el JAVA_HOME en el disco duro.
  - [2]Click en "Open".



PARTE 2

CARGAR MODELO

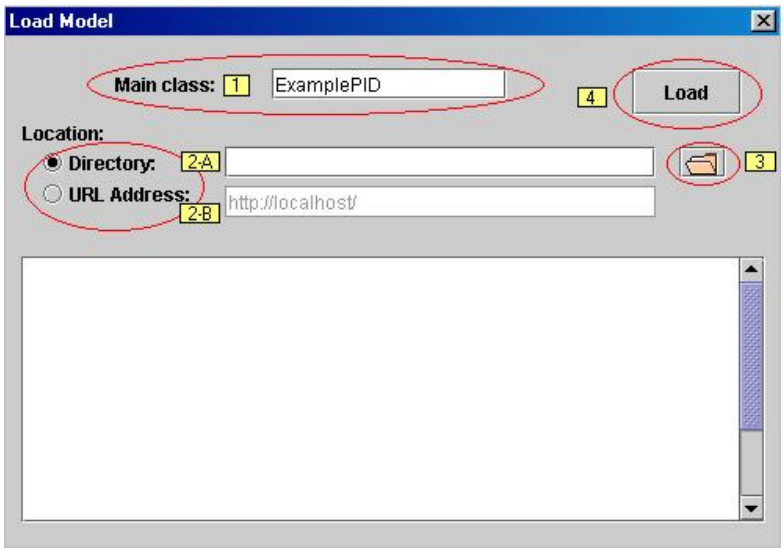
En esta parte se realiza la carga de un modelo para luego realizar sobre él el trazado. El modelo son un conjunto de clases JAVA que están contenidas en una carpeta. Al programa hay que indicarle donde se ubica la carpeta que contiene el modelo. Si el modelo no está compilado, lo compila y si encuentra algún error lo muestra por pantalla.

El modelo se puede cargar de dos formas distintas:

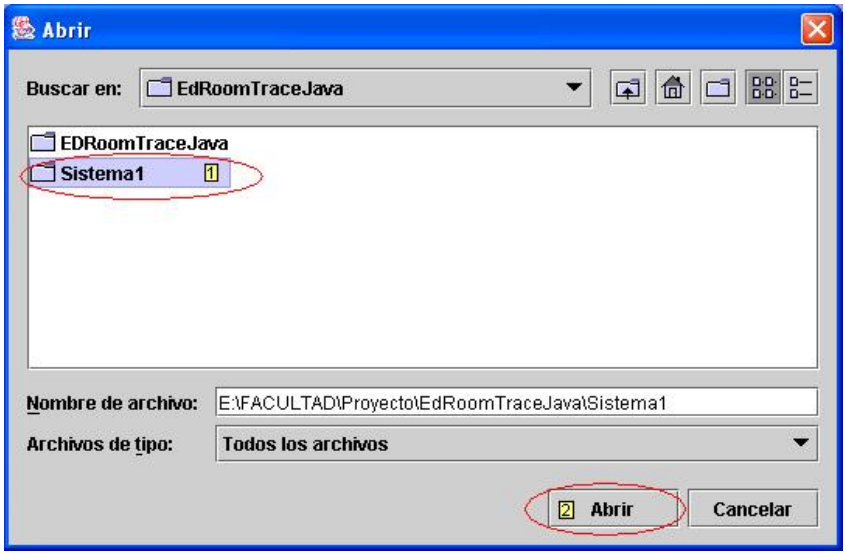
- Desde el disco duro: La carpeta que contiene las clases JAVA está en el disco duro del ordenador.
- Desde una URL: Se puede introducir una URL para que el programa busque en ella los clases que forman el modelo y las cargue en el trazador.

- **1- Pulsar en "File"-->"Load Model".**

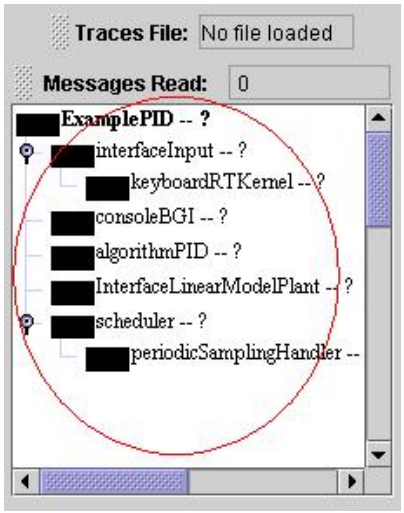
- **2- Cargar modelo**
  - [1] Escribir el nombre de la clase principal.
  - [2-A] Seleccionar la carpeta en la que se encuentra el modelo pulsando en [3] o
  - [2-B] Escribiendo una dirección desde la que descargar un modelo.
  - [4] Pulsar en "Load" para compilar y cargar el modelo.



