DEPARTAMENTO DE
ELECTRÓNICA Y COMPUTADORES

UNIVERSIDAD DE CANTABRIA

Trabajo de Investigación

DISEÑO E IMPLEMENTACIÓN DE UN EDITOR GRÁFICO PARA EL MODELADO Y ANÁLISIS DE SISTEMAS DE TIEMPO REAL

María del Pilar del Rio Trueba
Septiembre - 2005

Índice

1. Introducción

1.1. Motivación y antecedentes

El área de los sistemas de tiempo real es una de las ramas de la ciencia y la tecnología que más desarrollo ha experimentado en los últimos años. Su aplicación se encuentra ampliamente difundida en casi todos los campos, como los sistemas de control, los sistemas de comunicación, el procesamiento de señales, y muchos más. La envergadura de tales sistemas puede ir desde pequeños sistemas empotrados basados en microcontroladores, hasta sistemas basados en varios computadores trabajando en red.

Consecuentemente y atendiendo a su creciente complejidad, urge la elaboración de técnicas y pautas genéricas que ayuden en el desarrollo organizado de este tipo de sistemas. Es este el pensamiento que orienta los actuales esfuerzos investigadores en los diferentes centros de investigación a realizar trabajos correspondientes a diversas áreas entre las que podemos citar el manejo de bases de datos, el desarrollo de lenguajes de programación en tiempo real y sistemas operativos o la elaboración de técnicas y metodologías para el análisis y diseño de sistemas de tiempo real (área en la que se engloba el presente trabajo).

Si atendemos a las principales características que presentan las aplicaciones de tiempo real, esto es, un estricto cumplimiento de tiempos, una alta confiabilidad, un manejo intensivo de datos, y un alto grado de predecibilidad y adaptabilidad, es fácil deducir que el correcto diseño y funcionamiento de un sistema de este tipo es una labor complicada. Es por esto que para el diseño de un sistema de tiempo real es común que se utilice la simulación del sistema antes de montar el prototipo. La simulación es más eficaz, más rápida y sobre todo mucho menos costosa. Para poder simular un sistema se necesita por un lado de un programa de simulación y por otro lado de modelos adecuados (suficientemente exactos para el tipo de aplicación y rápidos para los componentes que forman el sistema que se va a simular). Sin embargo, la simulación por si sola no es suficiente para sistemas de tiempo real estricto, ya que no hay garantía de que durante la simulación se alcancen todos los posibles estados del sistema.

Esta problemática motivó la creación y desarrollo del software de análisis y modelado MAST (Modeling and Analysis Suite for Real-Time Applications) [1][2][5][7][13] dentro del Departamento de Electrónica y Computadores de la Universidad de Cantabria. Su principal función es simplificar la aplicación de técnicas estándar y bien conocidas de análisis de sistemas de tiempo real, proporcionando al diseñador un conjunto de herramientas (ver apartado 1.5, “Herramientas MAST”) para aplicar técnicas de análisis de planificabilidad, de asignación óptima de prioridades o de cálculos de holguras, etc., sin necesidad de que éste tenga que conocer los detalles algorítmicos de los métodos.

La metodología MAST define un modelo capaz de describir el comportamiento temporal de un gran conjunto de sistemas de tiempo real, incluyendo sistemas distribuidos y conducidos por eventos con esquemas de sincronización complejos.

El modelo es apropiado para descripciones del sistema basadas en UML (Unified Modeling Language) [2] en las que se añade una vista de tiempo real al diseño y después se utiliza una herramienta automática para generar la descripción MAST. Esta descripción, en formato texto, se usa como entrada para las herramientas de análisis de tiempo real que incluyen los últimos progresos en teoría de planificación y que permiten conocer el comportamiento temporal de un sistema y si éste es planificable o no aplicando distintas técnicas.

MAST está en continuo proceso de desarrollo. De hecho, en el momento de la realización del presente trabajo, se habían implementado una interfaz gráfica para la configuración de las herramientas de análisis (figura 1.1) y otra para la visualización de resultados (figura 1.2). No obstante faltaba implementar un editor gráfico que facilitase al usuario el proceso de elaboración del modelo MAST y que le permitiese generar la descripción ASCII (en formato texto) del sistema automáticamente sin tener que utilizar la descripción UML. El principal objetivo de este trabajo es implementar y verificar dicho editor basándonos en el trabajo realizado en [22].

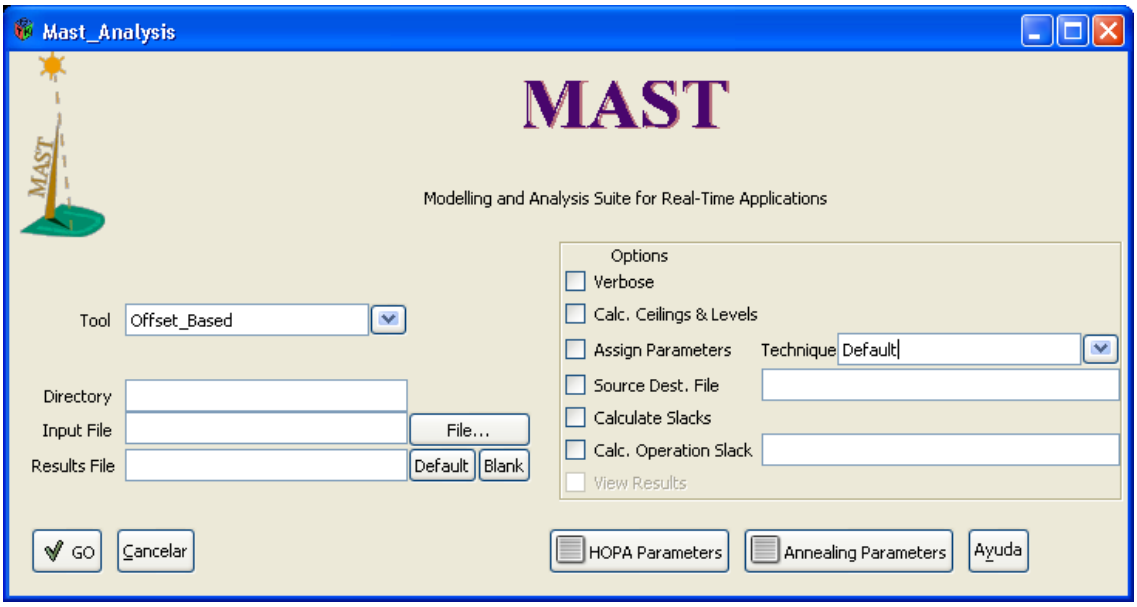

Figura 1.1: Pantalla de configuración de las herramientas de análisis

La necesidad de construir una herramienta gráfica que sirviese al usuario para elaborar el modelo de forma visual a través de una serie de pantallas y menús de configuración integrados en el editor, permitiendo entre otras cosas generar la descripción en formato texto del sistema automáticamente (prescindiendo de la descripción UML), y en definitiva, de convertir a MAST en un entorno más "amigable" fue la principal motivación de este trabajo.

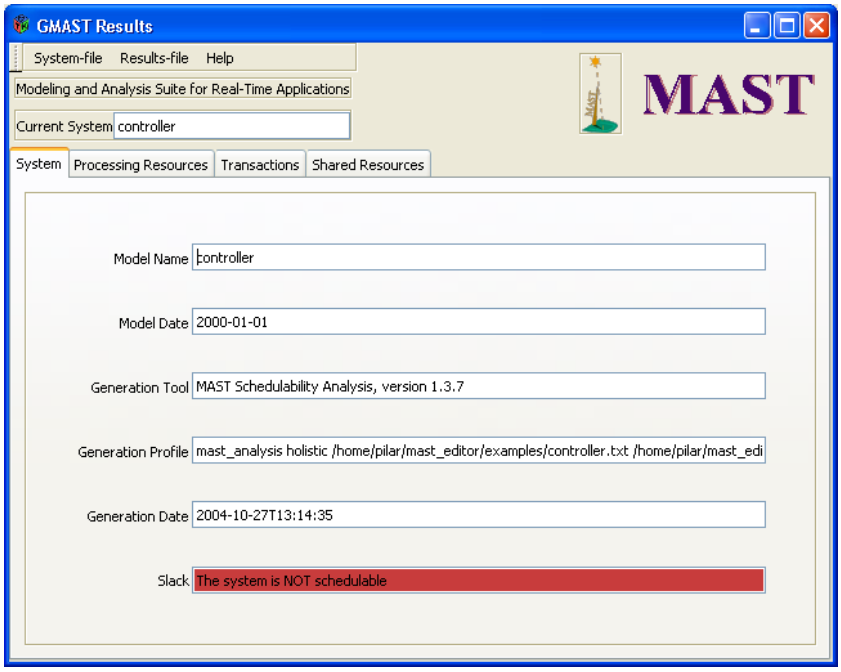

Figura 1.2: Pantalla de visualización de resultados

1.2. Sistemas de Tiempo Real

Este apartado se dedica a explicar algunos conceptos referentes a los sistemas de tiempo real con los que es importante familiarizarse para poder comprender y seguir el desarrollo de este trabajo. Primeramente, podemos definir un *sistema de tiempo real* como un sistema de procesamiento de información el cual tiene que responder a estímulos de entrada generados externamente en un intervalo de tiempo finito y específico. De esto se deduce que en un sistema de tiempo real las respuestas correctas dependen no sólo de los resultados sino también del instante en que se generan y que el hecho de no responder a tiempo es tan malo para el sistema como una mala respuesta.

La ejecución de cada una de las actividades que se realizan en un sistema de estas características es disparada por la llegada de un estímulo llamado *evento*, y la actividad propiamente dicha, que se realiza debido a la llegada de un evento, se conoce como la *respuesta*. Estas respuestas están formadas por diferentes actividades o tareas, cada una de las cuales constituye una *acción*. La ejecución de la respuesta puede tener impuestos ciertos requerimientos temporales que debe cumplir. En muchos casos, estos requerimientos se refieren al máximo tiempo del que se dispone para dar la respuesta desde la llegada del evento. Este tipo de requerimiento se conoce con el nombre de *plazo (deadline)*.

Generalmente, los sistemas de tiempo real constan de un conjunto de actividades que son planificables de manera independiente. Los eventos que llegan al sistema pueden generar la realización de actividades periódicas (se ejecutan a intervalos regulares) o aperiódicas (se ejecutan a intervalos no regulares), y pueden tener plazos estrictos (el sistema se considera que falla si no cumple con alguno de sus plazos) o no estrictos (el sistema puede incumplir alguno de sus plazos de vez en cuando, produciendo así una disminución de las prestaciones o de la calidad del sistema, pero no un fallo total).

Así, desde el punto de vista de pérdida de plazo global, podemos distinguir entre dos tipos de sistema:

- **Sistema de Tiempo Real estricto (*hard real-time system)*:** Cuando la pérdida de un plazo es crítica y provoca el fallo del sistema. Los factores realmente importantes son la planificabilidad (habilidad para cumplir *todos* los plazos) y el tiempo de respuesta en el peor caso.
- **Sistema de Tiempo Real no estricto (*soft real-time system*):** Cuando la pérdida de un plazo o de un número de ellos no es muy importante, aunque el sistema debe cumplir, en promedio, un cierto porcentaje de veces con su plazo.

Entre las principales características de los sistemas de tiempo real podemos citar las siguientes:

- Dependencia del Tiempo: se pueden ejecutar tareas o actividades en respuesta a estímulos externos o de forma periódica.
- Fiabilidad y Tolerancia a Fallos: deben funcionar aún en presencia de fallos o averías parciales.
- Concurrencia: se componen de un conjunto de actividades realizándose en paralelo.
- Eficiencia: es deseable una implementación eficiente.
- Complejidad y Tamaño: suelen ser grandes y por lo tanto es deseable usar técnicas modulares.
- Interacción con el Hardware: para su conexión con el exterior.
- Manejo de Medidas y Datos: adquiridos del exterior mediante interfaces.

La planificación de los recursos del sistema es uno de los problemas centrales en la construcción de los sistemas de tiempo real. Es tal su importancia en los sistemas actuales, que la selección de una política de planificación determina en gran medida la arquitectura de software y la realización del sistema. La planificación síncrona [18] ha sido la más empleada tradicionalmente. Se caracteriza porque la ejecución de las tareas se activa de forma sincronizada con un reloj. El ejecutivo cíclico es el método más empleado, y consiste en un procedimiento que activa cíclicamente un conjunto de tareas según un orden prefijado en un plan de ejecución. El carácter determinista de este método es su mayor ventaja. La dificultad de realizar y mantener el plan de ejecución es su mayor inconveniente.

La planificación basada en mecanismos expulsores y no expulsores por prioridades (estáticas o dinámicas) [18] constituye una alternativa más adecuada en sistemas complejos. En la planificación no expulsora, si una tarea comienza su ejecución no se detiene hasta que termina, aunque haya otras tareas de mayor prioridad esperando. Esto puede producir un retraso significativo para las tareas de alta prioridad, que es posible evitar con una planificación expulsora en la que una tarea de alta prioridad expulsa del procesador a una de baja prioridad. En el caso de planificación con mecanismos expulsores por prioridades fijas, la prioridad de la tarea la fija el usuario y no cambia hasta que éste vuelve a fijar una nueva prioridad. En cambio, en las políticas basadas en prioridades dinámicas, éstas pueden variar dinámicamente en función de lo cerca que se encuentre una tarea de su plazo o en función de alguna relación entre el tiempo que necesita para terminar su trabajo y el plazo que tiene asignado.

La planificación de las actividades del sistema se debe realizar cumpliendo los siguientes requisitos:

- Conseguir utilizaciones altas de los recursos, garantizando el cumplimiento de los plazos.
- Predecibilidad: Referida a la facilidad de analizar el comportamiento temporal del sistema (sobre todo en lo que respecta a la respuesta temporal de peor caso).
- Asegurar una respuesta rápida a los eventos aperiódicos garantizando el cumplimiento de los plazos de todas las tareas del sistema.
- Producir una baja sobrecarga (overhead) debida a la planificación en los procesadores.
- Mantener la estabilidad en situaciones de sobrecarga transitoria, asegurando la verificación de los plazos de las tareas críticas en esas condiciones.

Finalmente, como ejemplos de sistemas de tiempo real podemos mencionar aquellos destinados a aplicaciones como los sistemas de control empotrados (controladores de robots, máquinas industriales, etc.), la transmisión y procesado de imagen (visión artificial de tiempo real), sistemas de control y navegación de vehículos (automóviles, aviones, o naves espaciales), etc., ya que todos ellos presentan unas restricciones de tiempo real, debido a que no sólo hay que procesar la información a tiempo sino que también se deben generar los resultados dentro de su plazo.

1.3. Descripción general del entorno MAST

MAST (Modeling and Analysis Suite for Real-Time Applications) [1][5][7] es un entorno software que ofrece un conjunto de herramientas de código abierto para el modelado, diseño y análisis temporal de sistemas de Tiempo Real, que está siendo desarrollado por el Departamento de Electrónica y Computadores de la Universidad de Cantabria.

El entorno MAST proporciona componentes para el modelado de recursos hardware tales como procesadores, redes, dispositivos o temporizadores, así como de recursos software como *threads* (procesos ligeros), procesos, servidores o controladores que constituyen la plataforma del sistema, los componentes lógicos de la aplicación como, por ejemplo, clases, métodos o procedimientos, y mecanismos de sincronización como semáforos, monitores o mutex, y, finalmente, las situaciones de tiempo real que describen la carga de trabajo y los requerimientos temporales que corresponden a un modo de ejecución determinado.

Este modelo permite una descripción detallada del sistema, incluyendo los efectos de sincronización de eventos o mensajes, arquitecturas distribuidas y multiprocesador, así como la sincronización de recursos compartidos. La representación de un sistema usando este modelo puede ser analizada mediante un conjunto de herramientas que han sido desarrolladas dentro de MAST, incluyendo el análisis de planificabilidad para la comprobación de los requerimientos temporales estrictos y mediante herramientas futuras tales como la simulación de eventos discretos para requerimientos temporales no estrictos.

Dentro de las herramientas desarrolladas hasta el momento se incluyen el análisis de la planificabilidad de respuesta temporal de peor caso, el cálculo de tiempos de bloqueo, el cálculo de holguras *(slacks)*, la asignación de prioridades optimizada y herramientas de simulación del comportamiento temporal del sistema.

MAST define un modelo abierto para la descripción de sistemas de tiempo real conducidos o gobernados por eventos diseñado para ser ampliado con el fin de de soportar nuevas características en los sistemas. MAST se ha diseñado para tratar la mayoría de sistemas de tiempo real creados a partir de sistemas operativos y lenguajes estándar comerciales (por ejemplo POSIX y Ada). Esto incluye a los sistemas planificados con prioridades fijas o con prioridades dinámicas. Dentro de las prioridades fijas, se permiten diferentes estrategias de planificación, incluyendo planificación expulsora y no expulsora, rutinas de servicio a la interrupción, planificación de servidores esporádicos y de servidores de muestreo periódicos.

El modelo MAST se diseña para tratar tanto sistemas monoprocesador como sistemas multiprocesador, estos últimos con asignación estática de tareas a procesadores. En ambos casos, se pone énfasis en describir sistemas gobernados por eventos, en los cuales cada tarea puede generar múltiples eventos a su conclusión. Una tarea puede activarse por una combinación condicional de uno o más eventos. Los eventos externos que llegan al sistema pueden ser de diferentes tipos: periódicos, aperiódicos no acotados, esporádicos, a ráfagas o singulares (solo llegan una vez).

La metodología de modelado facilita la descripción independiente de los parámetros de sobrecarga *(overhead)* como la sobrecarga de los procesadores (incluyendo la sobrecarga de los servicios temporizados), sobrecarga de red y sobrecarga de los controladores de red. Esto libera al usuario de la necesidad de incluir todos estos *overheads* en el modelo de aplicación actual, simplificádolo así y eliminando mucha redundancia.

El modelo soporta tanto requerimientos temporales estrictos como no estrictos ya que es bastante frecuente encontrar sistemas con ambas clases de requerimientos. Por requerimientos de tiempo real estrictos se consideran requerimientos de plazos *(deadlines)* y fluctuaciones máximas de salida *(Max Output Jitter)*. Entre los requerimientos de tiempo real no estrictos, MAST considera plazos no estrictos y tasas de pérdida de plazos máxima *(Max Miss Ratio)*.

MAST incluye herramientas de análisis de planificabilidad que usan las últimas técnicas basadas en *offsets* para aumentar los resultados del análisis [14]. Estas técnicas son mucho menos pesimistas que las anteriores técnicas de análisis para sistemas distribuidos las cuales están también incluidas con el fin de completar dichas herramientas. También se incluyen herramientas de asignación de prioridades optimizadas. El conjunto de herramientas MAST es de código abierto (bajo licencia pública general GNU) y extensible; esto significa que otros equipos de investigación pueden realizar modificaciones. Las primeras versiones se dedicaron a los sistemas de prioridades fijas, pero actualmente ya existe soporte para sistemas planificados dinámicamente.

Con el fin de superar la complejidad de las actividades de modelado, se definen perfiles en niveles superiores de abstracción para algunas metodologías o entornos software específicos como aplicaciones orientadas a objetos, aplicaciones que usan el modelo de concurrencia del lenguaje Ada o las basadas en componentes software. Los

componentes de modelado de alto nivel en cada perfil implementan la semántica de abstracciones de metodología específicas y usándola no solo es más fácil construir el modelo de tiempo real si no que se hace más afín al entorno de desarrollo. Estos perfiles se explicarán más detalladamente en el apartado 1.6, “Perfiles de modelado de nivel superior”.

1.4. Modelo MAST para Sistemas de Tiempo Real

Un sistema de tiempo real se modela en MAST como un conjunto de *transacciones* [7]. Cada transacción se activa por uno o más eventos externos y representa un conjunto de *actividades* que serán ejecutadas en el sistema. Las actividades generan eventos que son internos a la transacción y que, a su vez, activarán otras actividades. En el modelo existen estructuras especiales manejadoras de eventos para utilizarlas de distintas maneras. Los eventos internos pueden tener requerimientos temporales asociados a ellos.