- **[3] Seleccionar la carpeta en la que se encuentra el modelo.**
  - [1] Seleccionar la carpeta en la que se encuentra el modelo.
  - [2] Pulsar en "Abrir".



- **3- Modelo cargado: Ahora se puede visualizar la jerarquía de actores del modelo en el árbol de la izquierda.**



PARTE 3

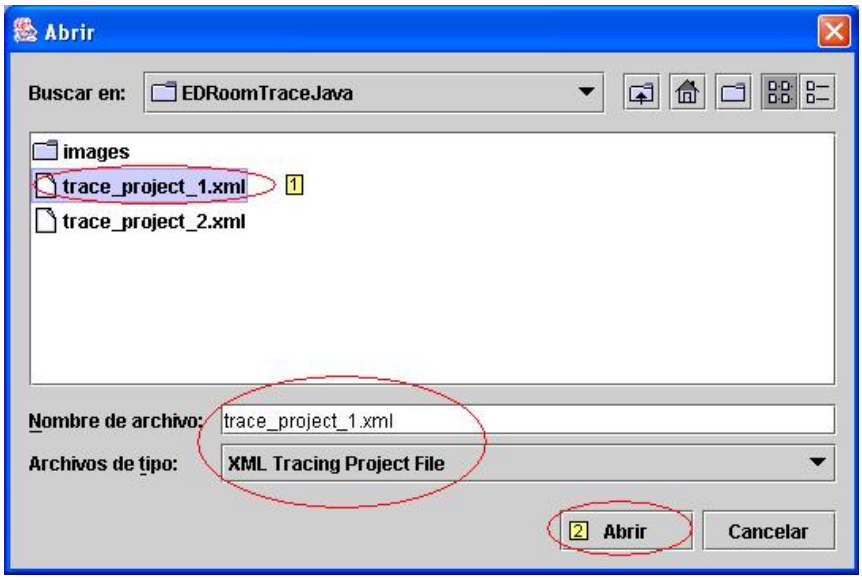
CARGAR UN PROYECTO DE TRAZA

Un proyecto es un archivo XML que contiene los grupos que se han creado para trazar un modelo en una ejecución anterior. Cada grupo contiene los actores que tiene asignados. Una vez cargado el proyecto se actualiza en árbol de los actores y se crean para grupo su ventana de traza.

1- Pulsar en "File" --> "Load Trace Project..."

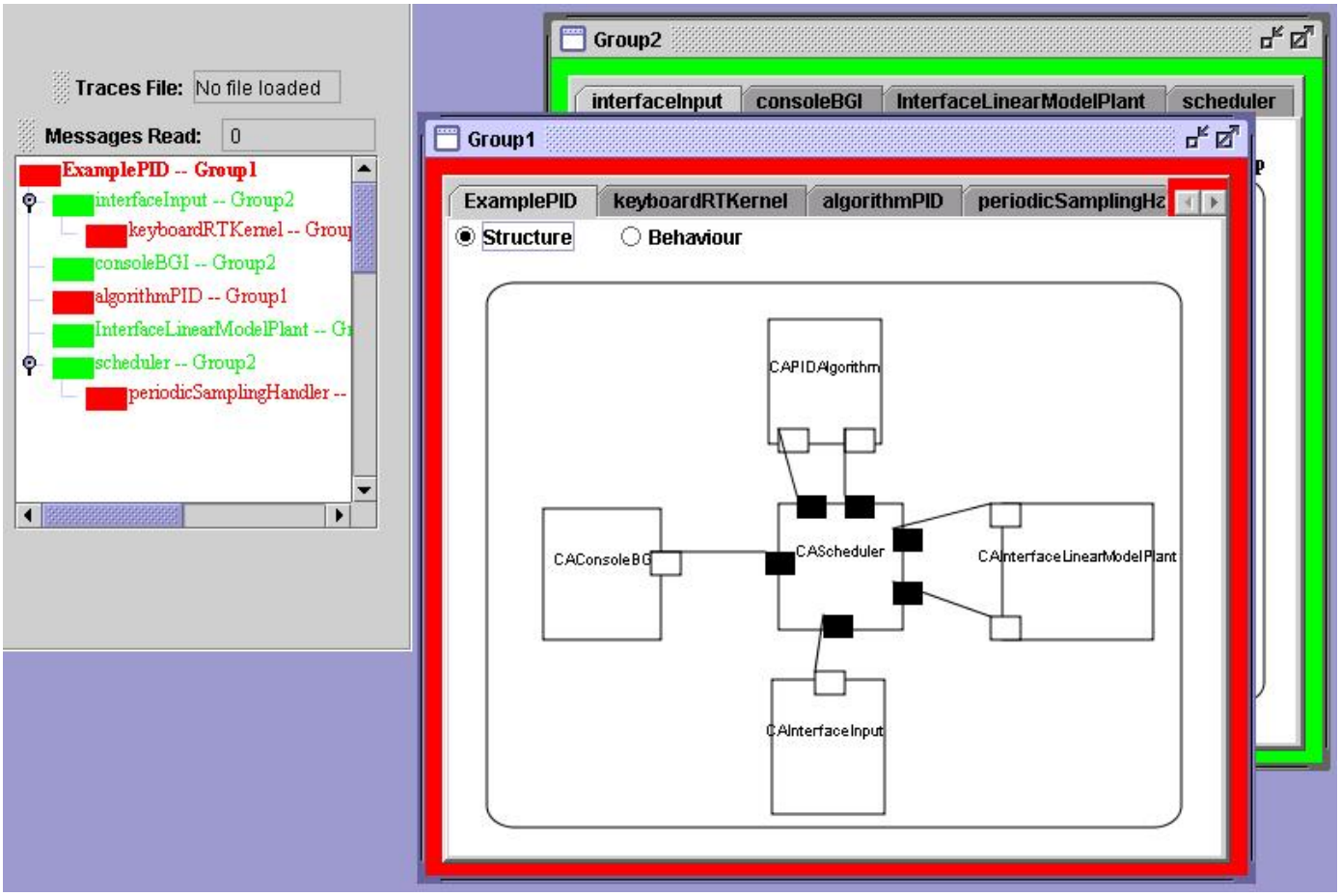
2- Buscar el fichero de proyecto de traza:

- [1] Seleccionar un fichero de proyecto de traza (la extensión de estos ficheros es ".xml").
- [2] Pulsar en "Abrir".



3- Fichero de proyecto de traza cargado:

- - Cada actor del árbol tiene el color y el nombre de su grupo.
- - Cada grupo tiene asignado un JInternalFrame que muestra sus actores junto con la representación del comportamiento y de la estructura de cada uno de ellos.



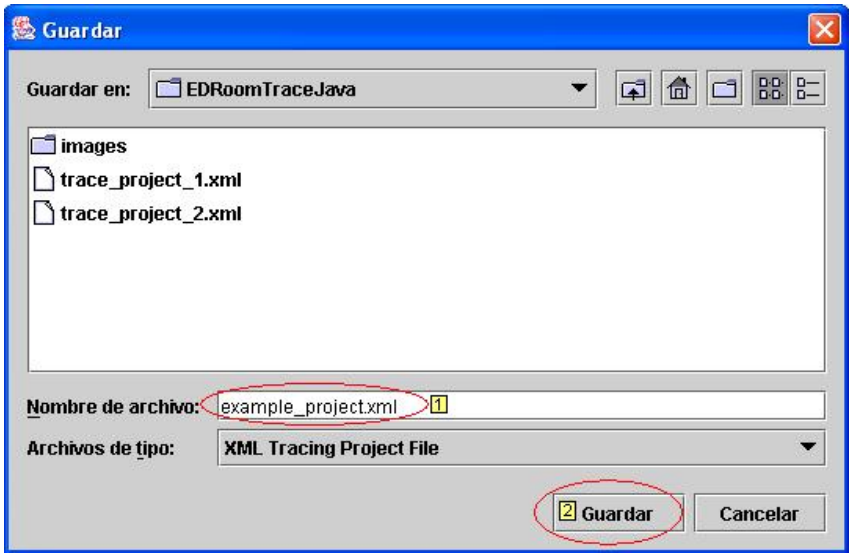
PARTE 4

GUARDAR UN PROYECTO DE TRAZA

En cualquier ejecución podemos guardar los grupos creados para trazar un sistema en un fichero XML para poder recuperarlo en una ejecución posterior. Así se evita volver a crear todos los grupos de cero cada vez que se quiera trazar un sistema específico.

- **1- Pulsar en "File" --> "Save Trace Project..."**

- **2- Seleccionar el nombre del fichero en el que se guardará:**
  - [1] Escribir el nombre del fichero con la extensión ".xml".
  - [2] Pulsar en "Guardar".



PARTE 5

CREAR UN NUEVO PROYECTO DE TRAZA

La creación de un proyecto de trazado para un sistema consiste en crear una serie de grupos. Cada grupo tiene un nombre y un color que no se puede repetir.

En la ventana de edición se pueden crear todos los grupos que se quieran y también borrarlos en el mismo momento.