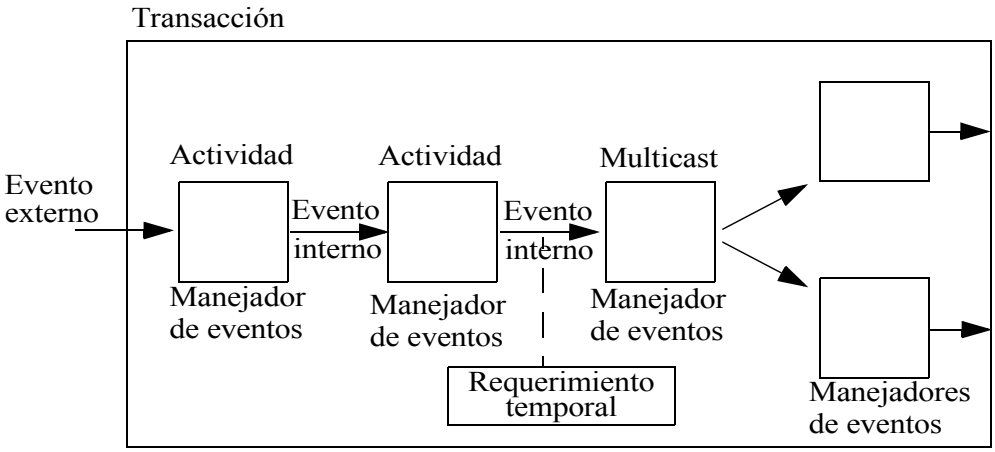

Figura 1.3: Esquema detallado de una transacción [1]

Las *transacciones* se representan mediante grafos mostrando el flujo de *eventos* entre los *manejadores de eventos*, que son representados como cajas dentro del grafo. La figura 1.3 [7] muestra un ejemplo de una transacción de un sistema. Esta transacción está activada por un solo evento externo. Después de haber ejecutado dos actividades, un manejador de eventos *multicast* genera dos eventos que activan a las dos últimas actividades en paralelo.

En el modelo MAST existen dos clases de *manejadores de eventos*:

- El *manejador estructural (Structural Handler)*: Sólo manipula los eventos y no consume recursos o tiempo de ejecución. El manejador de eventos *multicast* de la figura 1.3 es un ejemplo de este tipo de manejador.
- La *actividad (Activity)*: Representa la ejecución de una operación, por ejemplo, un procedimiento o función en un procesador, o la transmisión de un mensaje en una red.

Los elementos que definen una actividad están descritos en la figura 1.4. Podemos comprobar que cada actividad está activada por un evento *entrante*, y genera un evento *saliente* cuando se haya completado. Si se necesita generar algún evento intermedio, la actividad será dividida en partes. Cada actividad ejecuta una *Operación* que representa un

trozo de código para ejecutar en un procesador o un mensaje para ser enviado a través de una red. Una operación puede tener una lista de *Recursos compartidos* que precisa utilizar de un modo mutuamente exclusivo.

La actividad es ejecutada por un *Servidor de planificación*, que representa una entidad planificable en el *Planificador* al cual se asigna. Este planificador que tiene una política de planificación asociada pertenece a un *Recurso de Procesado* (un procesador o una red), si bien a la hora de modelar planificadores jerárquicos el esquema es más complejo. Un claro ejemplo del modelo de un servidor de planificación en un procesador es una tarea o *thread*. Una tarea puede ser la responsable de ejecutar varias actividades y, de este modo, las operaciones asociadas (procedimientos). Al servidor de planificación se le asigna un objeto de *parámetros de planificación* que contiene la información de políticas y parámetros de planificación utilizados. Algunos *Recursos de procesado* pueden tener referencias a *Temporizadores del sistema* o a *Drivers de red*, que representan diversos efectos de *overhead* dentro del sistema.

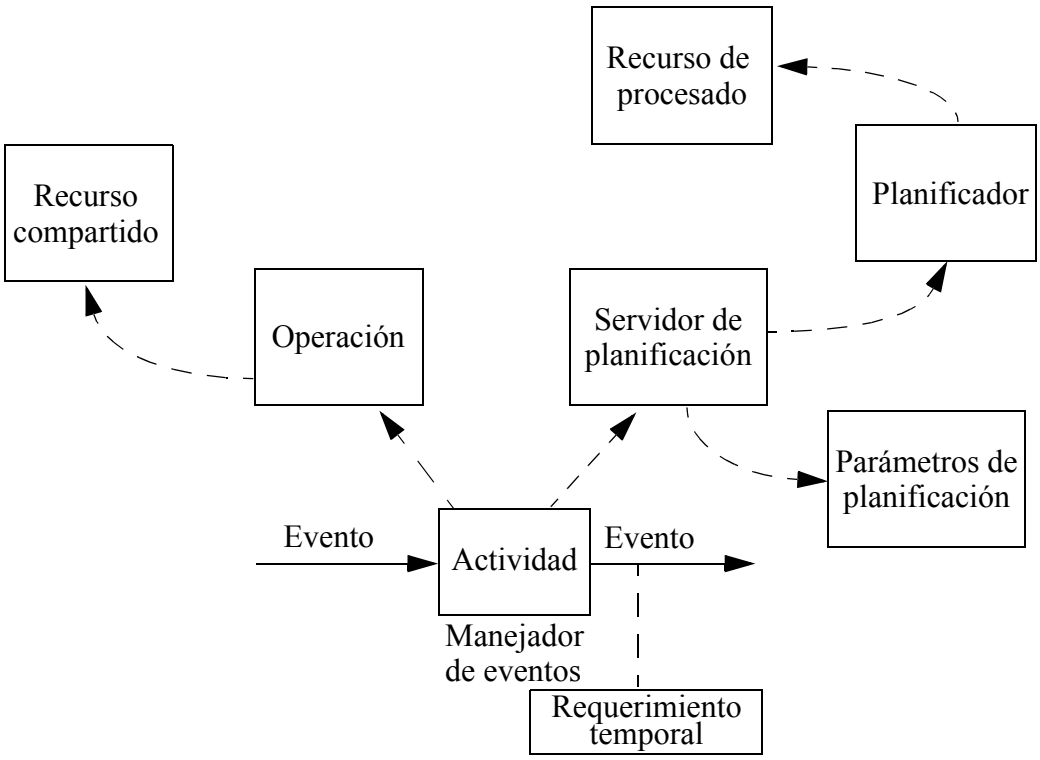

Figura 1.4: Elementos que definen una actividad [1]

Para tener una idea más precisa de cada uno de los conceptos mencionados anteriormente, remitimos al lector al siguiente capítulo, el cual se dedica a describir de manera detallada cada uno de los elementos del modelo MAST, así como cada una de las clases definidas hasta la fecha y atributos de los que constan.

1.5. Herramientas MAST

Los principales componentes del entorno MAST aparecen en la figura 1.5. La descripción de un sistema MAST se especifica a través de un fichero ASCII que sirve como entrada de las herramientas de análisis. Un parser convierte la descripción ASCII en una estructura de datos que es usada por las herramientas. El parser se ha construído

utilizando *ayacc* [10] que es un equivalente para el lenguaje Ada del popular generador de parsers *yacc*. Esto nos da un alto grado de flexibilidad para añadir nuevas capacidades al lenguaje de descripción.

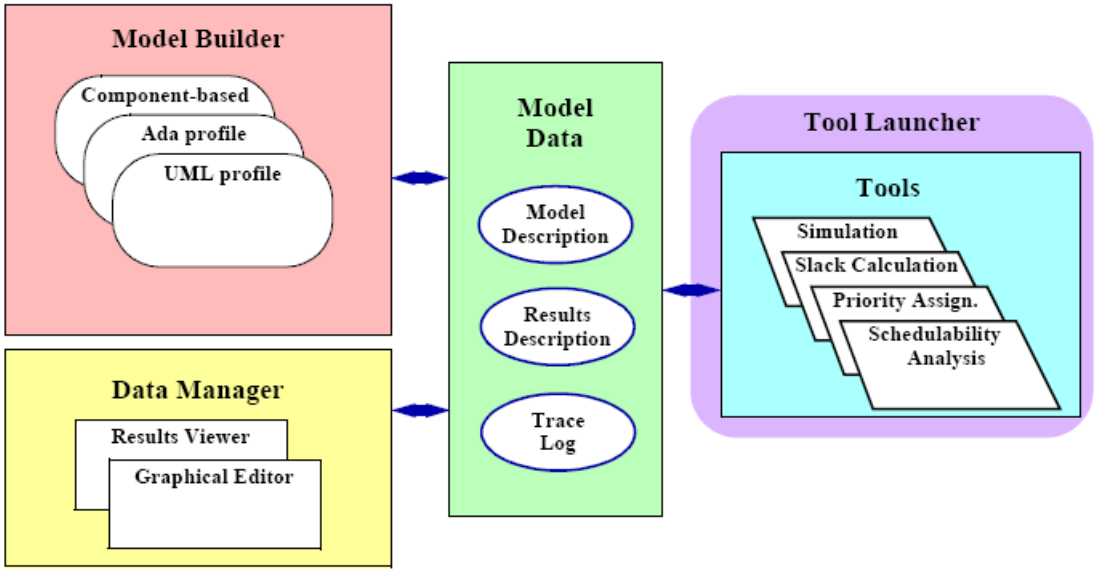

Figura 1.5: Entorno de herramientas MAST [7]

La estructura de datos generada por el parser es construida utilizando técnicas orientadas a objectos, para hacerla fácilmente extensible. Las herramientas de análisis operan sobre esta estructura de datos y son capaces de utilizar diferentes técnicas de análisis de planificabilidad de peor caso generando un conjunto de resultados con el comportamiento temporal del sistema.

Antes de describir las herramientas que podemos aplicar para analizar el modelo del sistema hemos de considerar los diferentes tipos de sistemas soportados por las herramientas. Se comprueban tanto las restricciones como la consistencia a efectos de llamar a las herramientas apropiadamente. MAST distingue entre los siguientes tipos de sistemas atendiendo a los criterios impuestos por los algoritmos implementados por las herramientas:

- Monoprocesador o Multiprocesador: Algunas herramientas soportan sólo un recurso de procesado y otras soportan varios incluyendo sistemas multiprocesador y sistemas distribuidos.
- En referencia a la arquitectura estructural de la aplicación, los sistemas pueden clasificarse como sigue (cada categoría incluye las anteriores):
  - Sólo Transacciones Simples: el sistema está compuesto de transacciones con un sólo manejador de eventos que alberga una actividad.
  - Transacciones Lineales: el sistema contiene transacciones construidas con manejadores de eventos lineales, incluyendo Divisores de Frecuencia (*Rate Divisors)*, Retrasos Locales *(Delays)* y Retrasos Globales *(Offsets)*.
  - Transacción No Lineal: las transacciones del sistema pueden incluir cualquier manejador de eventos, incluyendo la activación múltiple de eventos.

El conjunto de herramientas de análisis es capaz de usar los siguientes tipos de técnicas de análisis de planificabilidad de peor caso para generar un conjunto de resultados con el comportamiento temporal del sistema:

- No Lineal *(Non-Linear)*: Esta técnica está actualmente bajo implementación y puede aplicarse a sistemas Multiprocesador y con Transacciones No Lineales [9]. Está basada en la transformación del sistema en un modelo equivalente en el cual se pueden aplicar las técnicas RMA (Rate Monotonic Analysis) existentes para analizar sistemas de tiempo real distribuidos.
- Basada en Offsets Optimizada *(Offset Based Optimized)*: Corresponde a la técnica presentada en [4], y es aplicable a los sistemas Multiprocesador y con Transacciones Lineales. Esta es la mejor técnica disponible actualmente para sistemas distribuidos lineales.
- Basada en Offsets *(Offset Based)*: Como la anterior, es aplicable a sistemas Multiprocesador y con Transacciones Lineales. Fue presentada en [14]. Esta técnica obtiene mejores resultados que el análisis de tipo *Holistic*, pero no explota al máximo los beneficios de los controles de precedencia en la transacción, en oposición a la técnica basada en offsets completa.
- *Holistic*: Esta es la clásica técnica para sistemas distribuidos [22], que soporta sistemas Multiprocesador y con Transacciones Lineales. Aunque es claramente menos exacta que las técnicas basadas en offsets, se incluye para hacer el conjunto de herramientas más completo y para disponer de un referente para la comparación de resultados.
- Prioridades Variables *(Varying Priorities)*: La técnica desarrollada en [6] puede ser aplicada en sistemas Monoprocesador y con Transacciones Simples, en los cuales puede haber prioridades predeterminadas permanentes. Soporta tareas que se descomponen en varias subtareas de prioridades distintas que se ejecutan en serie.
- Método Clásico de la Prioridad al más Frecuente *(Classic Rate Monotonic)*: Este es el clásico análisis de respuesta temporal desarrolado por Joseph y Pandya [11], y extendido a plazos arbitrarios por Lehoczky [12]. Soporta sólo sistemas Monoprocesador y con Transacciones Simples sin prioridades predeterminadas permanentes.
- EDF Monoprocesador *(Earliest Deadline First Monoprocessor)*: Esta técnica es aplicable a sistemas monoprocesador con un planificador EDF (soportado también interrupciones de prioridades fijas).
- *EDF Within Priorities*: Técnica aplicable a sistemas con un planificador primario de prioridades fijas y uno o más planificadores EDF secundarios.

Todas las técnicas usadas incluyen capacidades de análisis para plazos arbitrarios, manejo de fluctuaciones de entrada y salida, eventos periódicos, esporádicos y aperiódicos, y diferentes políticas de planificación de prioridades fijas (como expulsoras o no expulsoras), dinámicas, servidores de muestreo periódico, esporádicos, etc. Los tiempos de bloqueo relativos al uso de recursos compartidos y de secciones no expulsables son calculados automáticamente.

El conjunto de herramientas MAST proporciona una opción para calcular automáticamente un conjunto optimizado de prioridades. MAST incluye varios algoritmos de asignación de prioridad [9]:

- Monoprocesador: Si los plazos están al final del período o antes, se realiza una asignación de plazo monotónica.
- HOPA (Heuristic Optimum Priority Assignment): Es un algoritmo de optimización basado en la distribución de plazos globales de cada transacción entre las diferentes acciones que componen esa transacción.
- Templado Simulado *(Simulated Annealing)*: Es una técnica de optimización general para funciones discretas que ha sido usada anteriormente para resolver problemas similares.
- *Deadline Assignment*: Para tareas EDF, determina plazos iguales a los especificados como requerimientos temporales y asigna los niveles de expulsión de las tareas en el orden adecuado, esto es, usando el valor obtenido de restar del plazo la fluctuación (en orden inverso).

Todas las técnicas de asignación de prioridades operan realizando iteraciones del análisis. La figura 1.6 contiene los diferentes pasos seguidos por las herramientas, mostrando explícitamente la iteración.

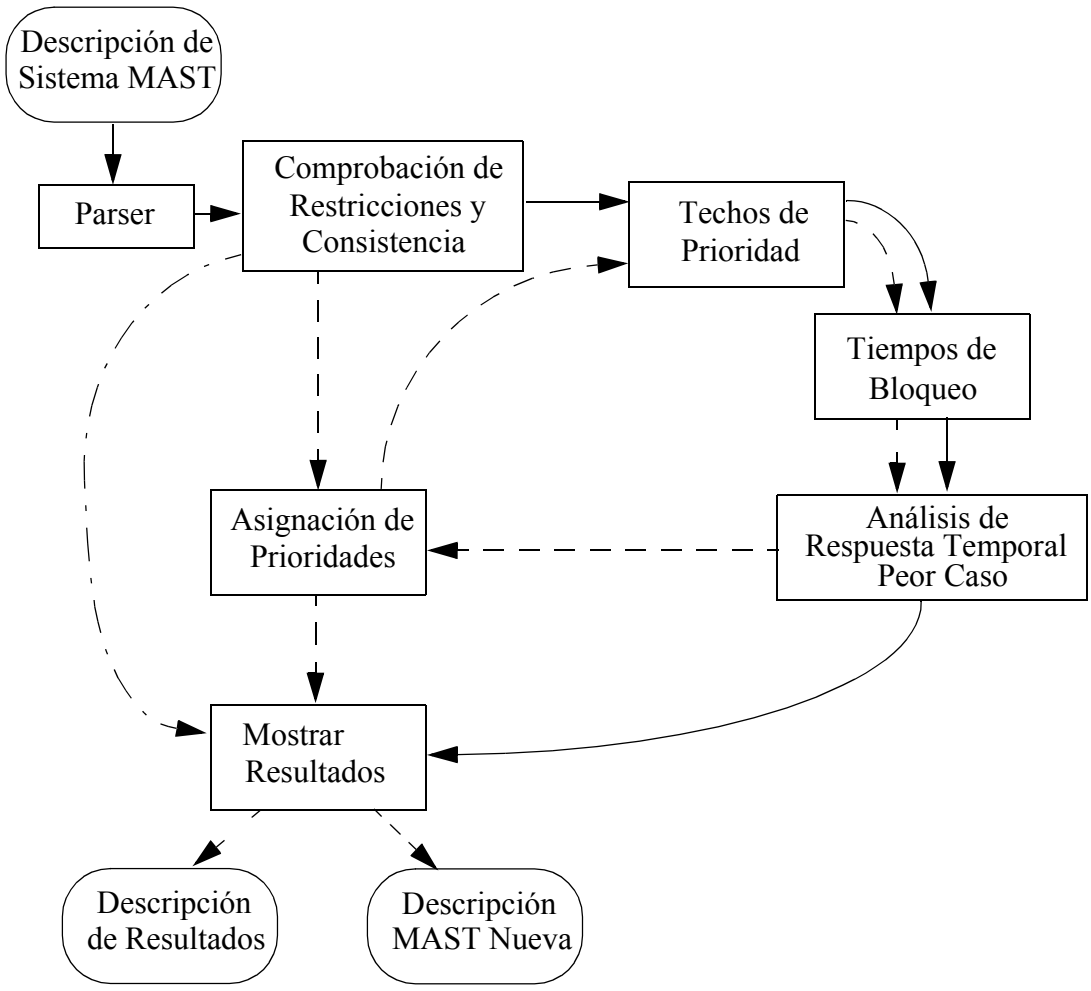

Figura 1.6: Pasos internos en las herramientas de análisis [7]

Los resultados temporales de las herramientas de análisis de peor caso pueden ser comparados con los requerimientos temporales para determinar la planificabilidad.

Finalmente, hay una opción interesante en las herramientas de análisis de planificabilidad MAST, relacionadas con el cálculo de holguras *(slacks)*. La holgura es el porcentaje en el que todas las operaciones involucradas en un sistema o en una parte de él pueden ser incrementadas sin hacer el sistema no planificable o decrementadas para que lo sea. Esto significa que: si el *slack* es positivo, es el porcentaje por el que todos los tiempos de ejecución de las operaciones puede ser incrementado mientras el sistema siga siendo planificable; si es negativo, es el porcentaje por el que esos tiempos de ejecución han de ser reducidos para hacer el sistema planificable; si es cero, significa que el sistema es planificable, pero sin ningún margen de crecimiento o seguridad. Se pueden calcular los *slacks* de operaciones aisladas, transacciones, recursos de procesado y de todo el modelo general.

El lenguaje de implementación del parser y de las herramientas de análisis es Ada, debido a que ofrece un soporte completo de funcionalidades orientadas a objetos y a sus construcciones que facilitan la creación de software fiable.

Asimismo, es posible describir una perspectiva de tiempo real del sistema utilizando una herramienta UML (Unified Modeling Language) [2] añadiendo las clases y objetos necesarios para describir el comportamiento de tiempo real del sistema. La aplicación diseñada está ligada con una perspectiva de tiempo real para obtener una descripción completa del sistema y de su comportamiento y requerimientos temporales. Se ha desarrollado una herramienta automática dentro de la herramienta Rose UML CASE [13], que extrae la descripción de tiempo real del sistema de la descripción UML, generando el archivo de descripción MAST. Con esta aproximación no se necesita una estructura especial, pero el diseñador debe incorporar la persectiva de tiempo real dentro de la descripción UML.

1.6. Perfiles de modelado de nivel superior

Como se ha visto anteriormente, MAST ofrece un conjunto completo de componentes de modelado básicos. Estos son sencillos y cercanos a los elementos hardware y software que son importantes en el comportamiento temporal del sistema. Para sistemas algo complejos el modelo se compone de cientos o miles de componentes básicos, lo que implica que validar el modelo puede ser bastante difícil. Con el fin de evitar la complejidad del modelado, se han definido perfiles en un nivel superior de abstracción de modelado para algunos entornos o metodologías específicos. Los componentes de modelado de nivel superior definidos en cada perfil implementan la semántica de las abstracciones específicas de la metodología asociada y utilizándolos el modelo de tiempo real no sólo es más fácil de construir sino que también se acerca más el entorno de desarrollo. El uso de herramientas CASE adaptadas a la composición de modelos de los componentes de modelado definidos en estos perfiles ha demostrado ser útil a la hora de construir del modelo MAST del sistema automáticamente. En este momento hay varios perfiles definidos:

- UML_MAST: Perfil para modelar sistemas que se diseñan usando técnicas de representación orientadas a objetos y herramientas CASE. Incluye soporte de herramienta CASE para crear modelos y generar el archivo MAST.