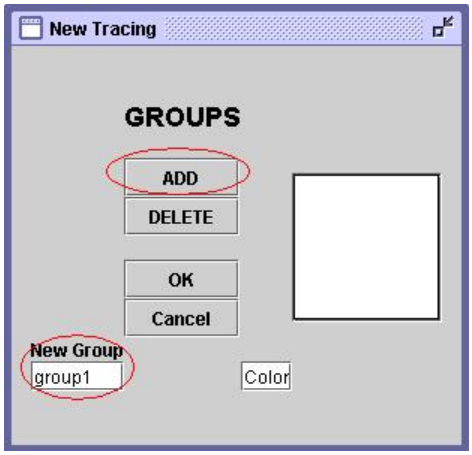
Una vez aceptados los grupos introducidos, se pueden asignar a los actores pulsando con el ratón el árbol de edición. Si no tenemos ninguna ventana de trazado de abierta, se pueden cambiar los grupos a los actores.

Después de asignar los grupos a los actores se crean las ventanas de trazado para cada grupo. Cada ventana se identifica por el color y nombre del grupo. En ella aparecen todos los actores que tengan asociados.

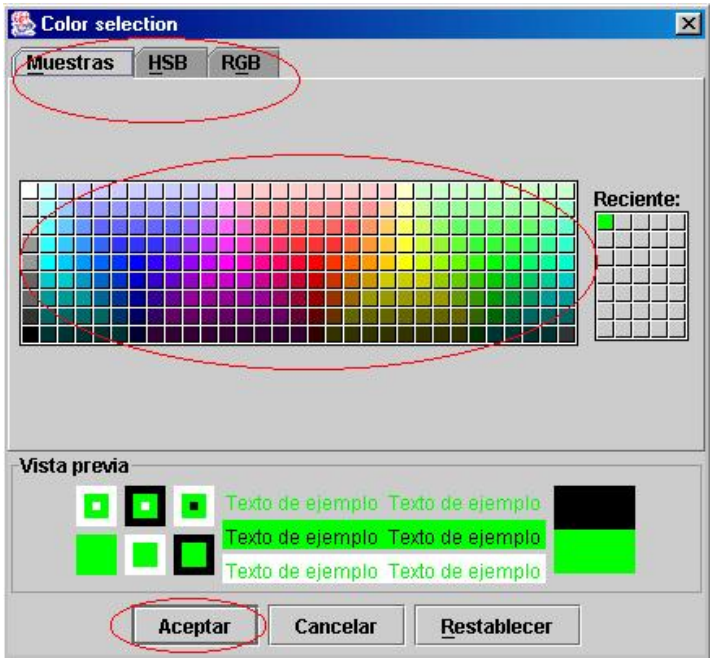
No es obligatorio asignar un grupo a cada actor. Al contrario, una vez abiertas las ventanas de trazado, se pueden asignar a actores sin grupo algún grupo y en ese momento se actualiza la ventana correspondiente. Incluso se pueden crear grupos con una traza abierta. Lo que no se puede hacer es borrar un grupo con una traza abierta.

- **1- Click en "File" --> "New Trace Project..."**

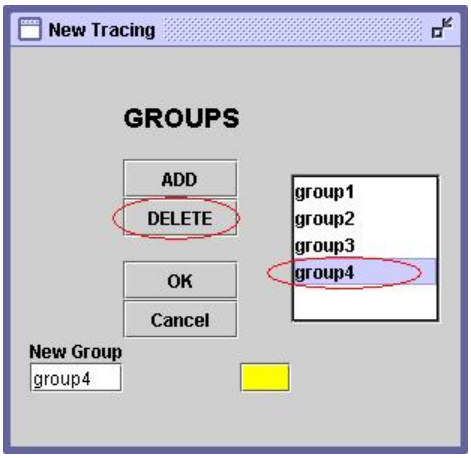
- **2- Crear Grupos**
  - [1]Escribir el nombre del grupo en campo "New Group".
  - [2]Click en "ADD".



- [3]Elegir un color de la paleta para identificar el grupo.
- [4]Click en "Aceptar". El grupo se ha añadido en la lista.

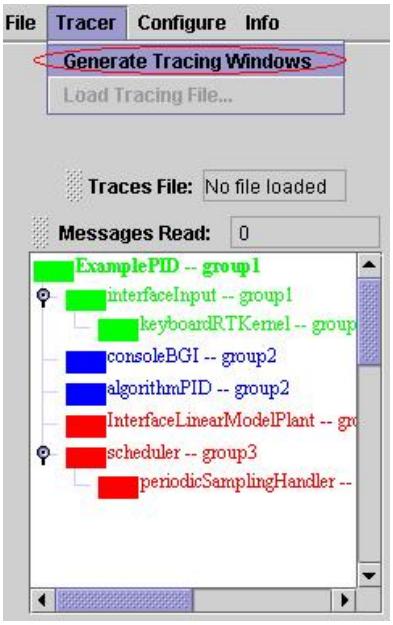


- **3- Borrar Grupos**
  - [5]Elegir el grupo de la lista.
  - [6]Click en "DELETE". El grupo se elimina de la lista.