- ADA_MAST: Perfil para modelar aplicaciones de tiempo real distribuidas, creadas usando componentes lógicos Ada básicos. La metodología se ha diseñado para facilitar el modelado de sistemas escritos para plataformas Ada 95.
- CBSE_MAST: Perfil para modelar sistemas de tiempo real basados en componentes software. La metodología hace posible construir el modelo del sistema añadiendo modelos de los componentes al modelo de la plataforma CBSE usado.

1.7. Objetivos de este trabajo

Como se ha comentado anteriormente, desarrollar un sistema con requerimientos de tiempo real es una labor compleja y costosa. Por esta razón es importante disponer de herramientas de análisis y simulación precisas que sirvan como asistentes en el diseño y modelado de sistemas que, una vez analizados y verificado su comportamiento, puedan ser implementados en la práctica.

Dentro del grupo de investigación del departamento se está desarrollando el software MAST que es un entorno para el modelado y análisis de sistemas de tiempo real. MAST proporciona un conjunto de herramientas para la configuración, análisis y simulación del sistema como se explicó en el apartado 1.5.

El objetivo general del presente trabajo es implementar y verificar una de las herramientas MAST, el editor gráfico. Para la consecución de la finalidad global del trabajo se establecieron los siguientes objetivos específicos:

- Diseñar e implementar una interfaz que represente gráficamente la disposición de los elementos que conforman el modelo del sistema, así como las relaciones existentes entre los diferentes elementos.
- Desarrollar un conjunto de ventanas y menús que permita configurar los elementos del modelo MAST y sus respectivos atributos.
- Implementar la aplicación del editor gráfico para que sea capaz de generar automáticamente la descripción MAST de un sistema en formato ASCII a partir de los parámetros introducidos por el usuario.
- Diseñar la aplicación del editor gráfico para que lea y guarde en archivo los parámetros y elementos del modelo así como sus atributos gráficos.
- Permitir a los usuarios no interesados en utilizar la descripción UML usar las herramientas de análisis de MAST.
- Conseguir que la aplicación sea fácilmente extensible con el fin de incluir posibles modificaciones futuras de la especificación MAST.
- Completar el trabajo desarrollado en [22] consiguiendo implementar una herramienta integrada e independiente que permita modelar y analizar la planificabilidad de un sistema de tiempo real siguiendo la metodología MAST.

1.8. Síntesis de la memoria

En esta memoria se recoge, a modo de resumen, buena parte del trabajo realizado estructurado en cinco capítulos, además de la presente introducción:

En el capítulo 2 se enuncian y explican de manera detallada todos los elementos que componen el modelo MAST y sus parámetros asociados. El lector deberá familiarizarse con la terminología empleada en esta sección ya que dichos conceptos serán manejados a lo largo del desarrollo del trabajo y de la redacción de la memoria. Es importante comprender la estructura del modelo y las relaciones existentes entre los diferentes elementos que lo componen, con el fin de utilizar y configurar los parámetros del modelo de forma correcta y así poder efectuar un análisis de los sistemas adecuado.

En el capítulo 3 se describe detalladamente la estructura de la interfaz gráfica, a través de la descripción de su manual de usuario que indica cómo utilizar la aplicación. Se ha redactado de este modo con la intención de que pueda ser utilizado de forma independiente como guía de usuario del software.

El capítulo 4 se ha dedicado a explicar los pasos seguidos en la implementación de la interfaz gráfica, incluyendo el diseño y la programación de los módulos software que la componen y mencionando las herramientas utilizadas durante dicho proceso.

En el capítulo 5 se muestran algunos ejemplos de sistemas de Tiempo Real que han sido modelados mediante MAST. Dichos ejemplos se han incluido a modo orientativo, como referencia para aquellos usuarios que quieran modelar y analizar sistemas de Tiempo Real utilizando las herramientas MAST.

Finalmente, en el capítulo 6 se realiza un breve resumen del trabajo realizado en forma de conclusiones. Además se citan posibles mejoras, ampliaciones y líneas futuras de trabajo que complementen al que se ha desarrollado en el presente proyecto de investigación.

2. Elementos del Modelo MAST

2.1. Introducción

Anteriormente se ha explicado cómo MAST proporciona un conjunto de herramientas que permite a los investigadores que desarrollan aplicaciones de tiempo real verificar el comportamiento temporal de dichas aplicaciones. Para ello es preciso construir el modelo MAST del sistema en cuestión y realizar posteriormente el análisis utilizando las herramientas disponibles. Por este motivo, es fundamental conocer los distintos elementos que componen dicho modelo y sus características con el fin de diseñar y analizar los sistemas eficientemente.

Este capítulo se dedica a describir en detalle la clasificación de los distintos elementos definidos actualmente en el modelo MAST y sus respectivos atributos. Esta clasificación puede ser ampliada o extendida para incorporar otras características de los sistemas de tiempo real.

En los siguientes apartados se enumeran las clases de elementos de los que consta MAST y, entre paréntesis, se expone la nomenclatura utilizada por el modelo MAST [1], [5] y [7].

2.2. Recursos de procesado *(Processing Resources)*

Representan recursos con la capacidad de ejecutar actividades abstractas. Se utiliza para modelar procesadores, equipos, redes de comunicación, etc. que ejecutan las operaciones requeridas en el sistema. Sus atributos comunes son los siguientes:

- Nombre *(Name)*: Utilizado como identificador del recurso de procesado.
- Factor de velocidad *(Speed Factor)*: Todos los tiempos de ejecución de las operaciones se expresan en unidades de tiempo normalizadas. El tiempo de ejecución real se obtiene dividiendo el tiempo de ejecución normalizado entre el factor de velocidad.

Hasta el momento se han definido dos clases, los componentes de tipo **Procesador** *(Processor)* que representan procesadores, coprocesadores o equipos embarcados que ejecutan actividades formuladas como código de programa y los componentes de tipo **Red** *(Network)* que representan redes de comunicación cuya actividad consiste en transferir información entre procesadores. Ambas clases son abstractas y las únicas clases concretas que actualmente están definidas como extensiones de las anteriores son respectivamente:

- Procesador regular (*Regular Processor*): Representa un procesador con los overheads básicos asociados con sus servicios de interrupción hardware.Tiene los siguientes atributos adicionales:

- Rango de prioridades de interrupción *(Max Interrupt Priority, Min Interrupt Priority):* Válido para actividades planificadas por rutinas de servicio a la interrupción.
- Overheads (peor, medio y mejor) asociados al cambio de rutina de servicio a la interrupción (*ISR Switch Overheads (Worst, Average, Best))*.
- Temporizador del sistema *(System Timer)*: Referencia al temporizador del sistema descrito en el apartado 2.3, "Temporizadores del sistema (System Timers)" que influye en el *overhead* de las actividades asociadas al temporizador del sistema *(System Timed Activities)*, ver apartado 2.14, "Manejadores de eventos (Event Handlers)".

- Red basada en Paquetes (*Packet Based Network*): Representa una red que utiliza algún protocolo de tiempo real basado en paquetes no expulsables para transmitir mensajes en la red. Hay redes que soportan prioridades en sus protocolos estándar (p. ej., CAN bus) y otras redes que precisan de un protocolo adicional de nivel superior que trabaja por encima de los protocolos estándar (p. ej., ethernet, líneas serie). Esta clase tiene los siguientes atributos adicionales:
  - Tipo de transmisión (*Transmission Kind*): Indicará si la comunicación es Simplex, Half Duplex o Full Duplex.
  - Ancho de banda (*Throughput)*: Ancho de banda normalizado en bits por unidad de tiempo.
  - Máximo bloqueo (*Max Blocking*): El máximo bloqueo es causado por la no expulsión de los paquetes de mensaje. Normalmente, tiene el mismo valor del tiempo de transmisión de paquete máximo, pero su valor por defecto es cero, para el caso en que el *overhead* de la red es despreciable.
  - Tamaños de paquete máximo y mínimo (*Max Packet Size* y *Min Packet Size*): Describen la cantidad de información incluida en un paquete, excluyendo la información de protocolo. El tamaño máximo se usa para calcular el número de paquetes en los que se trocea un mensaje, calculado como el entero máximo resultante de dividir el tamaño del mensaje entre el tamaño máximo del paquete. Este número se multiplica por el tiempo de overhead del paquete del planificador para calcular la sobrecarga de la red. El tamaño mínimo se usa para calcular el periodo más corto de los overheads asociados con la transmisión de cada paquete, y por lo tanto tiene un gran impacto en la sobrecarga causada por los controladores de red en los procesadores que la utilizan.
  - Tiempos de transmisión máximo y mínimo de paquete (*Max Packet Transmission Time* y *Min Packet Transmission Time*): El tiempo máximo se utiliza para el cálculo del modelo de overhead de la red, representa un tiempo de bloqueo de la red ya que se supone que los paquetes no son expulsados. El tiempo mínimo representa el período más corto de los *overheads* asociados a la transmisión de cada paquete y, por tanto, tiene un gran impacto en

el overhead que causan los controladores de red (*Network drivers*) en los procesadores por el uso de ésta.

- Lista de controladores (*List of Drivers*): Una lista de referencias a los controladores de red descritos en el apartado 2.4, "Controladores de red (Network Drivers)", que contienen el modelo de *overhead* del procesador asociado con la transmisión de mensajes a través de la red.

2.3. Temporizadores del sistema *(System Timers)*

Representan los distintos *overheads* asociados con el modo en que el sistema maneja los eventos temporizados. Existen dos clases:

- Reloj despertador (*Alarm Clock*): Representa sistemas cuyos eventos temporizados son activados por una interrupción del temporizador hardware. El temporizador es programado para generar la interrupción en el instante de tiempo del evento temporizado más próximo. Por consiguiente, cada uno puede tener su propia interrupción. Esto conlleva un overhead asociado a la interrupción. Sus atributos son:
  - Overheads (peor, medio y mejor) (*Overheads (Worst, Average, Best)*): *Overhead* asociado a la interrupción del temporizador, suponiendo que se ejecuta a la más alta prioridad de interrupción.
- Reloj periódico (*Ticker*): Representa a un sistema que tiene un reloj periódico, como puede ser una interrupción periódica que llega al sistema. Cuando esta interrupción llega, todos los eventos temporizados cuyo tiempo de expiración haya pasado son activados. Otros eventos no temporizados son manejados en el instante en que se generan. En este modelo, el *overhead* introducido por la interrupción del temporizador es localizado en una sola interrupción periódica, pero se introduce una ligera fluctuación para todos los eventos temporizados, debido a que la máxima resolución temporal es el período del reloj. Los atributos son:
  - Overheads (peor, medio y mejor) (*Overheads (Worst, Average, Best)*): *Overhead* asociado a la interrupción del temporizador, suponiendo que se ejecuta a la más alta prioridad de interrupción.
  - Período *(Period*): Período de la interrupción del reloj.

2.4. Controladores de red *(Network Drivers)*

Representan las operaciones ejecutadas en un procesador como consecuencia de la transmisión o recepción de un mensaje o un paquete a través de la red. Existen varios tipos:

- Controlador de paquetes (*Packet Driver*): Representa un controlador que es activado en cada transmisión o recepción de un mensaje. Sus atributos son:
  - Servidor de paquetes (*Paquet Server*): Es una referencia al servidor de planificación que ejecuta el controlador (que a su vez contiene una referencia al procesador y a los parámetros de planificación).

    - Operación de envío de paquete *(Paquet Send Operation*): Operación ejecutada cada vez que un paquete es enviado.
    - Operación de recibo de paquete *(Paquet Receive Operation*): Operación ejecutada cada vez que un paquete es recibido.
- Controlador de paquetes basados en caracteres (*Character Packet Driver*): Es una especialización del *packet driver* en el que hay un *overhead* adicional asociado a la transmisión de cada carácter como ocurre en algunas líneas serie. Sus atributos son los del *packet driver* incluyendo además:
    - Servidor de caracteres (*Character Server*): Referencia al servidor de planificación que ejecuta la parte del controlador asociada al envío o recepción de cada carácter (que a su vez contiene una referencia al procesador y a los parámetros de planificación).
    - Operación de envío de carácter *(Character Send Operation*): Operación ejecutada cada vez que un carácter es enviado.
    - Operación de recibo de carácter *(Character Receive Operation*): Operación ejecutada cada vez que un carácter es recibido.
    - Tiempo de transmisión de carácter *(Character Transmission Time*): Tiempo empleado en la transmisión de un carácter.
- Controlador de paquetes Real-Time Ethernet Protocol (*RT-EP Packet Driver*): Es una especialización del *packet driver* en el que hay un *overhead* adicional asociado al paso de testigo. Sus atributos son los del *packet driver* incluyendo además:
    - Particionado de paquete *(Message Partitioning*): Indica si el *driver* es capaz de particionar mensajes de gran longitud y reconstruirlos en el destino. El valor por defecto es *Yes*.
    - Modelo de sobrecarga para el análisis del tiempo de respuesta (*RTA Overhead Model)*: Es un atributo exclusivo de la herramienta de análisis del tiempo de respuesta en el peor caso. El valor determina el modelo de overherad que será utilizado para el driver. Los posibles valores son:
        - *Coupled*: En este modelo de *overhead* una operación de envío y recepción de mensaje se adjuntan a la transacción que causa la transmisión.
        - *Decoupled*: En este modelo, la operación de envío y recepción se ejecutan periódicamente con un periodo igual al tiempo de transmisión mínimo (o a algún otro tiempo dependiente del driver).
    - Número de estaciones *(Number of Stations*): Corresponde al número de estaciones o procesadores conectados a través de la red.
    - Demora de testigo *(Token Delay*): Es un retardo que se introduce en el procesamiento del testigo para así poder reducir el *overhead* de CPU.
    - *Failure Timeout*: Es el tiempo de timeout configurado en el protocolo para el mecanismo de detección de errores.

- Reintentos de transmisión del testigo *(Token Transmission Retries)*: Número máximo de fallos, y sus retransmisiones, que se permiten en cada vuelta de testigo.
- Reintentos de transmisión de paquete (*Packet Transmission Retries)*: Corresponde al número de fallos que se permiten al transmitir un paquete de información.
- Servidor de interrupción de paquete *(Packet Interrupt Server*): Es el servidor que está ejecutando la rutina de interrupción que almacena cada trama recibida en la red independientemente de si se trata de un paquete de información o de arbitrio.
- Operación de rutina de servicio a la interrupción de paquete *(Packet ISR Operation*): Corresponde con la operación ejecutada por el servidor de interrupción cada vez que se recibe una trama, independientemente de la naturaleza del paquete.
- Operación de chequeo de testigo *(Token Check Operation*): Es la operación que se ejecuta para recibir y chequear un paquete de testigo.
- Operación de manejo de testigo *(Token Manage Operation*): Es la operación que se ejecuta para mandar el testigo.
- Operación de descarte de paquete *(Packet Discard Operation*): Es la operación que se ejecuta cuando se reciben, debido al modo promiscuo, y descartan tramas dirigidas a otro destino.
- Operación de retransmisión de testigo (*Token Retransmission Operation)*: Representa la operación que se ejecuta cada vez que se retransmite un testigo.
- Operación de retransmisión de paquete (*Packet Retransmission Operation)*: Corresponde con la operación que se ejecuta cada vez que se retransmite un paquete de información.

2.5. Planificadores *(Schedulers)*

Representan los objetos del sistema operativo que implementan las estrategias de planificación apropiadas para gestionar la capacidad de procesado que les ha sido asignada.

Los planificadores pueden tener una estructura jerárquica (figura 2.1) y se dividen en dos tipos: **primarios** y **secundarios**. Un **planificador primario** ofrece a los servidores de planificación asociados toda la capacidad del procesor al que está asignado. Un **planificador secundario** reparte a los servidores solo la capacidad que a él se le ha asignado por su servidor asociado. Sus atributos comunes son los siguientes:

- Nombre *(Name)*: Utilizado como identificador del planificador
- Política *(Policy)*: Define la política de planificación implementada por el planificador para repartir la capacidad que tiene asignada entre los servidores de planificación que hay asociados a él.

Hay definidas dos clases de planificadores, en función de la fuente de capacidad de procesado:

- Planificador primario (*Primary Scheduler*): Representa el planificador de sistema de base para su recurso de procesado asociado. Tiene el siguiente atributo adicional:
  - Anfitrión *(Host)*: Referencia al recurso de procesado asociado.
- Planificador secundario (*Secondary Scheduler*): Representa un planificador que es capaz de gestionar solo una cierta fracción de capacidad de procesado, que a él se le ha asignado por un servidor de planificación asociado a su vez con otro planificador. Tiene el siguiente atributo adicional:
  - Servidor *(Server)*: Referencia al servidor de planificación asociado.

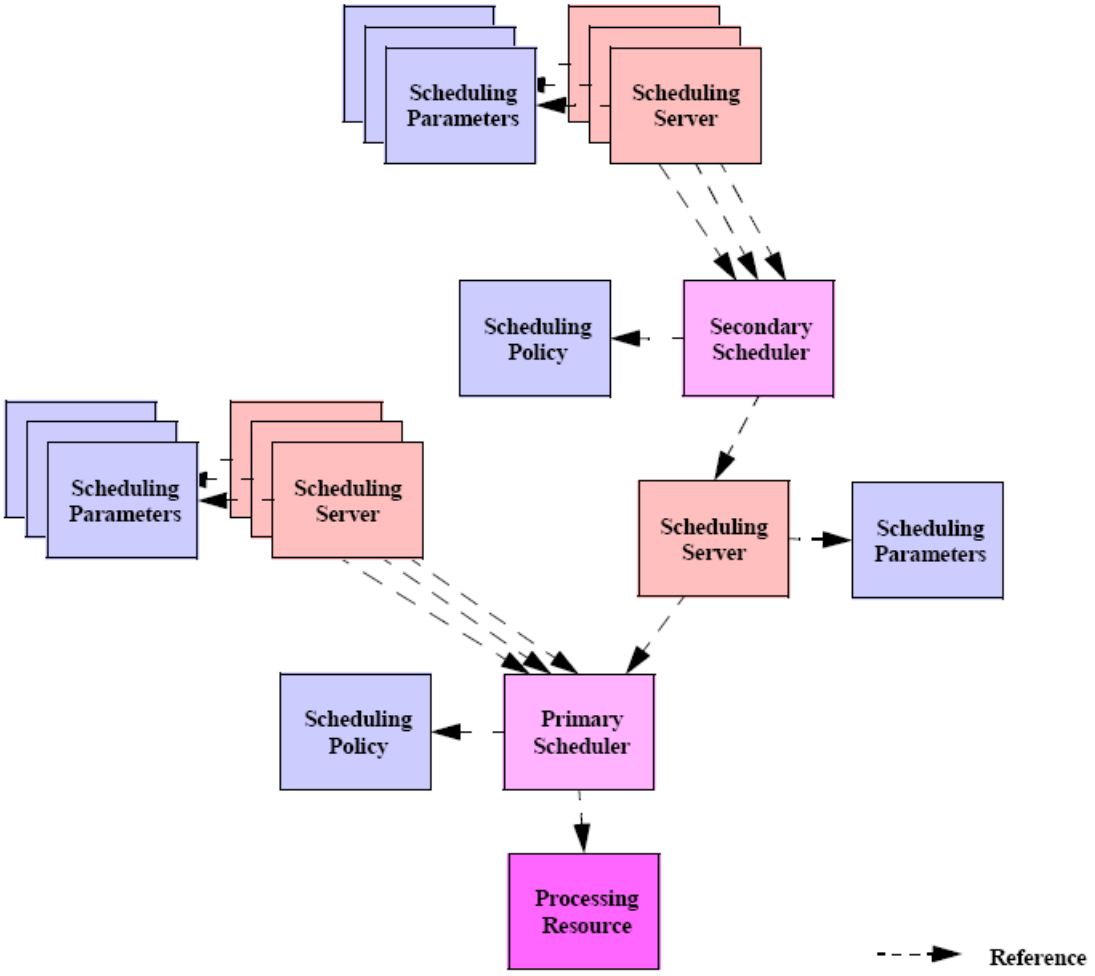

Figura 2.1: Estructura jerárquica de planificadores

2.6. Políticas de planificación *(Scheduling Policies)*

Una política de planificación representa la estrategia básica que es implementada en un planificador para repartir la capacidad de procesado que tiene asignada a los servidores de planificación asociados. Cada uno de estos servidores tiene un objeto de Parámetros de Planificación que describe los parámetros usados por el planificador.