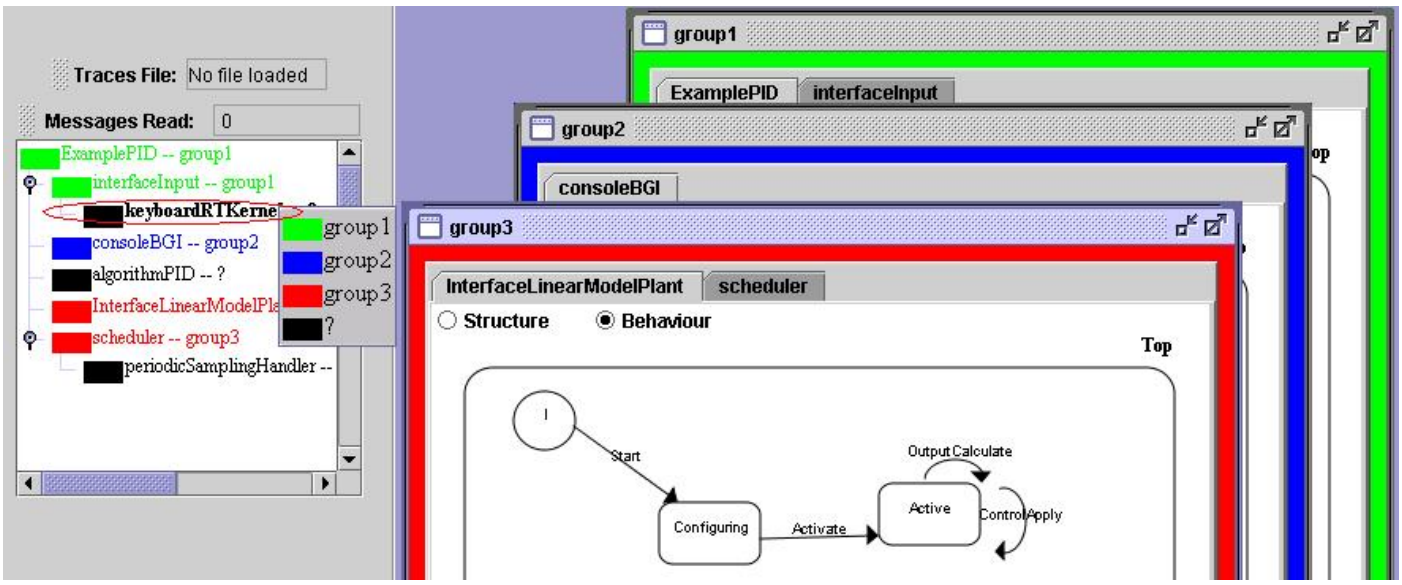


- **4- Click en "Ok" para actualizar los grupos. El formulario desaparece**
- **5- Asignar grupos en el JTree**
  - [7]Elegir un nodo del JTree.
  - [8]Pulsar el botón derecho del ratón y aparecerá un JPopupMenu con los grupos disponibles.
  - [9]Elegir una opción y el nodo se actualiza con el nombre y color del grupo.

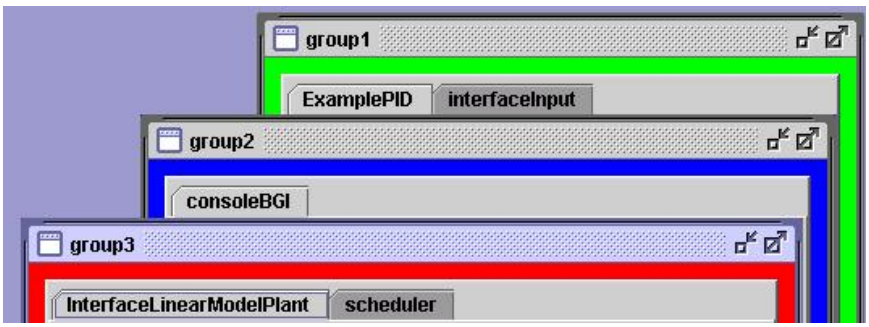
- **6- Fin de las Asignaciones**
  - [10]Click en "Tracer" --> "Generate Tracing Windows"