Hay definidas varias clases de Políticas de Planificación:

- Prioridad fija (*Fixed Priority*): Representa una política de prioridades fijas. Posee los siguientes atributos:
  - Overheads (peor, medio y mejor) asociados al cambio de contexto (*Context Switch (Worst, Average, Best)*): Indican los overheads asocidados al cambio de contexto.
  - Prioridades máxima y mínima *(Max Priority* y *Min Priority)*: Indican el rango de prioridades válidos para las operaciones sobre los servidores de planificación planificados con esta política.
- EDF (*Earliest Deadline First*): Sus atributos son:
  - Overheads (peor, medio y mejor) asociados al cambio de contexto *(Context Switch (Worst, Average, Best*)): Indican los overheads asocidados al cambio de contexto.
- Prioridad fija basada en paquetes (*Fixed Priority Packet Based*): Representa una política de prioridades fijas utilizada en una red de comunicaciones por paquetes. Se supone que los paquetes de la red no son expulsables.
  - Overheads (peor, medio y mejor) asociados al paquete *(Packet Overhead (Worst, Average, Best)*): Este es el (peor, medio y mejor) *overhead* asociado a cada paquete producido por los mensajes de protocolo o cabeceras que deben ser mandados antes o después de cada paquete útil.
  - Prioridades máxima y mínima *(Max Priority* y *Min Priority)*: Indican el rango de prioridades válidos para los mensajes mandados con esta política.

2.7. Parámetros de planificación *(Scheduling Parameters)*

Estos parámetros se adjuntan a un servidor de planificación y son usados por su planificador a la hora de tomar decisiones cuando el servidor está compitiendo con otros servidores. Algunas políticas de planificación dejan coexistir en el sistema varios comportamientos de planificación. Por ejemplo, la política de planificación de prioridades fijas permite tanto comportamientos expulsores como no expulsores. Estos comportamientos se determinan por el tipo de parámetros de planificación.

Existen dos clases de parámetros:

- De prioridades fijas: Solo son aplicables a planificadores con política de prioridades fijas. Sus atributos comunes son los siguientes:
  - Prioridad *(Priority):* Número natural entre 1 y 32767 que representa la prioridad de planificación.
  - Preasignada *(Preassigned Priority)*: Si a este parámetro se le asigna el valor *"NO",* la prioridad se puede asignar mediante las herramientas de asignación de prioridades. Si no es así, la prioridad es fija y no puede modificarse con dichas herramientas. Su valor por defecto es *"NO"*.

- Tipo de política *(Policy Type)*: Indica el tipo de política de planificación. Existen varias clases, algunas con atributos adicionales:
  - Basada en prioridades fijas no expulsora (*Non Preemptible Fixed Priority)*: No tiene atributos adicionales.
  - Basada en prioridades fijas *(Fixed Priority)*: Política de prioridades fijas expulsora. No tiene atributos adicionales.
  - Basada en prioridades fijas de interrupción *(Interrupt Fixed Priority)*: Representa una rutina de servicio a la interrupción. No tiene atributos adicionales. El atributo *Preassigned* no puede tener el valor *"NO"*, ya que las prioridades de las interrupciones siempre están preasignadas.
  - De muestreo periódico *(Polling)*: Representa una tarea periódica que pregunta por la llegada de su evento de entrada. La ejecución del evento puede ser pospuesta hasta el siguiente período. Sus atributos adicionales son:
    - Overheads de muestreo periódico (peor, medio y mejor) (*Polling Overheads (Worst, Average, Best)*): *Overhead* asociado a la tarea de muestreo.
    - Período de muestreo (*Polling Period*): Período de la tarea de muestreo.
  - De servidor esporádico *(Sporadic)*: Representa una tarea planificada según el algoritmo de planificación de servidor esporádico. Sus atributos adicionales son:
    - Prioridad de fondo (*Background Priority*): Representa la prioridad a la que se ejecuta la tarea cuando no hay capacidad de ejecución disponible.
    - Capacidad inicial (*Initial Capacity*): Valor inicial de la capacidad de ejecución.
    - Período de relleno (*Replenishment Period*): Período después del cual una porción de la capacidad de ejecución consumida es repuesta.
    - Relleno pendiente máximo (*Maximum Pending Replenishments)*: Número máximo de operaciones de reposición pendientes simultáneamente.
- EDF *(Earliest Deadline First)*: Solo son aplicables a planificadores con política EDF. Sus atributos son los siguientes:
  - Plazo *(Deadline):* Plazo relativo del servidor de planificación asociado.
  - Preasignado *(Preassigned Deadline)*: Si a este parámetro se le asigna el valor *"NO",* el plazo se puede asignar mediante las herramientas de asignación de plazos. Si no es así, el plazo es fijo y no puede modificarse con dichas herramientas. Su valor por defecto es *"NO"*.

Los parámetros de planificación pueden ser sobreescritos en la definición de las operaciones, ver apartado 2.11, "Operaciones (Operations)". Para ello se ha definido una clase especial:

- Parámetros de planificación sobreescritos *(Overridden Scheduling Parameters*): Representa un nivel de prioridad superior al nivel de prioridad normal al cual la operación asociada se ejecutaría. Esta clase es abstracta y las únicas clases específicas definidas como extensión de la misma son:
  - Parámetros de planificación de prioridad fija sobreescrita *(Overridden Fixed Priority Scheduling Parameters*): El cambio de prioridad está activo solo hasta que la operación se completa.
  - Parámetros de planificación de prioridad fija sobreescrita permanente *(Overridden Permanent Fixed Priority Scheduling Parameters*): El cambio de prioridad es efectivo hasta que se define otra prioridad permanente o hasta el final del segmento de actividades.

2.8. Parámetros de sincronización (*Synchronization Parameters*)

Parámetros usados por el servidor cuando está efectuando un acceso a un recurso compartido en modo mutuamente exclusivo. Deben ser especificados siempre que los parámetros de la política de planificación no aporten información suficiente para los protocolos de sincronización usados. Hay un solo tipo:

- *Stack Resource Protocol*: Sus atributos son los siguientes:
  - Nivel de expulsión *(Preemption Level)*: Nivel de explusión usado para el recurso. Puede ser calculado por las herramientas MAST si se quiere.
  - Preasignado *(Preassigned)*: Si a este parámetro se le asigna el valor *"NO",* el nivel de expulsión se puede asignar mediante una herramienta MAST. Si no es así, el nivel es fijo y no puede modificarse con dicha herramienta. Su valor por defecto es *"NO"*.

2.9. Servidores de planificación (*Scheduling Servers*)

Representan entidades planificables en un recurso de procesado. Se utiliza para modelar procesos, threads o tareas en las que se planifican las actividades del sistema que se modela. Si el recurso de procesado es un procesador, el servidor de planificación es un proceso, una tarea o un thread de control. Solo existe una clase definida llamada regular *(Regular)*. Sus atributos son:

- Nombre *(Name)* : Identificador del servidor.
- Parámetros de Planificación (*Scheduling Parameters):* Referencia a los parámetros de planificación asociados al servidor.
- Planificador *(Scheduler)*: Planificador asociado el servidor.

- Parámetros de sincronización (*Synchronization Parameters)*: Referencia a los parámetros de sincronización asociados al servidor.

2.10. Recursos compartidos (*Shared Resources*)

Representan los recursos que se comparten entre distintas tareas y que deben ser usados de un modo mutuamente exclusivo. Sólo se permiten los protocolos que evitan las inversiones de prioridad no acotadas. Existen tres clases, dependiendo del protocolo:

- Recurso de techo de prioridad inmediato *(Immediate Ceiling Resource):* Utiliza el protocolo de techo de prioridad. Sus atributos son:
  - Nombre *(Name)* : Identificador del recurso.
  - Techo *(Ceiling):* Techo de prioridad usado para el recurso. Puede ser calculado automáticamente por la herramienta.
  - Preasignado *(Preassigned)*: Si a este parámetro se le asigna el valor *"NO",* el techo de prioridad se puede asignar mediante la herramienta de cálculo de techos de prioridad. Si no es así, el techo de prioridad es fijo y no puede modificarse con dicha herramienta. Su valor por defecto es *"NO"*.
- Recurso de herencia de prioridad *(Priority Inheritance Resource):* Utiliza el protocolo de herencia de prioridad. Su único atributo es el nombre *(Name).*
- *Stack Resource Protocol*: Utiliza un protocolo similar al protocolo de techo de prioridad pero que funciona para políticas no basadas en prioridades. Sus atributos son:
  - Nombre *(Name)*: Identificador del recurso.
  - Nivel de expulsión *(Preemption Level)*: Nivel de explusión usado para el recurso. Puede ser calculado por las herramientas MAST si se quiere.
  - Preasignado *(Preassigned)*: Si a este parámetro se le asigna el valor *"NO",* el nivel de expulsión se puede asignar mediante una herramienta MAST. Si no es así, el nivel es fijo y no puede modificarse con dicha herramienta. Su valor por defecto es *"NO"*

2.11. Operaciones (*Operations*)

Representan una parte de código a ejecutar por un procesador o un mensaje que es enviado a través de una red. Tienen los siguientes atributos comunes:

- Nombre *(Name)*: Identificador de la operación.
- Parámetros de planificación sobreescritos *(Overridden Scheduling Parameters*): Representa un nivel de prioridad superior al nivel de prioridad normal al cual la operación se ejecutaría. Se contemplan dos casos en función de las características del cambio de prioridad:
  - Para una prioridad fija sobreescrita *(Overridden Fixed Priority*), el cambio de prioridad está activo solo hasta que la operación se completa.

- Para una prioridad fija sobreescrita permanente *(Overridden Permanent Fixed Priority*), el cambio de prioridad es efectivo hasta que se define otra prioridad sobreescrita permanente o hasta el final del segmento de actividades.

Se definen las siguientes clases de operaciones:

- Simple (*Simple*): Representa un trozo de código sencillo o un mensaje. Presenta los siguientes atributos adicionales:
  - Tiempos de ejecución (peor, medio y mejor) *(Execution Time (Worst, Average Best))*: Se expresa en unidades de tiempo normalizadas. Para los mensajes, representa el tiempo de transmisión.
  - Recursos compartidos a bloquear (*Shared Resources To Lock*): Lista de referencias a los recursos compartidos que se han de bloquear antes de ejecutar la operación.
  - Recursos compartidos a liberar (*Shared Resources To Unlock*): Lista de referencias a los recursos compartidos que se han de liberar después de ejecutar la operación.
- Compuesta (*Composite*): Representa una operación compuesta de una secuencia ordenada de otras operaciones, simples o compuestas. El tiempo de ejecución no se especifica ya que es la suma de los tiempos de ejecución de las operaciones que la componen. Sus atributos adicionales son:
  - Lista de operaciones (*Operation List*): Lista de referencias a las otras operaciones.
- Englobante (*Enclosing*): Como la operación compuesta, representa una operación que contiene otras operaciones como parte de su ejecución, pero en este caso el tiempo de ejecución ha de ser especificado, ya que no es la suma de los tiempos de ejecución de las operaciones que la componen, debido a que otros trozos de código pueden ejecutarse adicionalmente. Las operaciones englobadas necesitan ser consideradas con el fin de calcular los tiempos de bloqueo asociados con el uso de sus recursos compartidos. Sus atributos adicionales son los mismos que los de la operación compuesta incluyendo además:
  - Tiempos de ejecución (peor, medio y mejor) *(Execution Time (Worst, Average Best))*: Se expresa en unidades de tiempo normalizadas. Para los mensajes, representa el tiempo de transmisión.
- Transmisión de Mensaje *(Message Transmission*): Representa un mensaje a transmitir a través de una red. Sus atributos adicionales son:
  - Tamaños de paquete (máximo, medio y mínimo) *(Message Size (Max, Avg, Min))*: El tiempo de transmisión se calcula dividiendo el tamaño del paquete por el *Throughput* y el *Speed Factor* de la red correspondiente.

2.12. Eventos (*Events*)

Representan canales de secuencias de eventos a través de los cuales se generan las instancias de eventos particulares. Una instancia de evento activa una instancia de una actividad, o influye en el comportamiento del manejador de eventos al que es dirigido. Existen dos clases:

- Eventos internos *(Internal Events)*: Generados por un manejador de eventos. Sus atributos son:
  - Nombre *(Name):* Identificador del evento.
  - Requerimientos temporales *(Timing Requirements)*: Referencia a los requerimientos temporales impuestos en la generación del evento, ver apartado 2.13, "Requerimientos temporales (Timing Requirements)".
- Eventos externos *(External Events):* Modelan las interacciones del sistema con componentes o dispositivos externos a través de interrupciones, señales, etc., o con dispositivos de temporización hardware. Tienen un doble papel en el modelo: por una parte, establecen las tasas o patrones de llegada de las actividades al sistema, por otra parte, proporcionan referencias para definir requerimientos temporales globales. Tenemos las siguientes clases definidas para representar los diferentes patrones de llegada:
  - Periódico (*Periodic*): Representa una cadena de eventos generados periódicamente. Sus atributos son el nombre *(Name)*, el periodo *(Period),* la máxima fluctuación *(Max Jitter)* o máxima cantidad de tiempo a añadir al tiempo de activación de cada instancia de evento y la fase *(Phase)* que es el instante de la primera activación en caso de que no tenga fluctuación (después de ese tiempo los siguientes eventos son periódicos, posiblemente con fluctuación).
  - Unitario (*Singular*): Representa un evento generado solo una vez. Sus atributos son el nombre y la fase (instante de la primera activación).
  - Esporádico (*Sporadic*): Representa una cadena de eventos aperiódicos que tienen un tiempo entre llegadas mínimo. Sus atributos son el nombre, el tiempo entre llegadas mínimo *(Min Interarrival)*, el tiempo entre llegadas promedio *(Average Interarrival)* y la función de distribución de los eventos aperiódicos *(Distribution)* que puede ser uniforme *(Uniform)* o poisoniana *(Poisson)*.
  - No acotado (*Unbounded*): Representa una cadena de eventos para la cual no es posible establecer una cota superior del número de eventos que pueden llegar en un intervalo dado. Sus atributos son: el nombre, el tiempo entre llegadas promedio *(Average Interarrival)* y la función de distribución *(Distribution)*.
  - A ráfagas (*Bursty*): Representa una cadena de eventos aperiódicos que tienen una cota superior del número de eventos que pueden llegar en un intervalo dado. Dentro de este intervalo, los eventos pueden llegar a ráfagas. Tienen los siguientes atributos: el nombre,

el intervalo de acotación *(Bound Interval)* en el cual la cantidad de eventos está acotada, el máximo número de eventos que pueden llegar en dicho intervalo *(Max Arrivals)*, el tiempo entre llegadas promedio (*Average Interarrival)* y la función de distribución (*Distribution*).

2.13. Requerimientos temporales (*Timing Requirements*)

Representan los requerimientos temporales impuestos en el instante de la generación del evento interno asociado. Existen de distintos tipos:

- Plazo *(Deadline):* Representan un valor de tiempo máximo permitido para la generación del evento asociado. Los plazos pueden ser estrictos (han de cumplirse siempre) o no estrictos (se han de cumplir en promedio) y, en función a qué instante de tiempo se referencien, pueden ser locales (el plazo es relativo a la llegada del evento que activó dicha actividad) o globales (el plazo es relativo a la llegada de un evento referenciado que que es un atributo del plazo). Atendiendo a esto se definen cuatro clases:
  - Plazo global estricto *(Hard Global Deadline)*: Sus atributos son el valor del plazo *(Deadline)* y una referencia al evento referenciado *(Referenced Event*).
  - Plazo global no estricto *(Soft Global Deadline):* Sus atributos son el valor del plazo *(Deadline)* y una referencia al evento referenciado *(Referenced Event*).
  - Plazo local estricto *(Hard Local Deadline):* Su único atributo es el valor del plazo *(Deadline)*.
  - Plazo local no estricto (*Soft Local Deadline):* Su único atributo es el valor del plazo *(Deadline)*.
- Requerimiento de máxima fluctuación de salida *(Max Output Jitter Requirement):* Representa un requerimiento para limitar la fluctuación con la cual un evento interno es generado. La fluctuación se calcula como la diferencia entre la respuesta temporal del peor y del mejor caso de la actividad que genera el evento asociado. Sus atributos son la fluctuación máxima de salida *(Max Output Jitter)* y una referencia al evento *(Referenced Event*).
- Tasa de pérdida máxima *(Max Miss Ratio):* Representa un tipo de plazo no estricto en el cual el plazo no puede ser perdido más alla de una determinada tasa. Sus atributos son el plazo *(Deadline)* y la tasa o porcentaje *(Ratio)* de pérdida de plazos máxima. Hay dos tipos de requerimientos: globales y locales.
- Compuestos *(Composite):* Un evento puede tener varios requerimientos impuestos al mismo tiempo, que son expresados a través de un requerimiento temporal compuesto. Contiene una lista de requerimientos temporales simples.

2.14. Manejadores de eventos (*Event Handlers*)

Los manejadores de eventos representan acciones activadas por la llegada de uno o más eventos, y que a su vez pueden generar uno o más eventos a su salida. Hay dos clases de manejadores básicas: las actividades *(Activities)* y los manejadores estructurales. Las actividades representan la ejecución de una operación por un servidor de planificación, en un recurso de procesado y con unos parámetros de planificación dados. Los otros manejadores son sólo un mecanismo para manejar eventos sin efectos en el tiempo de ejecución. Cualquier *overhead* asociado con su implementación es asignado a las actividades asociadas.

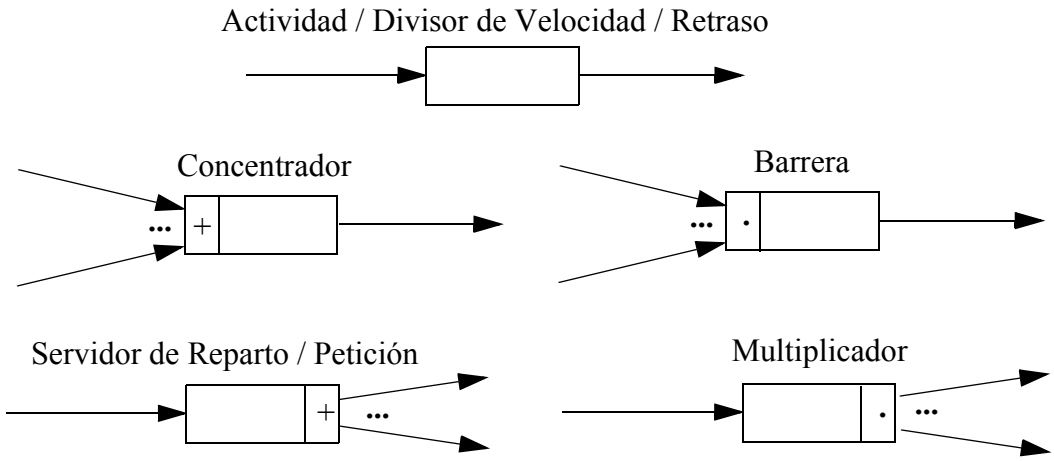

Figura 2.2: Clases de Manejadores de Eventos [7]

Los distintos tipos de manejadores son (ver figura 2.2):

- Actividad *(Activity):* Representa una instancia de una operación para ser ejecutada por un servidor de planificación. Sus atributos son:
  - Evento de entrada *(Input event*): Referencia al evento de entrada.
  - Evento de salida *(Output event)*: Referencia al evento de salida.
  - Operación de la actividad *(Activity Operation)*: Referencia a la operación.
  - Servidor de la actividad *(Activity Server)*: Referencia al servidor de planificación.
- Actividad temporizada de sistema *(System Timed Activity)*: Representa una actividad activada por el temporizador del sistema y por tanto sujeta a los *overheads* asociados con él. Tiene los mismos atributos que la anterior
- Concentrador *(Concentrator)* : Es un manejador que genera su evento de salida cuando llega cualquiera de sus eventos de entrada. Sus atributos son:
  - Eventos de entrada *(Input events*): Referencias a los eventos de entrada.
  - Evento de salida *(Output event)*: Referencia al evento de salida.
- Barrera *(Barrier):* Es un manejador que genera su evento de salida cuando todos sus eventos de entrada han llegado. Para poder realizar el análisis

del peor caso es necesario que todos los eventos de llegada sean periódicos con el mismo período. Sus atributos son los mismos que los del concentrador.

- Servidor de reparto *(Delivery Server):* Es un manejador que genera un evento en solo una de sus salidas cada vez que llega un evento de entrada. La salida se determina en el momento de generación del evento. Sus atributos son:
    - Evento de entrada *(Input event*): Referencia al evento de entrada.
    - Eventos de salida *(Output events)*: Referencia al evento de salida.
    - Política de reparto *(Delivery Policy)*: Política utilizada para determinar el camino de salida. Puede ser de escaneo *(Scan)* (la salida se elije de forma cíclica) o aleatoria *(Random)*.
- Servidor de petición *(Query Server):* Este manejador genera un evento en solo una de sus salidas cada vez que llega un evento de entrada. La salida se elige en el momento en que el evento se despacha por una de las actividades conectadas a un evento de salida. Sus atributos son:
    - Evento de entrada *(Input event*): Referencia al evento de entrada.
    - Eventos de salida *(Output events)*: Referencia al evento de salida.
    - Política de petición *(Request Policy)*: Política que determina el camino de salida cuando hay varias peticiones pendientes desde las actividades conectadas. Puede ser de escaneo *(Scan)* (la salida se elije de forma cíclica), de prioridad *(Priority)* (la actividad de prioridad más alta gana), FIFO o LIFO.
- Multiplicador *(Multicast):* Manejador que genera un evento en cada una de sus salidas cada vez que llega un evento. Sus atributos son el evento de llegada y los eventos de salida.
- Divisor de frecuencia *(Rate Divisor)*: Es un manejador que genera un evento de salida cuando el número de eventos de llegada es igual al factor de división *(Rate Factor)*. Sus atributos son:
    - Evento de entrada *(Input event*): Referencia al evento de entrada.
    - Evento de salida *(Output event)*: Referencia al evento de salida.
    - Factor de velocidad *(Rate Factor)*: Número de eventos que han de llegar para que se genere un evento de salida.
- Retraso Local (*Delay)*: Es un manejador que genera su evento de salida después de que haya transcurrido un cierto intervalo de tiempo desde la llegada del evento de entrada. Sus atributos son:
    - Evento de entrada *(Input event*): Referencia al evento de entrada.
    - Evento de salida *(Output event)*: Referencia al evento de salida.
    - Intervalo máximo de retraso *(Delay Max Interval)*: Intervalo de tiempo máximo usado para generar el evento de salida.
    - Intervalo mínimo de retraso *(Delay Min Interval)*: Intervalo de tiempo mínimo usado para generar el evento de salida.
- Retraso Global *(Offset):* Es similar al anterior, excepto que el intervalo de tiempo es relativo a la llegada de algún evento previo. Si el intervalo de

tiempo ya ha pasado cuando el evento de entrada llega, el evento de salida es generado inmediatamente. Sus atributos son los mismos del anterior más la referencia al evento determinado *(Referenced Event)*.

2.15. Transacciones (*Transactions*)

Una transacción es un grafo de manejadores y eventos que representan las actividades interrelacionadas que se ejecutan en el sistema. Sólo hay una clase de transacción definida llamada regular *(Regular)*. Una transacción tiene los siguientes atributos:

- Nombre *(Name):* Identificador de la transacción.
- Eventos externos *(External Events)*: Lista de eventos externos.
- Eventos internos *(Internal Events)*: Lista de eventos internos con sus requerimientos temporales (si es que los tienen).
- Manejadores de Eventos *(Event Handlers)*: Lista de manejadores.

2.16. Modelo General (*Overall Model*)

El modelo general MAST representa una posible situación de un sistema de tiempo real que necesita ser analizada, esto es, un modo de operación del sistema junto con un modelo de la carga de trabajo que tiene que ejecutar. La información global acerca de la situación se describe en el objeto *Model* que contine los siguientes atributos:

- Nombre del modelo *(Model Name):* Identificador del modelo.
- Fecha del modelo *(Model Date)*: Fecha en la que se creó el modelo de la situación de tiempo real.

Este objeto junto con todos los mencionados anteriormente, nos va a permitir definir y analizar todas aquellas situaciones de tiempo real que puedan darse en un sistema concreto y comprobar si, efectivamente, los requerimientos temporales exigidos a dicho sistema se cumplen atendiendo a los criterios de planificación asignados.

3. Descripción Funcional del Editor: Manual de Usuario

3.1. Introducción

El conjunto de herramientas MAST surge con la finalidad de realizar el modelado y análisis de los sistemas de tiempo real, proporcionando facilidades para el análisis, simulación, modelado y representación gráfica que permiten al usuario realizar la caracterización de este tipo de sistemas.

La estructura principal de las herramientas sigue una secuencia lógica para el modelado y análisis de los sistemas de tiempo real. Dicha secuencia se puede resumir en los siguientes pasos: primero se debe elaborar el modelo MAST del sistema y configurar cada uno de los componentes que lo forman. Seguidamente se ha de decidir qué tipo de análisis se piensa realizar, determinando los algoritmos y técnicas de análisis a emplear. Para ello se dispone de interfaces gráficas que permiten la configuración del modelo y de las herramientas de análisis disponibles. A continuación se debe realizar la generación y almacenamiento de resultados y como último paso se llevaría a cabo la representación y análisis de los resultados obtenidos, determinando la planificabilidad del sistema. Estos pasos se pueden realizar de forma iterativa hasta verificar que se cumplen los requerimientos temporales del sistema y que la respuesta temporal del sistema es la adecuada.

En este capítulo vamos a describir de forma detallada cómo se realiza la elaboración y configuración del modelo MAST a través de una de las herramientas incluidas en el software MAST: el editor gráfico.

El editor se podría haber desarrollado diseñando una pantalla o ventana independiente para cada uno de los elementos del modelo MAST. Pero teniendo en cuenta cómo está estructurado el modelo y las diferentes opciones disponibles para las clases definidas en la especificación MAST esto daría lugar a un elevado número de ventanas. Por eso se optó por utilizar una estructura de capas que redujera dicho número de pantallas y que proporcionara un aspecto más contenido y consistente al software. El acceso a cada una de estas capas se realizará de forma directa mediante un conjunto de "pestañas" pertenecientes a la misma ventana y que se corresponden con los diferentes diagramas diseñados para facilitar la creación y configuración del modelo MAST de una forma visual y sencilla.

Previamente a la elaboración de dichos diagramas se clasificaron los elementos del modelo MAST jerárquicamente, dividiéndolos en dos tipos (ver figura 3.1):

- **Elementos de Primer Nivel:** Son los elementos que se especifican de manera independiente del resto. A este tipo pertenecen los Recursos de

Procesado, Planificadores, Servidores de Planificación, Recursos Compartidos, Operaciones y Transacciones.

- **Elementos de Segundo Nivel:** Son los elementos que están directamente contenidos en otros elementos del modelo. A esta clase pertenecen los Temporizadores del Sistema, Controladores de Red, Políticas de Planificación, Parámetros de Planificación, Eventos, Requerimientos Temporales y Manejadores de Eventos.

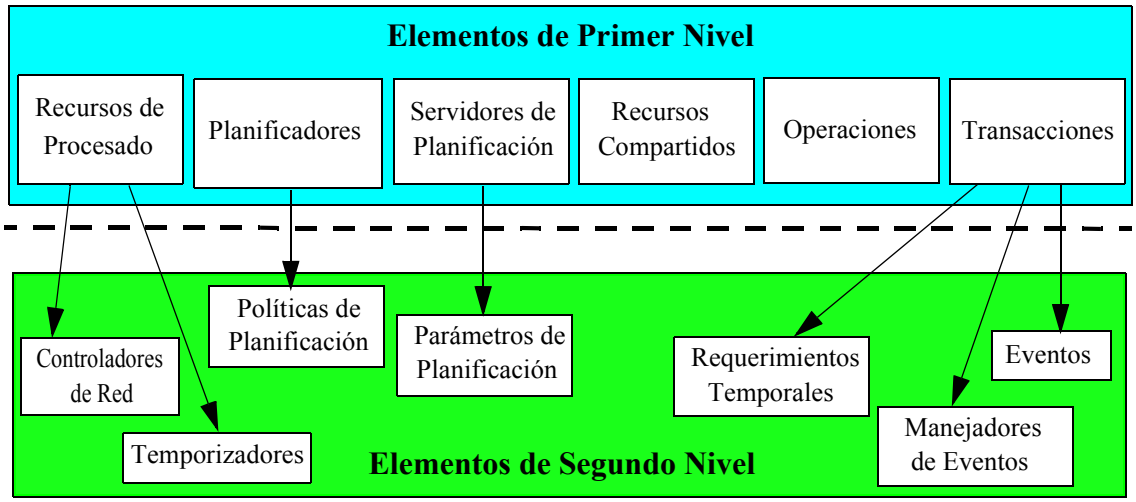

Figura 3.1: Jerarquía de Elementos del modelo MAST

De este modo, cada uno de los diagramas del editor se corresponde básicamente con una representación gráfica de los elementos de primer nivel, incluyendo sus atributos y los elementos de segundo nivel a los que hacen referencia, si bien en algunos diagramas pueden aparecer también elementos de primer nivel de distinta clase asociados entre sí.

Además, existen otras ventanas adicionales que actúan como complemento del módulo principal que van a permitir, entre otras cosas, configurar los atributos de cada clase de elemento o hacer uso de funcionalidades añadidas como puede ser el manejo de ficheros. El desarrollo de estos módulos auxiliares se ha realizado conforme han ido surgiendo necesidades en la implementación, y no se han incluído directamente en la ventana principal para evitar su sobrecarga.

Lo que se pretende en este capítulo es realizar una descripción exhaustiva de todas y cada una de las ventanas y opciones del software desarrollado, para que pueda ser utilizada tanto como guía de diseño de la aplicación como de manual de usuario. La descripción está realizada siguiendo la estructura del bloque principal, e incluyendo las pantallas secundarias que le afectan directamente.

3.2. Arranque de la aplicación

En este apartado se explica el procedimiento a seguir para arrancar la aplicación tanto en entornos Windows como Linux ya que las herramientas MAST dan soporte para ambas plataformas. Una vez realizada la instalación del software, para arrancar la aplicación es suficiente con invocar la herramienta desde la línea de comandos:

```
> gmasteditor [archivo_de_entrada]
```

donde "archivo_de_entrada" es un parámetro opcional que corresponde al nombre del archivo que contiene la descripción MAST del sistema. El nombre del archivo especificado puede tener extensión "txt" (extensión por defecto) o cualquier otra que se desee, si bien también se permite la lectura de archivos sin extensión.

La figura 3.2 muestra el aspecto de la ventana principal que aparece al arrancar la aplicación. En este caso, se ha especificado el archivo a cargar al inicio, de ahí que aparezcan elementos en el diagrama contenido en la primera pestaña de la pantalla, el cual corresponde a la Capa de Recursos de Procesado y Planificadores (*Processing Resources And Schedulers*). Si no se hubiese determinado el nombre del archivo tanto este diagrama como los de las demás capas aparecerían vacíos.

3.3. Configuración del modelo MAST

Para la creación del modelo MAST de un sistema de tiempo real, el editor proporciona un conjunto de ventanas que ayudan en esa labor. El bloque principal contiene una única ventana, la cuál aparece nada más arrancar la aplicación, como se muestra en la figura 3.2. Esta pantalla está estructurada en cinco capas asociadas a las correspondientes clases de elementos de primer nivel (ver figura 3.1). Cada una de estas capas contiene un diagrama con la disposición de dichos elementos. Esta ventana principal lleva a su vez asociadas distintas ventanas auxiliares que nos permitirán configurar los elementos de primer nivel del modelo, sus atributos y las referencias existentes entre éstos y los elementos de segundo nivel.

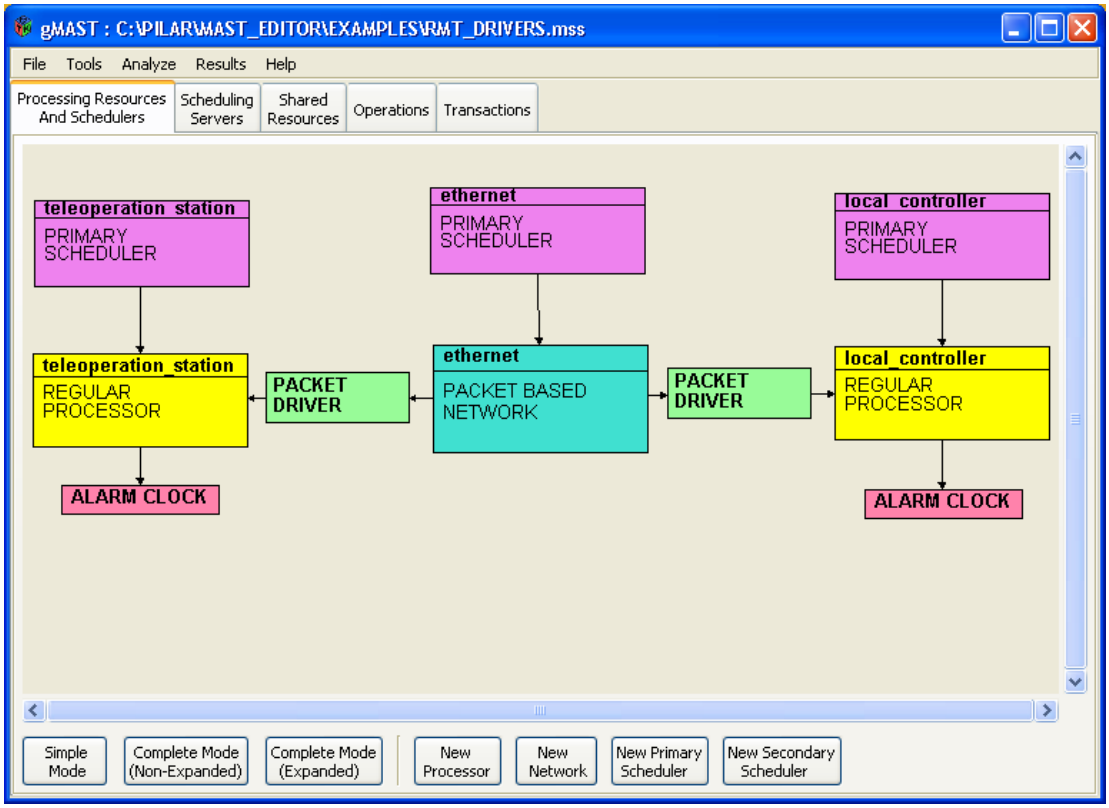

Figura 3.2: Ventana Principal del Editor Gráfico

3.3.1. Ventana Principal

La pantalla principal presenta un aspecto como el de la figura 3.2. Para facilitar la definición de la ventana, se divide su distribución en dos grupos: menú y área de pestañas. A continuación se realiza una pequeña descripción de cada uno de ellos:

- **Menú**: En él están recogidas las opciones para el manejo de archivos, las herramientas, el análisis del sistema, el visor de resultados y la ayuda del editor. Se compone de los siguientes elementos (ver figura 3.2):
  - **File**: Contiene las opciones de crear, abrir, importar y guardar ficheros (ver apartado 3.4, "Operaciones con ficheros"), así como la de salir de la aplicación.
  - **Tools**: Contiene una asistente para la creación de transacciones.(ver apartado 3.5, "Herramientas (Tools)").
  - **Analyze**: Su función es la configuración de las herramientas de análisis por medio de la ventana mostrada en la figura 3.3. En ella se pueden definir los archivos de entrada y salida, la técnica de análisis de planificabilidad del sistema (*Tool*), y determinar las opciones de análisis (*Options*):
    - ***Verbose***: Análisis con mensajes detallados, pensados principalmente para la depuración de las herramientas.
    - ***Calculate Ceilings and Levels***: Cálculo previo al análisis de los techos de prioridad de recursos compartidos de dicho tipo, así como de los niveles de expulsión, si se usa el protocolo de sincronización SRP.
    - ***Assign Parameters***: Asignación de prioridades óptima (previa al análisis) utilizando una determinada técnica (*Technique*) cuyos parámetros se pueden ajustar con los botones *HOPA Parameters* y *Annealing Parameters*.
    - ***Calculate Slacks***: Cálculo de Holguras de cada transacción y del sistema total.
    - ***Calculate Operation Slack***: El análisis es iterado para obtener la holgura de la operación asociada al nombre especificado en la entrada de texto de la parte derecha.
  - **Results**: Su función es la visualización de los resultados del análisis del sistema (figura 3.4). Se compone de cuatro solapas:
    - ***System***: Se especifica el nombre y fecha del modelo, los detalles de generación del análisis (herramienta *(Tool)*, perfil *(Profile)* y fecha *(Date)*) y se muestra si el sistema es planificable o no y el *slack* si se especificó al analizar.
    - ***Processing Resources***: Se muestran los recursos de procesado del sistema y el *slack* si se especificó al analizar.
    - ***Transactions***: Se muestran las transacciones del sistema y el *slack* si se especificó al analizar.
    - ***Shared Resources***: Se muestran los recursos compartidos del sistema, especificando el nombre y el tipo.
  - **Help**: Está asociado a la herramienta de ayuda.

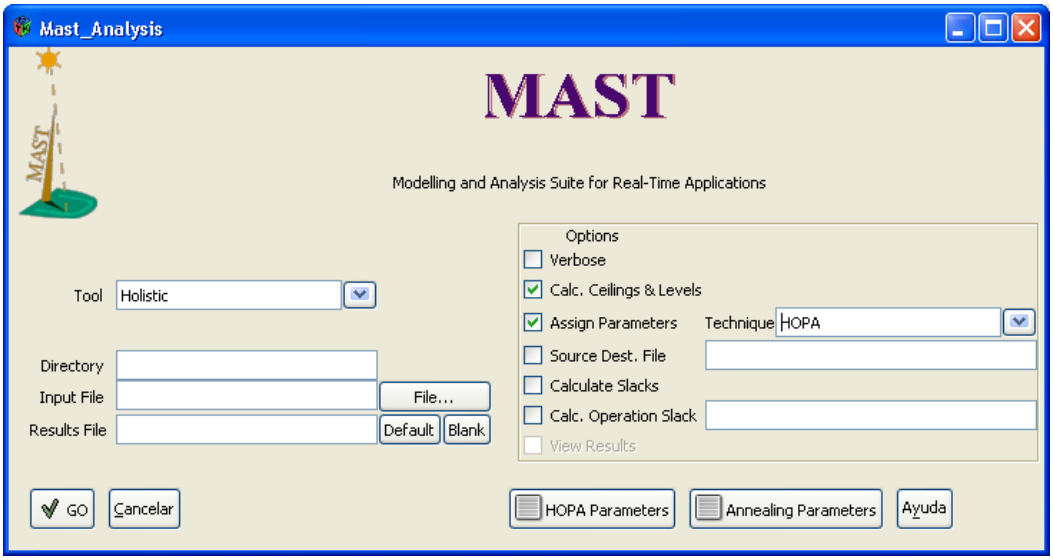

Figura 3.3: Pantalla de configuración de las herramientas de análisis

- **Área de Pestañas**: Esta parte contiene las cinco capas que componen el núcleo de este bloque del programa. El título de cada pestaña indica el tipo de elemento de primer nivel al que corresponde. En los siguientes apartados se hace una descripción detallada de cada una de las capas, si bien todas comparten una estructura común basada en dos elementos:
  - **Barra de Herramientas**: Proporciona varias funcionalidades, como pueden ser la definición del nivel de detalle de los diagramas o la de creación de nuevos elementos del modelo.
  - **Área de Representación Gráfica**: Es la zona reservada para la visualización de los diagramas definidos para el modelo.