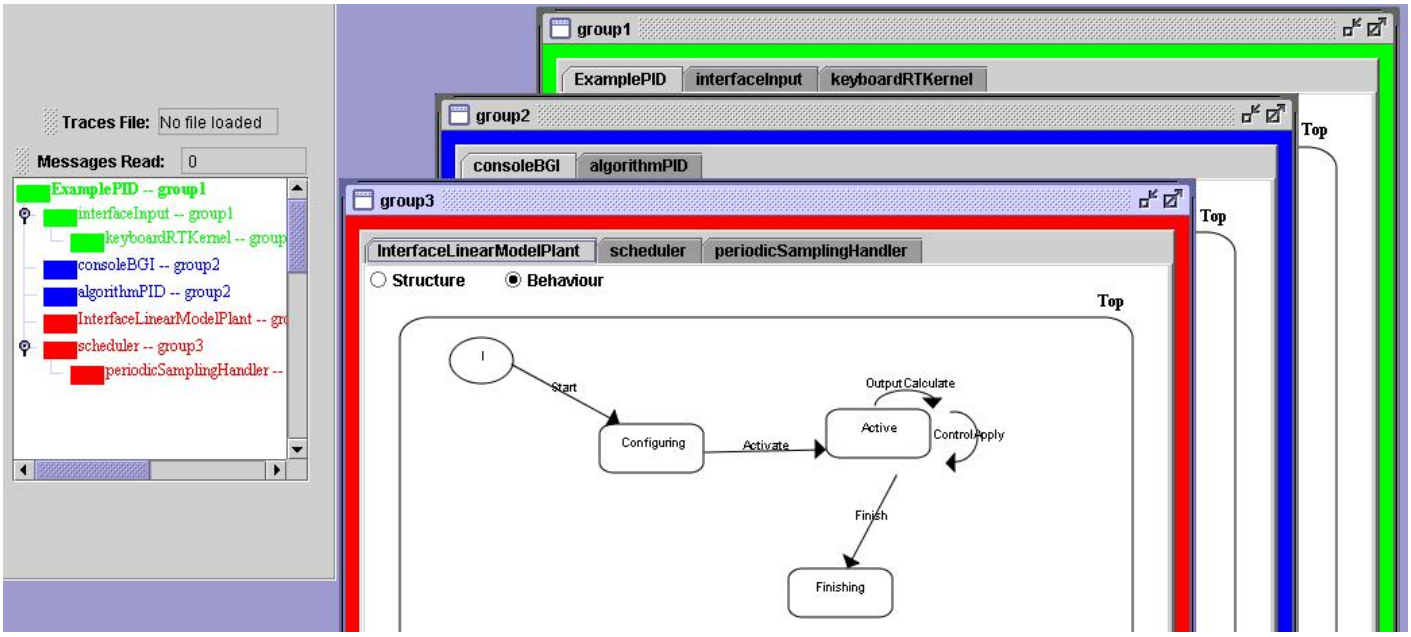
  - [11]Ahora se puede ver los grupos generados con sus actores correspondientes



- **7- Notas:**
  - [a]Se puede crear un proyecto de traza sin tener que asignar grupo a todos los nodos



- [b]Se puede asignar un grupo a un nodo (sin grupo) cuando se está ejecutando una traza y el grupo ya existe en la depuración



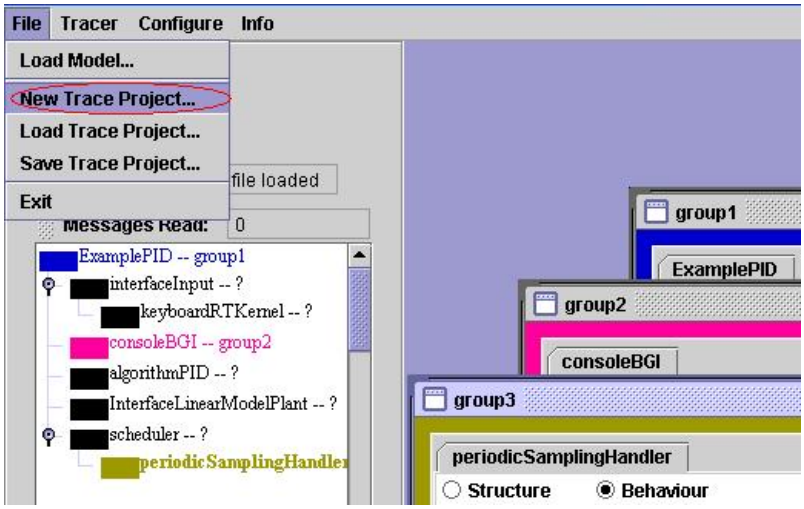
- [c]Se puede cambiar el grupo a un nodo si no hay una traza ejecutandose
- [d]No se puede eliminar grupos cuando se está ejecutando una traza pero si se pueden crear grupos en ejecución