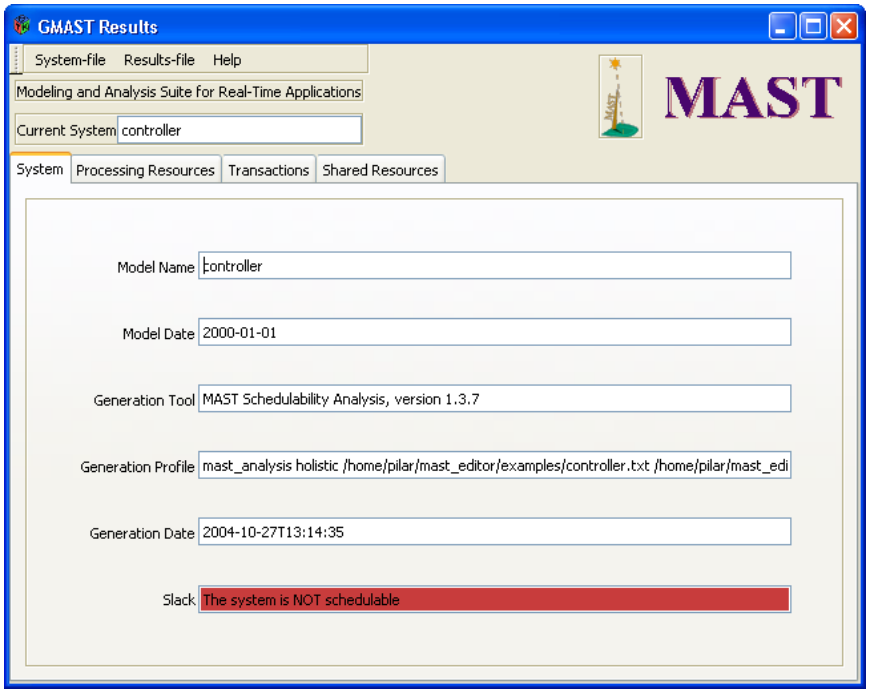

Figura 3.4: Pantalla del visor de resultados

3.3.2. Capa de Recursos de Procesado y Planificadores

En esta capa se visualiza el esquema general de los dispositivos y equipos que componen el sistema (Recursos de Procesado), esto es, los procesadores, las redes y los temporizadores y controladores que tales elementos pueden llevar asociados respectivamente. Además, se incluyen los Planificadores que representan objetos del sistema operativo que implementan políticas de planificación para gestionar la capacidad de procesado asignada a ellos. Es una representación básica que nos aporta una idea del equipamiento que estamos manejando, de las características de los dispositivos y de las politicas de planificación utilizadas, determinando parámetros como prioridades, velocidad, tipo de transmisión, sobrecargas debidas a los temporizadores, sobrecargas asociadas a la transmisión de mensajes, tipo de políticas de planificación, etc.

Su presentación es la que se muestra en las figuras 3.2 y 3.5. La capa se divide en dos partes: un área de dibujo en la que aparece el **Diagrama de la Plataforma** *(Platform Diagram)* con los elementos mencionados anteriormente y una barra de herramientas.

En el Diagrama de la Plataforma, cada **procesador** *(processor)* está representado por una caja de color amarillo en la que se especifican sus propiedades (nombre, tipo, prioridad de interrupción, velocidad, etc). Si un procesador tiene asociado un temporizador *(timer)*, éste también aparecerá a su lado en el diagrama representado por una caja de color magenta.

Las **redes** *(network)* se representan con una caja de color verde en la que se pueden visualizar sus principales propiedades (nombre, tipo, eficiencia, tamaño de paquete, velocidad, tipo de tranmisión, etc). Los controladores de red *(drivers)* también aparecen el dibujo y se asocian a la red y al procesador al que hacen referencia mediante flechas.

Los **planificadores** *(scheduler)* pueden tener una estructura jerárquica y se dividen en dos tipos: **primarios** y **secundarios**. Un **planificador primario** ofrece a los servidores de planificación asociados toda la capacidad del procesor al que está asignado y se representan con una caja de color violeta. Un **planificador secundario** reparte a los servidores solo la capacidad que a él se le ha asignado por su servidor asociado y se representan por cajas de color morado.

La función del diagrama y de los objectos gráficos que lo componen es ayudar a identificar los recursos de procesado del sistema, los planificadores y sus elementos asociados y dar una visión esquemática que haga más fácil la construcción del modelo.

La mayoría de las opciones que proporciona esta capa están implementadas en la **barra de herramientas** que aparece en la parte inferior de la ventana, dividida en dos grupos: por un lado, los botones referentes al nivel de detalle que se quiere mostar en el diagrama y, por otro lado, los botones de creación de nuevos elementos del modelo. En total hay siete botones, cuya función se define a continuación.

- **Modo Sencillo** *(Simple Mode)*: dibuja el diagrama con un nivel de detalle muy simple, mostrando en cada elemento el tipo y, en caso de que lo tenga, también su nombre, como se puede ver en la figura 3.2.
- **Modo Completo No Expandido** *(Complete Mode Non Expanded)*: selecciona un nivel de detalle mayor, mostrando más atributos en los objetos gráficos, como se muestra en la figura 3.6.

- **Modo Completo Expandido** *(Complete Mode Expanded)*: con esta opción se selecciona el mayor nivel de detalle disponible para el diagrama.
- **Nuevo Procesador** *(New Processor)*: muestra una ventana auxiliar con el diálogo que permite configurar las propiedades de un procesador (ver figura 3.7).
- **Nueva Red** *(New Network)*: muestra el diálogo que permite configurar las propiedades de una red (ver figura 3.9).
- **Nuevo Planificador Primario** *(New Primary Scheduler)*: muestra el diálogo que permite configurar las propiedades de un planificador primario (ver figura 3.11)
- **Nuevo Planificador Secundario** *(New Secondary Scheduler)*: muestra el diálogo que permite configurar las propiedades de un planificador secundario (ver figura 3.12)

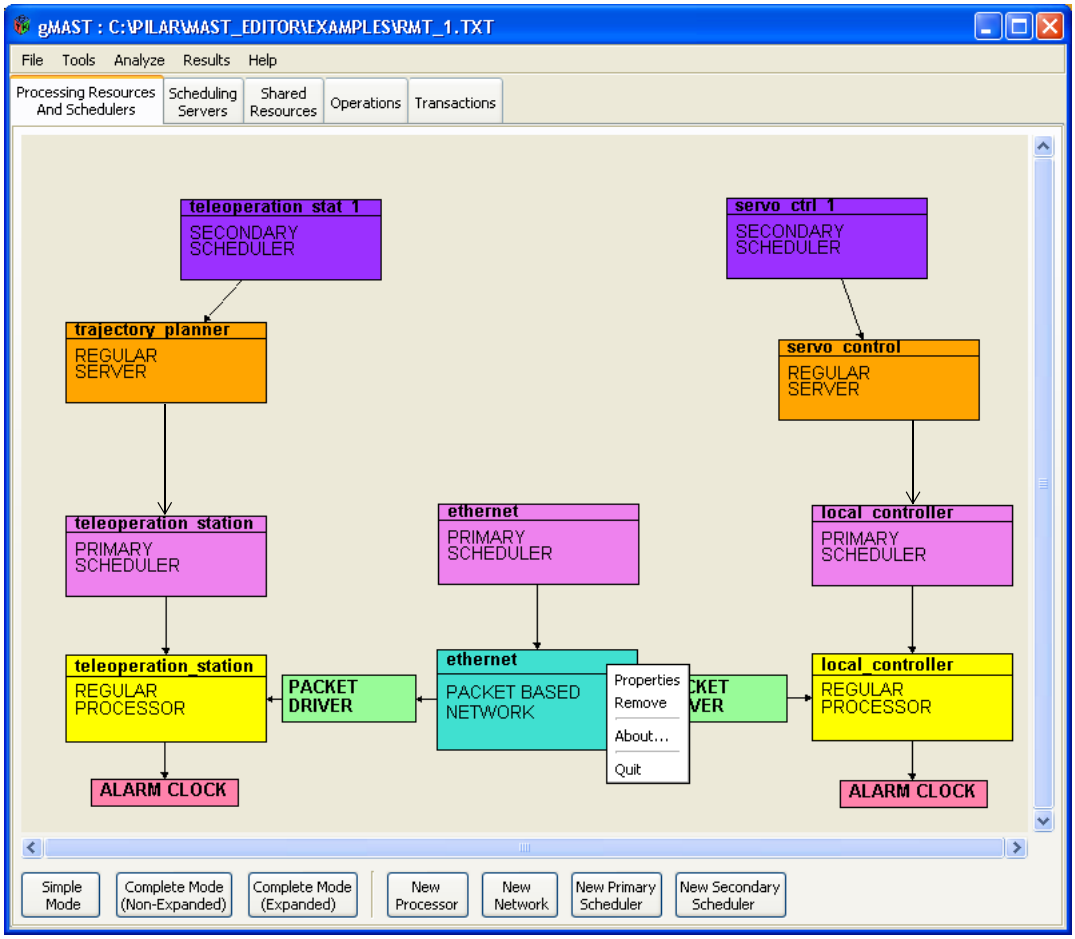

Figura 3.5: Capa de Recursos de Procesado y Planificadores: Simple Mode

Si se observan los diagramas de la figura 3.5 y de la figura 3.6, se pueden apreciar los efectos de seleccionar un nivel de detalle u otro para el mismo sistema, utilizando los botones de la barra de herramientas. En el primer caso se optó por el modo sencillo y en el segundo por el modo completo no expandido.

Es preciso señalar que incluso optando por el nivel de detalle máximo, en el diagrama no se muestran todas las propiedades de los elementos. Esto es debido a que el espacio del que se dispone dentro de los objetos gráficos es bastante limitado. No

obstante, parece razonable que una vez añadido un elemento al sistema, en un momento dado pueda darse la circunstancia de que el usuario necesite cambiar o, en su defecto, consultar todas las propiedades del elemento, o que incluso quiera eliminarlo del sistema.

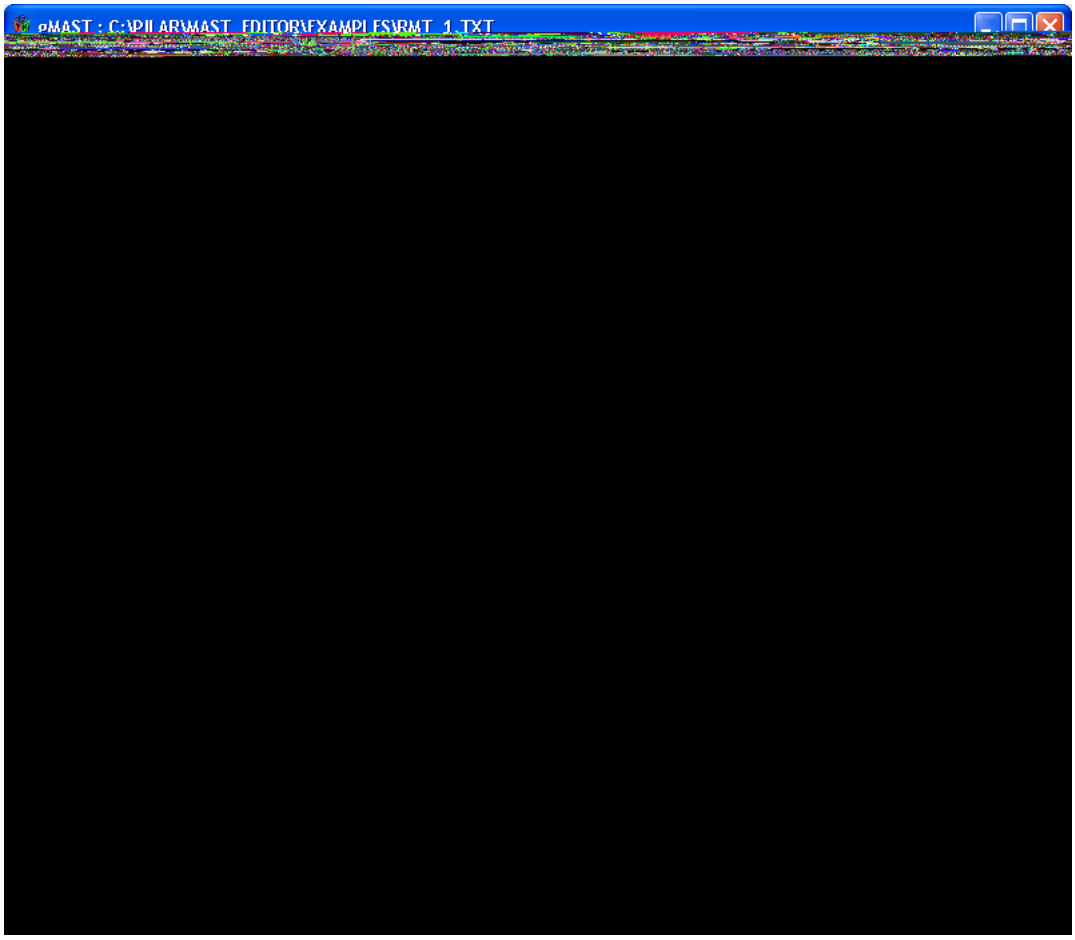

Figura 3.6: Capa de Recursos de Procesado y Planificadores: Complete Mode (Non-Expanded)

Para ello se ha habilitado una funcionalidad añadida a las de la barra de herramientas y que consiste en la elaboración de un **menú emergente** que se activa cuando el usuario se sitúa con el ratón y pulsa el botón derecho encima de un elemento. Entonces aparece una lista de opciones como se muestra en la parte central de la figura 3.5. Dicho menú no es exclusivo de esta capa sino que es común a todas las capas de la interfaz, presentando las siguientes opciones:

- **Borrar** *(Remove)*: elimina el elemento del sistema y del diagrama automáticamente.
- **Propiedades** *(Properties)*: muestra el correspondiente diálogo de propiedades del elemento con todos sus atributos. Es la opción más útil, ya que nos va a permitir consultar o cambiar los parámetros que precisemos.
- **Acerca** *(About)*: muestra la herramienta de ayuda.
- **Salir** *(Quit)*: cierra la aplicación, preguntando al usuario si quiere salvar el modelo editado.

También se pueden consultar las propiedades de un elemento haciendo doble click con el botón izquierdo del ratón encima del objeto elegido. Con esta acción aparece en

pantalla el correspondiente diálogo de propiedades conteniendo los atributos definidos para dicho elemento.

Finalmente, falta precisar el funcionamiento y la estructura de los diálogos que muestran las propiedades de los elementos que aparecen en el Diagrama de la Plataforma.

Primero, comentaremos la ventana de la figura 3.7, "Diálogo de Propiedades de un Procesador". Esta ventana contiene un diálogo que permite configurar todos los atributos de un procesador. A continuación explicamos dichos atributos (entre paréntesis y en cursiva aparece la terminología empleada en la ventana de diálogo):

- Tipo *(Processor Type)*: Indica el tipo de procesador. El único tipo que hay definido hasta el momento es el Procesador Regular (*Regular*), que representa un procesador con las sobrecargas asociadas a los servicios de interrupción hardware.
- Nombre *(Processor Name)*: Es una cadena de caracteres que se utiliza como identificador del procesador.
- Factor de velocidad *(Speed Factor)*: Indica la velocidad del procesador. Los tiempos de ejecución de las operaciones van expresados en unidades de tiempo normalizadas, con lo cuál el tiempo de ejecución real se obtiene dividiendo el tiempo de ejecución normalizado entre este factor de velocidad.
- Rango de prioridades de interrupción *(Maximum Interrupt Priority, Minimum Interrupt Priority)*: Representan las prioridades utilizadas para actividades planificadas por rutinas de servicio a la interupción. Su valor es un entero que puede oscilar entre 1 y 32767.

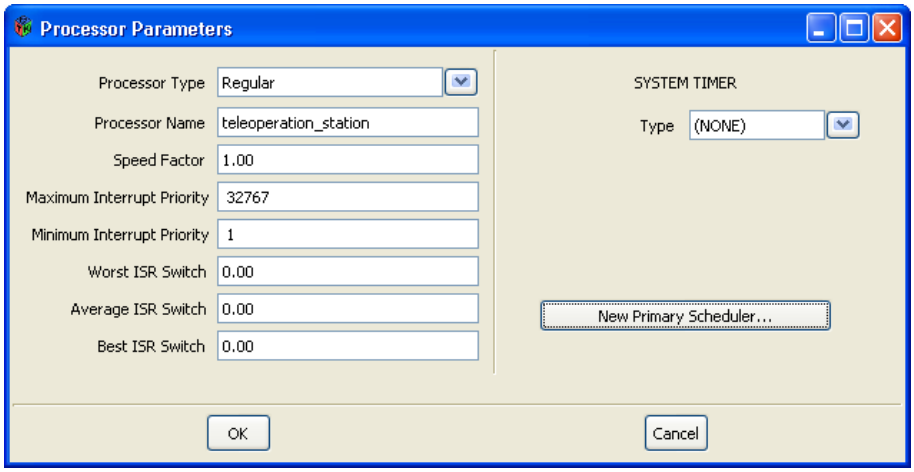

Figura 3.7: Diálogo de Propiedades de un Procesador

- Overheads (peor, medio y mejor) asociados al cambio de rutina de servicio a la interrupción (*ISR Switch (Worst, Average, Best)*): Su valor es un número real indicando el tiempo de cambio de contexto para rutinas de servicio o atención a la interrupción.
- Temporizador del sistema *(System Timer)*: Representa los distintos *overheads* asociados con el modo en que el sistema maneja los eventos temporizados. En función del tipo elegido aparecerán distintos atributos:
  - Tipo *(Type)*: existen varias opciones:
    - Ninguno (*NONE*): No hay ningún temporizador asociado.

- Reloj despertador (*Alarm Clock*): Los eventos temporizados del sistema son activados por una interrupción del temporizador hardware, programado para generar la interrupción en el instante de tiempo del evento temporizado más próximo. Esto conlleva un overhead asociado a la interrupción. Sus atributos son:
  - Overheads (peor, medio y mejor) (*Overheads (Worst, Average, Best)*): Tiempo asociado a la interrupción del temporizador, asumiendo que se ejecuta con la prioridad de interrupción más alta.
- Reloj periódico (*Ticker*): el sistema tiene un reloj periódico, como puede ser una interrupción periódica que llega al sistema. Cuando esta interrupción llega, todos los eventos temporizados cuyo tiempo de expiración haya pasado son activados. Los eventos no temporizados son manejados en el instante en que se generan. El *overhead* introducido por la interrupción del temporizador es localizado en una sola interrupción periódica, pero se introduce una ligera fluctuación para todos los eventos temporizados, debido a que la máxima resolución temporal es el propio período del reloj. Los atributos son:
  - Overheads (peor, medio y mejor) (*Overheads (Worst, Average, Best)*): Su significado es el mismo que el del anterior caso.
  - Período *(Period)*: Período de la interrupción del reloj.

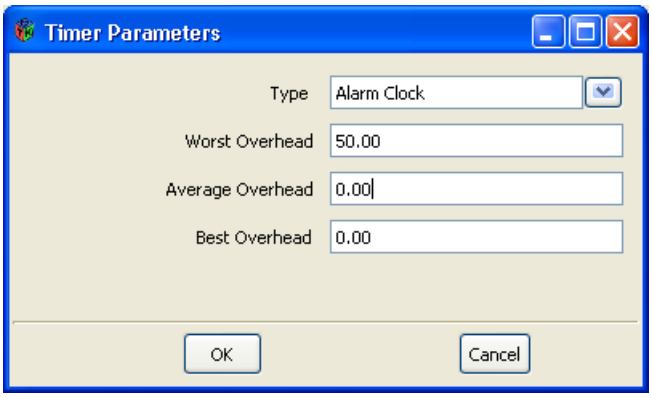

Figura 3.8: Diálogo de Propiedades de un Temporizador

Se pueden consultar o modificar las propiedades del temporizador usando el diálogo de los procesadores, así como pulsando con el ratón sobre el temporizador y seleccionando la opción *Properties* del menú emergente que muestra el diálogo de la figura 3.8 con las propiedades del temporizador.

En la parte derecha del diálogo de propiedades del procesador aparece el botón *New Primary Scheduler...* que sirve para crear un nuevo planificador primario y asociarlo al procesador a través del diálogo de la figura 3.11. Existen otros dos botones llamados *OK* y *Cancel*. Se pueden dar varios casos: por un lado, si el elemento ya existía previamente, pulsando *OK* se actualizan sus propiedades, mientras que pulsando *Cancel* el elemento permanece sin cambios, y por otro lado, si el elemento no existe, pulsando *OK* se crea dicho elemento y se añade al sistema, mientras que si se pulsa *Cancel* no se hace nada.

Si se cambia el tipo, se borra el anterior elemento y se añade uno nuevo al sistema con el nuevo tipo especificado.

En el caso de la ventana de la figura 3.9, "Diálogo de Propiedades de una Red", podemos decir que su comportamiento es análogo al de la anterior sólo que en ésta aparecen los atributos de una red, los cuales comentamos a continuación:

- Tipo *(Network Type)*: Indica el tipo de red. Sólo hay un tipo definido: la Red basada en paquetes (*Packet Based Network*) que representa una red que utiliza algún protocolo de tiempo real basado en paquetes no expulsables para transmitir mensajes en la red. Hay redes que soportan prioridades en sus protocolos estándar (p. ej., CAN bus) y otras redes que precisan de un protocolo adicional de nivel superior que trabaja por encima de los protocolos estándar (p. ej., ethernet, líneas serie).
- Nombre *(Network Name)*: Cadena de caracteres que identifica a la red.
- Factor de velocidad *(Speed Factor)*: Indica la velocidad de la red. Los tiempos de ejecución de las operaciones van expresados en unidades de tiempo normalizadas, con lo cuál el tiempo de ejecución real se obtiene dividiendo el tiempo de ejecución normalizado entre este factor de velocidad.
- Tipo de transmisión (*Transmission Kind*): Indica si la comunicación es Simplex, Half Duplex o Full Duplex.
- Máximo bloqueo (*Maximum Blocking*): Tiempo máximo de bloqueo causado por la no expulsión de los paquetes de mensaje. Normalmente, tiene el mismo valor del tiempo máximo de transmisión de paquete, pero su valor por defecto es cero, para el caso en que el *overhead* de la red es despreciable.
- Ancho de Banda (*Throughput)*: Ancho de banda normalizado en bits por unidad de tiempo.

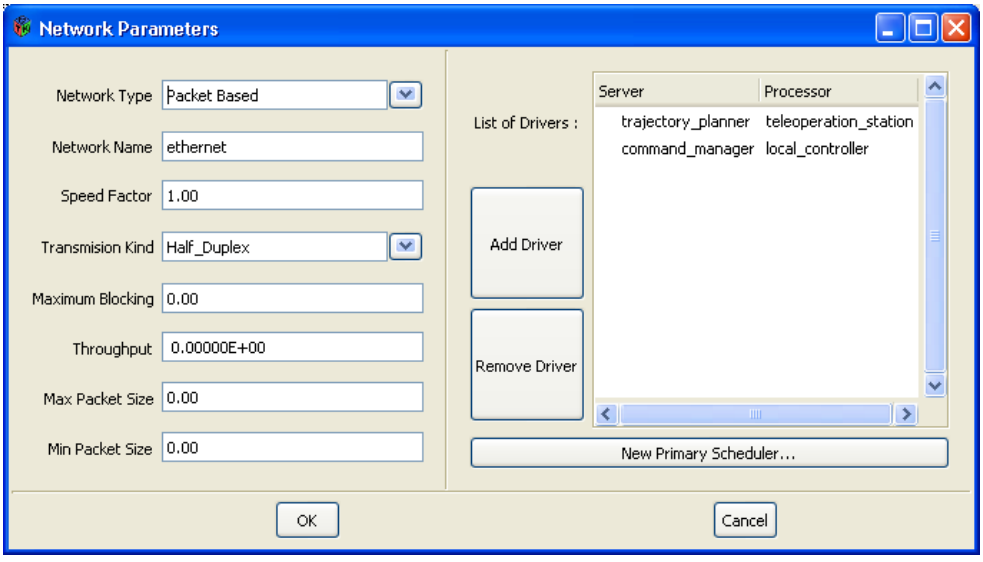

Figura 3.9: Diálogo de Propiedades de una Red

- Tamaños de paquete máximo y mínimo (*Max Packet Size* y *Min Packet Size*): Describen la cantidad de información incluida en un paquete, excluyendo la información de protocolo. El tamaño máximo se usa para calcular el número de paquetes en los que se trocea un mensaje, calculado como el entero máximo resultante de dividir el tamaño del mensaje entre el tamaño máximo del

paquete. Este número se multiplica por el tiempo de overhead del paquete del planificador para calcular la sobrecarga de la red. El tamaño mínimo se usa para calcular el periodo más corto de los overheads asociados con la transmisión de cada paquete, y por lo tanto tiene un gran impacto en la sobrecarga causada por los controladores de red en los procesadores que la utilizan.

- Lista de controladores (*List of Drivers*): Representan las operaciones ejecutadas en un procesador como consecuencia de la transmisión o recepción de un mensaje o un paquete a través de la red. Se listan indicando el servidor de planificación y el procesador al que está asociado cada controlador.

En la parte derecha del diálogo de propiedades de la red aparece el botón *New Primary Scheduler...* que sirve para crear un nuevo planificador primario y asociarlo a la red a través del diálogo de la figura 3.11. También aparecen en esta ventana los botones *OK* y *Cancel*, su comportamiento es análogo al de los mencionados anteriormente, sólo que en este caso en lugar de procesadores se actualizan o añaden redes en el sistema.

Merecen una mención especial los dos botones que aparecen en la parte central de la ventana: *Add Driver* y *Remove Driver.*

El botón *Remove Driver* permite eliminar el controlador seleccionado por el usuario de la lista de controladores de la red.

Por su parte, el botón *Add Driver* permite añadir y configurar un nuevo controlador a la red. Cuando se pulsa aparece una nueva ventana como la de la figura 3.10. En ella aparecen los atributos de un controlador:

- Tipo *(Driver Type)*: Indica el tipo de controlador. Sólo hay dos tipos definidos:
  - Controlador de paquetes (*Packet Driver*): El controlador se activa en cada transmisión o recepción de un mensaje. Sus atributos son:
    - Servidor de paquetes (*Paquet Server*): Servidor de planificación que ejecuta el controlador (que a su vez lleva asociado un procesador y unos parámetros de planificación). Para asignar el servidor al controlador hay dos posibilidades:
      - Crear un nuevo servidor pulsando el botón *New Server...* y configurando sus atributos en el diálogo de la figura 3.14.
      - Seleccionar el nombre de un servidor ya existente de la lista de servidores.
    - Operación de envío de paquete *(Paquet Send Operation*): Operación ejecutada cada vez que se envía un paquete. Se puede definir la operación de dos formas:
      - Creando una nueva operación. Para ello se ha de pulsar el botón *New Operation...* y configurar sus atributos en el diálogo de la figura 3.22.
      - Seleccionando el nombre de una operación ya existente de la lista de operaciones.
    - Operación de recepción de paquete *(Paquet Receive Operation*): Operación ejecutada cada vez que se recibe un paquete.

    - Particionado de paquete *(Message Partitioning)*: Indica si el *driver* es capaz de particionar mensajes de gran longitud y reconstruirlos en el destino. El valor por defecto es *Yes*.
- Controlador carácter-paquete (*Character Packet Driver*): Es una especialización del *packet driver* en el que hay un *overhead* adicional asociado a la transmisión de cada carácter como ocurre en algunas líneas serie. Sus atributos son los del *packet driver* incluyendo además:
    - Servidor de caracteres (*Character Server*): Servidor de planificación que ejecuta la parte del controlador asociada al envío o recepción de cada carácter.
    - Operación de envío de carácter *(Character Send Operation)*: Operación ejecutada cada vez que un carácter es enviado.
    - Operación de recepción de carácter *(Character Receive Operation)*: Operación ejecutada cada vez que un carácter es recibido.
    - Tiempo de transmisión de carácter *(Character Transmission Time)*: Tiempo empleado en la transmisión de un carácter.

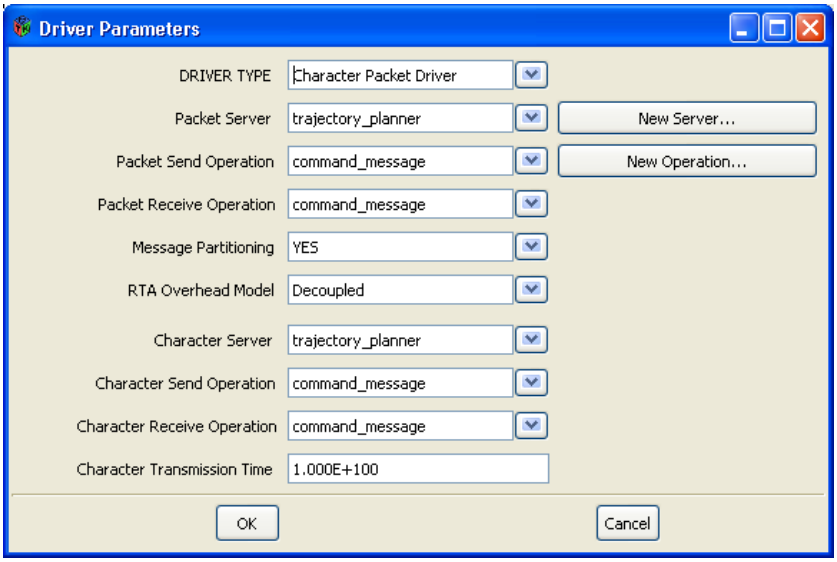

Figura 3.10: Diálogo de Propiedades de un Controlador

- Controlador de paquetes Real-Time Ethernet Protocol (*RT-EP Packet Driver*): Es una especialización del *packet driver* en el que hay un *overhead* adicional asociado al paso de testigo. Sus atributos son los del *packet driver* incluyendo además:
    - Modelo de sobrecarga para el análisis del tiempo de respuesta (*RTA Overhead Model)*: Es un atributo exclusivo de la herramienta de análisis del tiempo de respuesta en el peor caso. El valor determina el modelo de overherad que será utilizado para el driver. Los posibles valores son:
        - *Coupled*: En este modelo de *overhead* una operación de envío y recepción de mensaje se adjuntan a la transacción que causa la transmisión.

- *Decoupled*: En este modelo, la operación de envío y recepción se ejecutan periódicamente con un periodo igual al tiempo de transmisión mínimo (o a algún otro tiempo dependiente del driver).

- Número de estaciones *(Number of Stations*): Corresponde al número de estaciones o procesadores conectados a través de la red.
- Demora de testigo *(Token Delay*): Es un retardo que se introduce en el procesamiento del testigo para así poder reducir el *overhead* de CPU.
- *Failure Timeout*: Es el tiempo de timeout configurado en el protocolo para el mecanismo de detección de errores.
- Reintentos de transmisión del testigo *(Token Transmission Retries)*: Número máximo de fallos, y sus retransmisiones, que se permitiren en cada vuelta de testigo.
- Reintentos de transmisión de paquete (*Packet Transmission Retries)*: Corresponde al número de fallos que se permiten al transmitir un paquete de información.
- Servidor de interrupción de paquete *(Packet Interrupt Server*): Es el servidor que está ejecutando la rutina de interrupción que almacena cada trama recibida en la red independientemente de si se trata de un paquete de información o de arbitrio.
- Operación de rutina de servicio a la interrupción de paquete *(Packet ISR Operation*): Corresponde con la operación ejecutada por el servidor de interrupción cada vez que se recibe una trama, independientemente de la naturaleza del paquete.
- Operación de chequeo de testigo *(Token Check Operation*): Es la operación que se ejecuta para recibir y chequear un paquete de testigo.
- Operación de manejo de testigo *(Token Manage Operation*): Es la operación que se ejecuta para mandar el testigo.
- Operación de descarte de paquete *(Packet Discard Operation*): Es la operación que se ejecuta cuando se reciben, debido al modo promiscuo, y descartan tramas dirigidas a otro destino.
- Operación de retransmisión de testigo (*Token Retransmission Operation)*: Representa la operación que se ejecuta cada vez que se retransmite un testigo.
- Operación de retransmisión de paquete (*Packet Retransmission Operation)*: Corresponde con la operación que se ejecuta cada vez que se retransmite un paquete de información.

En la pantalla de la figura 3.10, “Diálogo de Propiedades de un Controlador” aparecen además dos botones: si se pulsa el botón *OK* el controlador se añade a la lista de controladores de la red, si por el contrario se pulsa *Cancel* entonces la lista de controladores de la red permanece sin cambios. Es preciso señalar que para configurar un controlador es necesario haber definido previamente algún servidor y alguna operación en el sistema ya que si no será imposible asignar los atributos del controlador correctamente.

Si pulsamos en el botón *New Primary Scheduler* de la parte inferior de la Capa de Recursos de Procesado y Planificadores aparece una ventana como la de la figura figura 3.11 con los parámetros de un planificador primario que explicamos a continuación:

- Nombre *(Name)*: Cadena de caracteres que identifica al planificador
- Anfitrión *(Host)*: Referencia al recurso de procesado asociado.
- Política *(Policy)*: Define la política de planificación implementada por el planificador para repartir la capacidad que tiene asignada entre los servidores de planificación que hay asociados a él. Existen tres tipos:
  - Prioridad fija (*Fixed Priority*): Representa una política de prioridades fijas. Posee los siguientes atributos:
    - Overheads (peor, medio y mejor) asociados al cambio de contexto (*Context Switch (Worst, Average, Best)*): Indican los overheads asocidados al cambio de contexto.
    - Prioridades máxima y mínima *(Max Priority* y *Min Priority)*: Indican el rango de prioridades válidos para las operaciones sobre los servidores de planificación planificados con esta política.
  - EDF (*Earliest Deadline First*): Sus atributos son:
    - Overheads (peor, medio y mejor) asociados al cambio de contexto *(Context Switch (Worst, Average, Best*)): Indican los overheads asocidados al cambio de contexto.
  - Prioridad fija basada en paquetes (*Fixed Priority Packet Based*): Representa una política de prioridades fijas utilizada en una red de comunicaciones por paquetes. Se supone que los paquetes de la red no son expulsables.
    - Overheads (peor, medio y mejor) asociados al paquete *(Packet Overhead (Worst, Average, Best)*): Este es el (peor, medio y mejor) *overhead* asociado a cada paquete producido por los mensajes de protocolo o cabeceras que deben ser mandados antes o después de cada paquete útil.

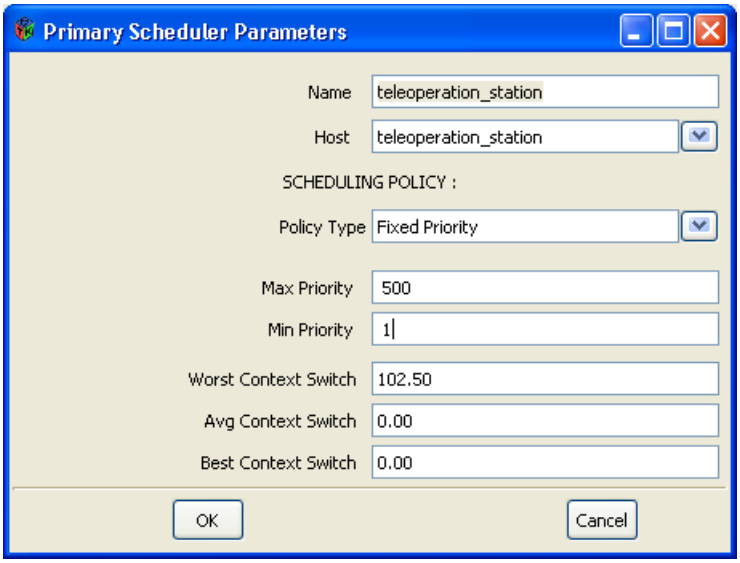

Figura 3.11: Diálogo de Propiedades de un Planificador Primario

- Prioridades máxima y mínima *(Max Priority* y *Min Priority)*: Indican el rango de prioridades válidos para los mensajes enviados con esta política.

En la parte inferior del diálogo de propiedades del planificador primario aparecen dos botones: *OK* y *Cancel*. Si el elemento ya existía previamente, pulsando *OK* se actualizan sus propiedades, mientras que pulsando *Cancel* el elemento permanece sin cambios. Por otro lado, si el elemento no existe, pulsando *OK* se crea dicho elemento y se añade al sistema, mientras que si se pulsa *Cancel* no se hace nada.

Si pulsamos en el botón *New Secondary Scheduler* aparece una ventana como la de la figura 3.12, "Diálogo de Propiedades de un Planificador Secundario" con los parámetros de un planificador secundario que explicamos a continuación:

- Nombre *(Name)*: Cadena de caracteres que identifica al planificador
- Servidor *(Server)*: Referencia al servidor de planificación asociado.
- Política *(Policy)*: Define la política de planificación implementada por el planificador para repartir la capacidad que tiene asignada entre los servidores de planificación que hay asociados a él. Hay dos opciones:
  - Prioridad fija (*Fixed Priority*): Representa una política de prioridades fijas. Posee los siguientes atributos:
    - Overheads (peor, medio y mejor) asociados al cambio de contexto (*Context Switch (Worst, Average, Best)*): Indican los overheads asocidados al cambio de contexto.
    - Prioridades máxima y mínima *(Max Priority* y *Min Priority)*: Indican el rango de prioridades válidos para las operaciones sobre los servidores de planificación planificados con esta política.
  - EDF (*Earliest Deadline First*): Sus atributos son:
    - Overheads (peor, medio y mejor) asociados al cambio de contexto *(Context Switch (Worst, Average, Best)*): Indican los overheads

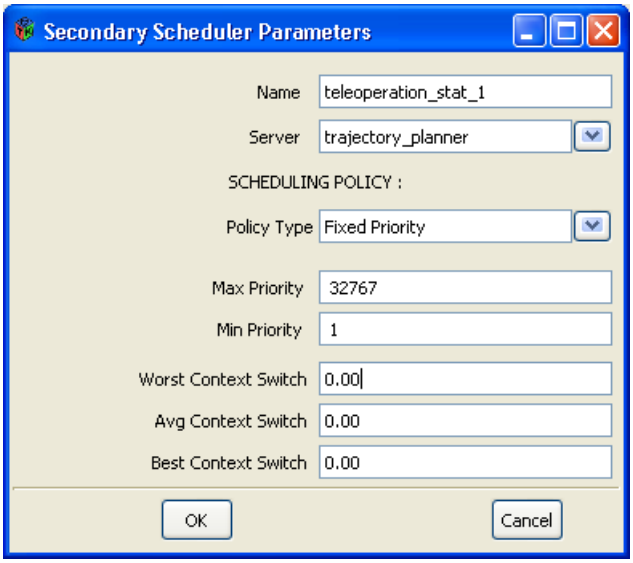

Figura 3.12: Diálogo de Propiedades de un Planificador Secundario

El comportamiento de los botones *OK* y *Cancel* es similar al visto para los planificadores primarios. Si el elemento ya existía previamente, pulsando *OK* se actualizan sus propiedades, mientras que pulsando *Cancel* el elemento permanece sin cambios. Si el elemento no existe, pulsando *OK* se crea dicho elemento y se añade al sistema, mientras que si se pulsa *Cancel* no se hace nada.

Pulsando con el ratón sobre los planificadores podemos ver el menú emergente que nos permite modificar o consultar las propiedades de planificadores a través de los diálogos correspondientes o borrarlos del sistema.

3.3.3. Capa de Servidores de Planificación

Dentro de esta capa se visualizan los servidores de planificación que modelan procesos, threads o tareas en las que se planifican las actividades del sistema. Cada servidor está asociado a un recurso de procesado (un procesador o una red) y contiene unos parámetros de planificación que determinan la política utilizada.

Se puede observar el aspecto de esta capa en la figura 3.13. Su apariencia es similar a la expuesta en la Capa de Recursos de Procesado, ya que se divide en dos áreas: una zona de dibujo en la que aparece el **Diagrama de los Servidores de Planificación** *(Scheduling Servers Diagram)* y una barra de herramientas.

En el Diagrama de los Servidores de Planificación los **servidores** *(servers)* se representan mediante cajas de color naranja en las que se especifican algunos de sus atributos (nombre, tipo, etc.). Asimismo, también se incluyen en el diagrama los planificadores asociados a dichos servidores, con el objetivo de dar una visión más completa del modo en que están relacionados los servidores con los planificadores.