PARTE 6

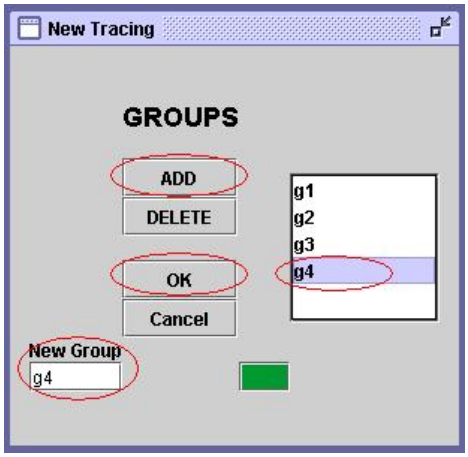
CREAR GRUPOS EN EJECUCIÓN

Una vez abierta una traza, se pueden crear nuevos grupos y asignarlos a actores que no tienen grupo. El proceso es idéntico al comentado en la parte 5.

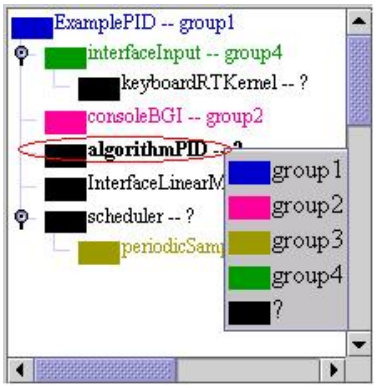
- **1- Click en "File" --> "New Trace Project..."**



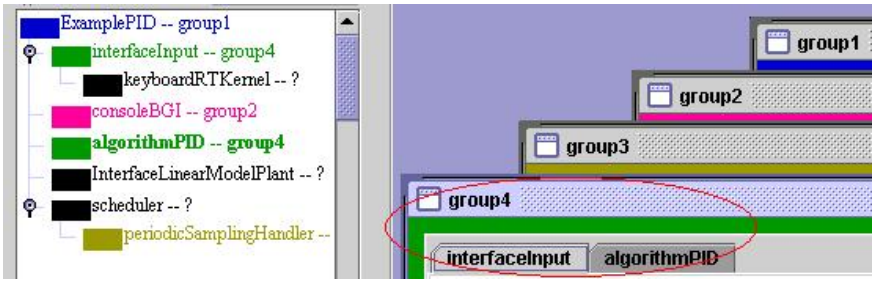
- **2- Crear Grupos**
  - [1]Escribir el nombre del grupo en el campo "New Group".
  - [2]Click en "ADD".
  - [3]Elegir un color de la paleta para el grupo.
  - [4]Click en "Aceptar" (Paleta). Se añade el grupo.
  - [5]Click en "Ok" (Formulario). Se actualizan los grupos.



- **3- Asignar Grupos en el JTree**
  - [6]Elegir un nodo del JTree.
  - [7]Pulsar el botón derecho del ratón y aparecerá un JPopupMenu con los grupos disponibles.
  - [8]Elegir una opción y el nodo se actualiza con el nombre y color del grupo.



- **4- Fin de las Asignaciones**
  - [9]Click en "Tracer" --> "Generate Tracing Windows"
  - [10]Ahora se ven todos los grupos con sus actores



PARTE 7

CARGAR FICHERO DE TRAZA

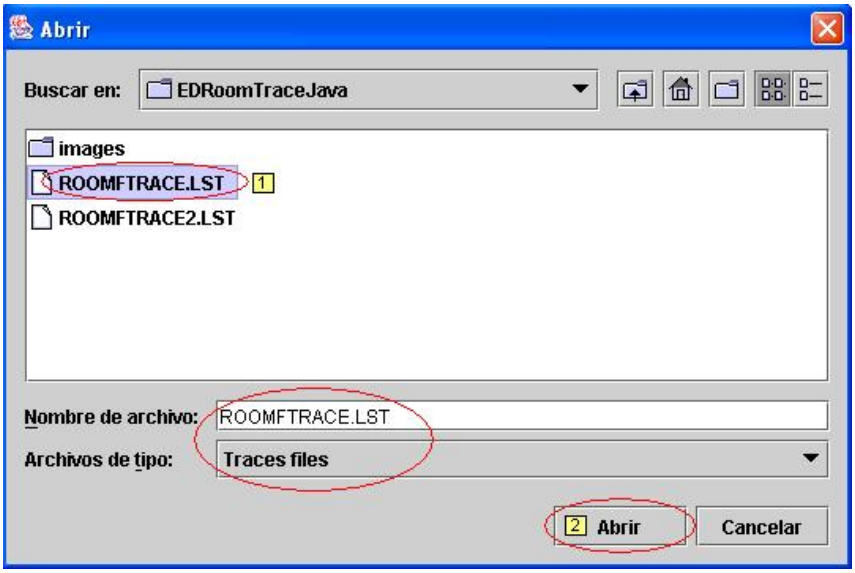
La traza de los sistemas se graba en archivos de formato ".lst". Al programa hay que indicarle que archivo de traza queremos abrir.

Una vez abierto el archivo, en pantalla aparece una botonera con los elementos para el trazado. Además se puede ver el nombre del fichero abierto y el número de mensajes procesados.

Se comentan la funcionalidad de los botones más adelante. También se incluyen las imágenes de cómo se visualizan los elementos cuando se les coloca un breakpoint y cuando un mensaje les ha alcanzado.

- **1- Click en "Tracer" --> "Load Tracing File..."**

- **2- Cargar Archivo**
  - [1] Elegir archivo con extensión "*.lst"
  - [2] Click en "Aceptar" para cargar el fichero.

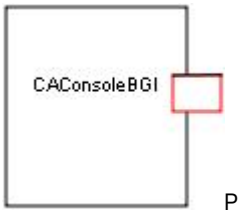


- **3- Botones de traza**

  -  Breakpoint Previo: La ejecución vuelve al breakpoint anterior
  -  Mensaje Anterior: Ejecuta el mensaje anterior

-  Ejecutar: Leer mensajes sin parar
-  Parar: Para la lectura de mensajes contínua
-  Siguiente Mensaje: Lee un mensaje
-  Siguiente Breakpoint: Leer mensajes hasta alcanzar el siguiente breakpoint

-  Restart Tracing: Los actores se reinician y el fichero de traza se lee desde el principio
-  Finish Tracing: Se cierra el fichero de traza y se esconden los botones

- **4- Poner Breakpoints**
  -  Estado (Behaviour): Click en un estado
  -  Transición (Behaviuor): Click en una transición
  -  Puerto (Structure): Click en un puerto
  -  Conexión (Structure): Click en una conexión

- **5- Elementos Alcanzables**
  -  Estado (Behaviour):
  -  Transición (Behaviuor):
  -  Puerto (Structure):
  -  Conexión (Structure):

PARTE 8

INFORMACION DE MEMORIA LIBRE

- **1- Click en "Info" --> "Free Memory"**

- **2- Información sobre la Memoria**
  - [1] Memoria (bytes) suministrada por la máquina virtual de java para el programa
  - [2] Máxima memoria (bytes) usada por el programa
  - [3] Indica el % de memoria usadad.
  - [4] Click en "Aceptar" para esconder el formulario.

PARTE 9

SALIR

- **1- Click en "File" --> "Exit"**

DOCUMENTACIÓN ENTREGADA

- Trazador de EdROOM en JAVA.
- Documentación del trazador en formato HTML generado mediante Javadoc.
- Manual de usuario, incluyendo instalación y manejo de la aplicación en formato HTML. Se incluye en los siguientes idiomas:
    - Inglés.
    - Español.
    - Japonés.
    - Francés.
    - Alemán.

También se incluye una página web desde la que se pueden descargar tanto la aplicación como la documentación, así como publicar noticias y foros relacionados. Esta página se puede ampliar dinámicamente, por lo que se pueden añadir nuevas secciones referentes a diversos proyectos.

Se incluye también un manual de instalación y administración de esta página.

BIBLIOGRAFÍA

[1] R.Polo, O, 2002, "ROOM Model Based Automatic Code Generator for Real Time Control Systems" Tesis Doctoral.

[2] R.Polo, O, 2002 "EDROOM. Automatic C++ Code Generator for Real Time Systems Modelled with ROOM".

[3] Selic, B., Gulleckson, G., and Ward, P.T., 1994, "Real-Time Object Oriented Modelling", NewYork, John Wiley and Sons.

[4] RTKernel. www.on-time.com

[5] CMX. www.cmx.com

[6] WxWorks. www.windriver.com

[7] RTAI. www.aero.polimi.it/~rtai/

[8] Sun MicroSystems www.sun.com

[9] XML http://www.w3c.org

[10] MySQL http://www.mysql.org

[11] GPL http://www.gnu.org/copyleft/lgpl.html

[12] Segundo Esteban San Román: Tesis Doctoral: Capítulo V : Implementación Experimental
http://www.dacya.ucm.es/segundo/research.htm

[13] Tomcat http://jakarta.apache.org/tomcat/