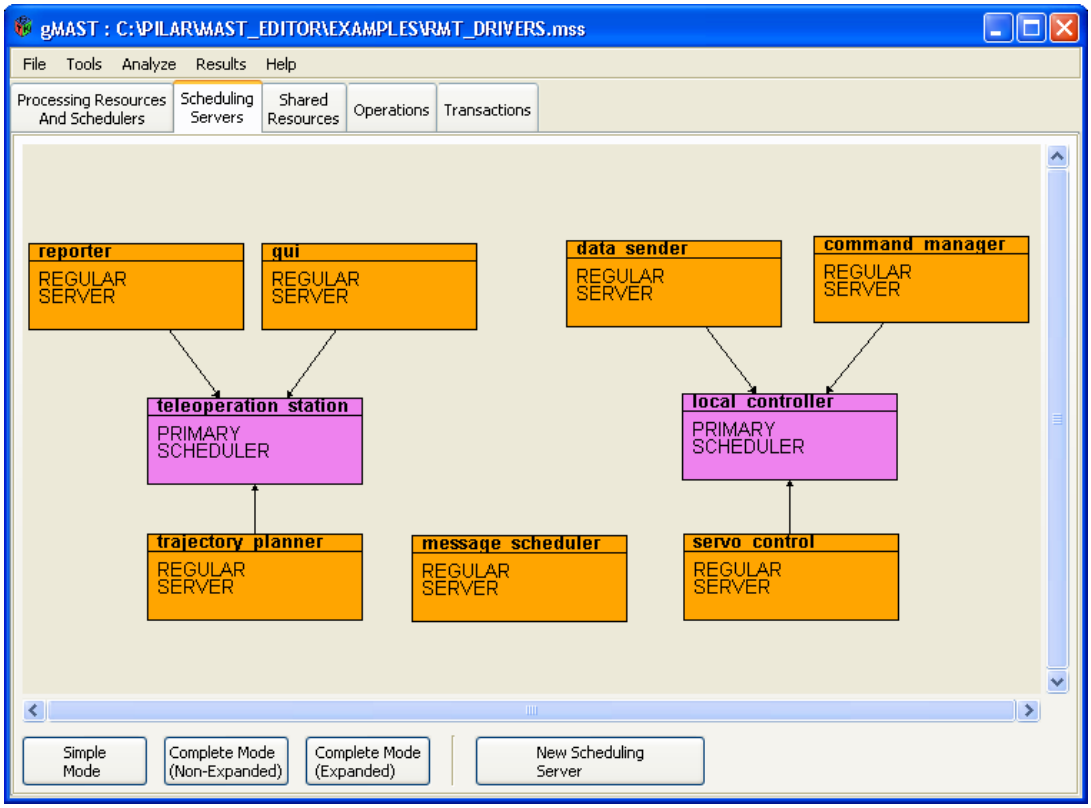

Figura 3.13: Capa de Servidores de Planificación

La **barra de herramientas** de la parte inferior de la ventana se dividide en dos grupos: por un lado, tres botones referentes al modo de visualización del diagrama y, por otro lado, el botón de creación de nuevos servidores. La función de los tres primeros *(Simple Mode, Complete Mode Non Expanded* y *Complete Mode Expanded)* es determinar el nivel de detalle con el que se dibujan los elementos del diagrama de la misma forma que se explica en el apartado 3.3.2, "Capa de Recursos de Procesado y Planificadores", mientras que el botón **Nuevo Servidor de Planificación** *(New Scheduling Server)* permite añadir un nuevo servidor al sistema mostrando una ventana auxiliar que contiene un diálogo con las propiedades del servidor (ver figura 3.14).

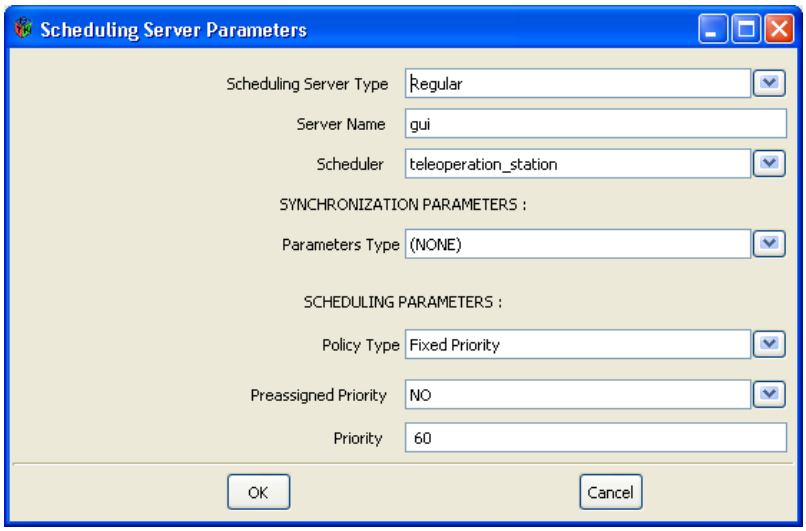

Figura 3.14: Diálogo de Propiedades de un Servidor de Planificación

Si el usuario necesita eliminar un elemento, consultar o modificar sus propiedades lo puede hacer mediante el **menú emergente** creado para tal fin y que es común a todas las capas de la forma en que se explicó en el apartado 3.3.2. También se pueden consultar las propiedades de un elemento del diagrama colocando el puntero del ratón encima de dicho elemento y haciéndo doble click con el botón izquierdo.

A continuación comentamos el funcionamiento de la ventana de la figura 3.14, "Diálogo de Propiedades de un Servidor de Planificación". Este diálogo muestra los atributos de un servidor (entre paréntesis aparece la terminología empleada en la ventana):

- Tipo *(Scheduling Server Type)*: Indica el tipo de servidor. Sólo hay un tipo definido actualmente: el Servidor de prioridades fijas (*Fixed Priority*), que representa un servidor con política de planificación basada en prioridades fijas.
- Nombre *(Name)*: Cadena de caracteres que identifica al servidor.
- Planificador *(Scheduler)*: Planificador asociado el servidor.
- Parámetros de sincronización (*Synchronization Parameters)*: Parámetros usados por el servidor cuando está efectuando un acceso a un recurso compartido en modo mutuamente exclusivo. Deben ser especificados siempre que los parámetros de la política de planificación no aporten información suficiente para los protocolos de sincronización usados. Hay un solo tipo:
  - *Stack Resource Protocol*: Sus atributos son los siguientes:
    - Nivel de expulsión *(Preemption Level)*: Nivel de explusión usado para el recurso. Puede se calculado por las herramientas MAST si se quiere.

- Preasignado *(Preassigned)*: Si a este parámetro se le asigna el valor *"NO",* el nivel de expulsión se puede asignar mediante una herramienta MAST. Si no es así, el nivel es fijo y no puede modificarse con dicha herramienta. Su valor por defecto es *"NO"*.

- Parámetros de Planificación (*Scheduling Parameters)*: Políticas y parámetros de planificación del servidor. Hay dos tipos:
  - De prioridades fijas: Sus atributos comunes son los siguientes:
    - Prioridad *(Priority):* Número natural entre 1 y 32767 que representa la prioridad de planificación.
    - Preasignada *(Preassigned Priority)*: Si a este parámetro se le asigna el valor *"NO",* la prioridad se puede asignar mediante las herramientas de asignación de prioridades. Si no es así, la prioridad es fija y no puede modificarse con dichas herramientas. Su valor por defecto es *"NO"*.
    - Tipo de política *(Policy Type)*: Indica el tipo de política de planificación. Existen varias clases, algunas con atributos adicionales:
      - Basada en prioridades fijas no expulsora (*Non Preemptible Fixed Priority)*: No tiene atributos adicionales.
      - Basada en prioridades fijas *(Fixed Priority)*: Política de prioridades fijas expulsora. No tiene atributos adicionales.
      - Basada en prioridades fijas de interrupción *(Interrupt Fixed Priority)*: Representa una rutina de servicio a la interrupción. No tiene atributos adicionales. El atributo *Preassigned* no puede tener el valor *"NO"*, ya que las prioridades de las interrupciones siempre están preasignadas.
      - De muestreo periódico *(Polling)*: Representa una tarea periódica que pregunta por la llegada de su evento de entrada. La ejecución del evento puede ser pospuesta hasta el siguiente período. Sus atributos adicionales son:
        - Overheads de muestreo periódico (peor, medio y mejor) (*Polling Overheads (Worst, Average, Best)*): *Overhead* asociado a la tarea de muestreo.
        - Período de muestreo (*Polling Period*): Período de la tarea de muestreo.
      - De servidor esporádico *(Sporadic)*: Representa una tarea planificada según el algoritmo de planificación de servidor esporádico. Sus atributos adicionales son:
        - Prioridad de fondo (*Background Priority*): Representa la prioridad a la que se ejecuta la tarea cuando no hay capacidad de ejecución disponible.
        - Capacidad inicial (*Initial Capacity*): Valor inicial de la capacidad de ejecución.

- Período de relleno (*Replenishment Period*): Período después del cual una porción de la capacidad de ejecución consumida es repuesta.
- Relleno pendiente máximo (*Maximum Pending Replenishments)*: Número máximo de operaciones de reposición pendientes simultáneamente.

- EDF *(Earliest Deadline First)*: Sus atributos son los siguientes:
  - Plazo *(Deadline):* Plazo relativo del servidor de planificación asociado.
  - Preasignado *(Preassigned Deadline)*: Si a este parámetro se le asigna el valor *"NO",* el plazo se puede asignar mediante las herramientas de asignación de plazos. Si no es así, el plazo es fijo y no puede modificarse con dichas herramientas. Su valor por defecto es *"NO"*.

Al final de la ventana se muestran dos botones: *OK* y *Cancel*. Si el servidor ya existía previamente, pulsando *OK* se actualizan sus propiedades, mientras que pulsando *Cancel* permanece sin cambios, si no existe, pulsando *OK* se crea el servidor y se añade al sistema, mientras que si se pulsa *Cancel* no se hace nada.

3.3.4. Capa de Recursos Compartidos

La estructura de esta capa es muy similar a la de las anteriores. Dentro de ella se visualiza el esquema de recursos del sistema que son compartidos entre distintas tareas y que deben ser usados de un modo mutuamente exclusivo. Es importante tener en cuenta que sólo se permiten los protocolos que evitan las inversiones de prioridad no acotadas.

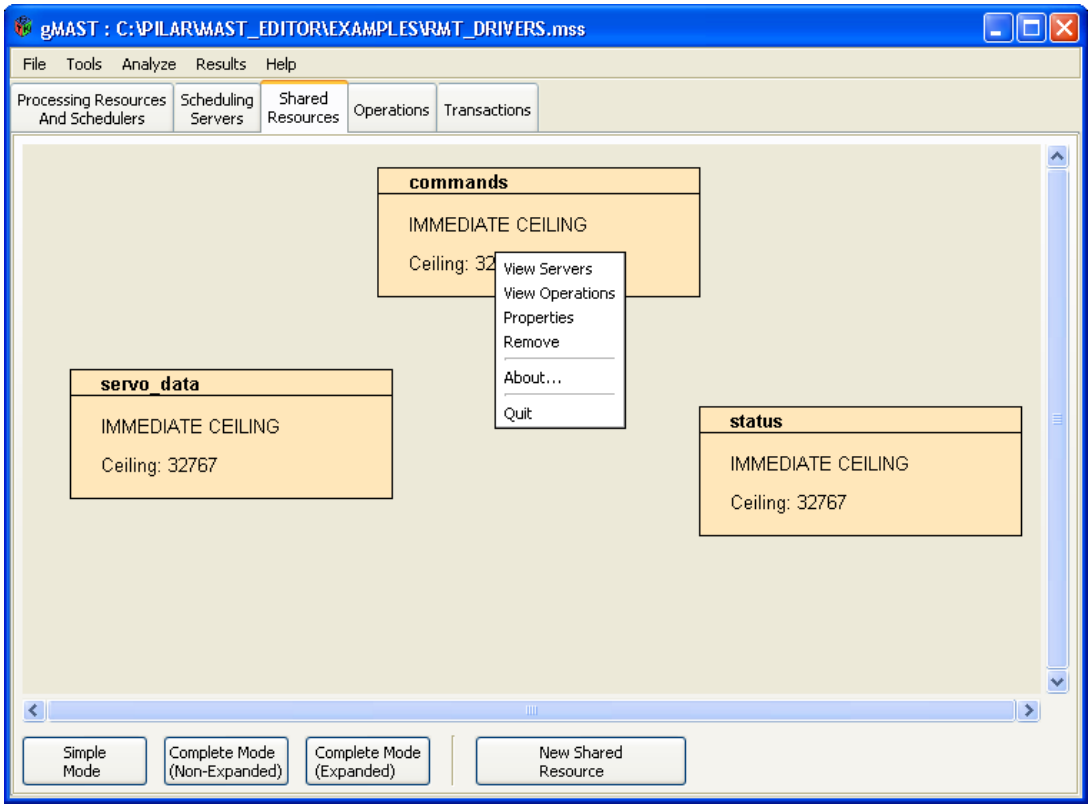

Figura 3.15: Capa de Recursos Compartidos

Como se puede observar en la figura 3.15, la capa se divide en dos partes: un área de dibujo y una barra de herramientas. En este caso, el área de dibujo contiene el **Diagrama de Recursos Compartidos** *(Shared Resources Diagram)* en el que aparecen los diferentes **recursos compartidos** *(shared resources)* del sistema representados por cajas de color beige, en las cuales se muestran los atributos de los recursos.

De forma análoga a las otras capas, la **barra de herramientas** se dividide en dos partes: primeramente, están los botones *Simple Mode, Complete Mode Non Expanded* y *Complete Mode Expanded* que ajustan el modo de visualización del diagrama y, por otro lado, está el botón para la creación de nuevos recursos. El comportamiento de los tres primeros ya se explicó en el apartado 3.3.2, "Capa de Recursos de Procesado y Planificadores". El botón **Nuevo Recurso Compartido** *(New Shared Resource)* permite crear un nuevo recurso. Al pulsarlo se muestra una ventana que contiene un diálogo con las propiedades del recurso compartido como se puede observar en la figura 3.18.

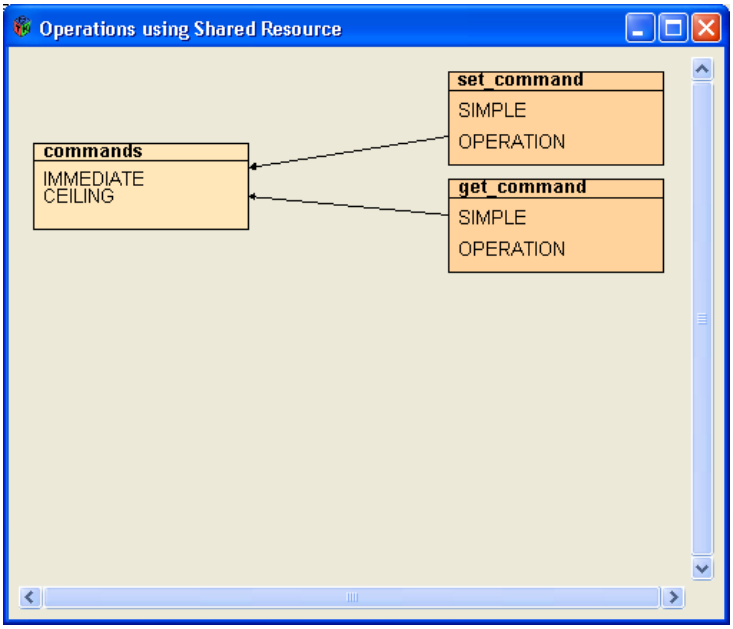

Figura 3.16: Ventana auxiliar de las operaciones asociadas al recurso

Para eliminar, consultar o cambiar las propiedades de un recurso compartido, el usuario puede utilizar el **menú emergente** cuyo comportamiento se explicó en el apartado 3.3.2. En esta capa el menú emergente incluye además otras dos opciones llamadas *View Operations* y *View Servers* que muestran ventanas auxiliares con las operaciones y servidores asociados al recurso respectivamente (ver figuras 3.16 y 3.17). También se puede consultar sus propiedades del recurso colocando el puntero del ratón encima del recurso y haciendo doble click con el botón izquierdo.

Por último, comentamos la estructura de la ventana de la figura 3.18, "Diálogo de Propiedades de un Recurso Compartido". Este diálogo muestra los atributos de un recurso compartido (entre paréntesis se muestra la terminología usada en dicha ventana):

- Tipo *(Type)*: Indica de qué tipo de recurso compartido se trata. Existen varios tipos definidos:
  - Recurso de techo de prioridad inmediato *(Immediate Ceiling Resource)*: Utiliza el protocolo de techo de prioridad. Sus atributos son:

- Nombre *(Name)*: Identificador del recurso.
- Techo *(Ceiling Priority)*: Techo de prioridad usado para el recurso. Puede ser calculado automáticamente por la herramienta MAST destinada a tal fin.
- Preasignado *(Preassigned)*: Si a este parámetro se le asigna el valor *"NO"*, el techo de prioridad se puede asignar mediante la herramienta de cálculo de techos de prioridad. Si no es así, el techo de prioridad es fijo y no puede modificarse con dicha herramienta. Su valor por defecto es *"NO"*.

- Recurso de herencia de prioridad *(Priority Inheritance Resource)*: Utiliza el protocolo de herencia de prioridad. Su único atributo es el nombre *(Name)*.

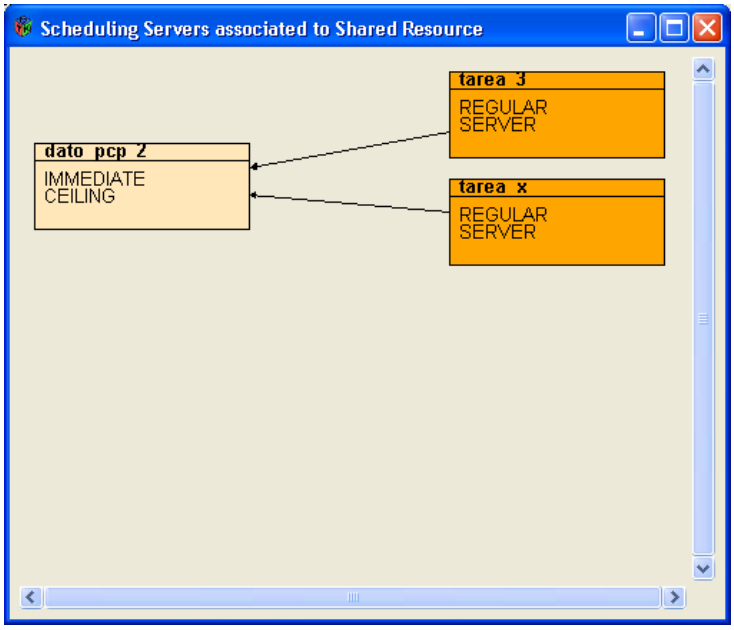

Figura 3.17: Ventana auxiliar con los servidores asociados al recurso compartido

- *Stack Resource Protocol*: Utiliza un protocolo similar al protocolo de techo de prioridad pero que funciona para políticas no basadas en prioridades. Sus atributos son:
  - Nombre *(Name)*: Identificador del recurso.

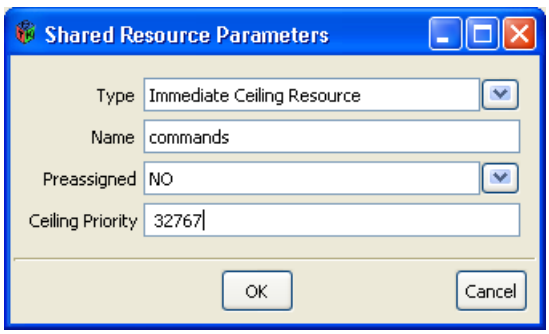

Figura 3.18: Diálogo de Propiedades de un Recurso Compartido

- Nivel de expulsión *(Preemption Level)*: Nivel de explusión usado para el recurso. Puede se calculado por las herramientas MAST si se quiere.
- Preasignado *(Preassigned)*: Si a este parámetro se le asigna el valor *"NO"*, el nivel de expulsión se puede asignar mediante una herramienta MAST. Si no es así, el nivel es fijo y no puede modificarse con dicha herramienta. Su valor por defecto es *"NO"*.

También se incluyen dos botones al final de la ventana: *OK* y *Cancel*. Si el recurso ya existía previamente, pulsando *OK* se actualizan sus propiedades, mientras que pulsando *Cancel* permanece sin cambios, si el recurso no existe, pulsando *OK* se crea y se añade al sistema, mientras que si se pulsa *Cancel* no se hace nada.

3.3.5. Capa de Operaciones

La Capa de Operaciones se utiliza para dibujar y esquematizar la disposición e interrelación entre las operaciones del sistema modelado. Dentro del modelo MAST, las operaciones representan una parte de código a ejecutar por un procesador o un mensaje que se envía por una red.

La estructura de la capa se puede observar la figura 3.19. El aspecto que presenta es similar al de las demás capas, ya que también se divide en dos áreas: una zona de dibujo en la que aparece el **Diagrama de las Operaciones** *(Operations Diagram)* y una barra de herramientas.

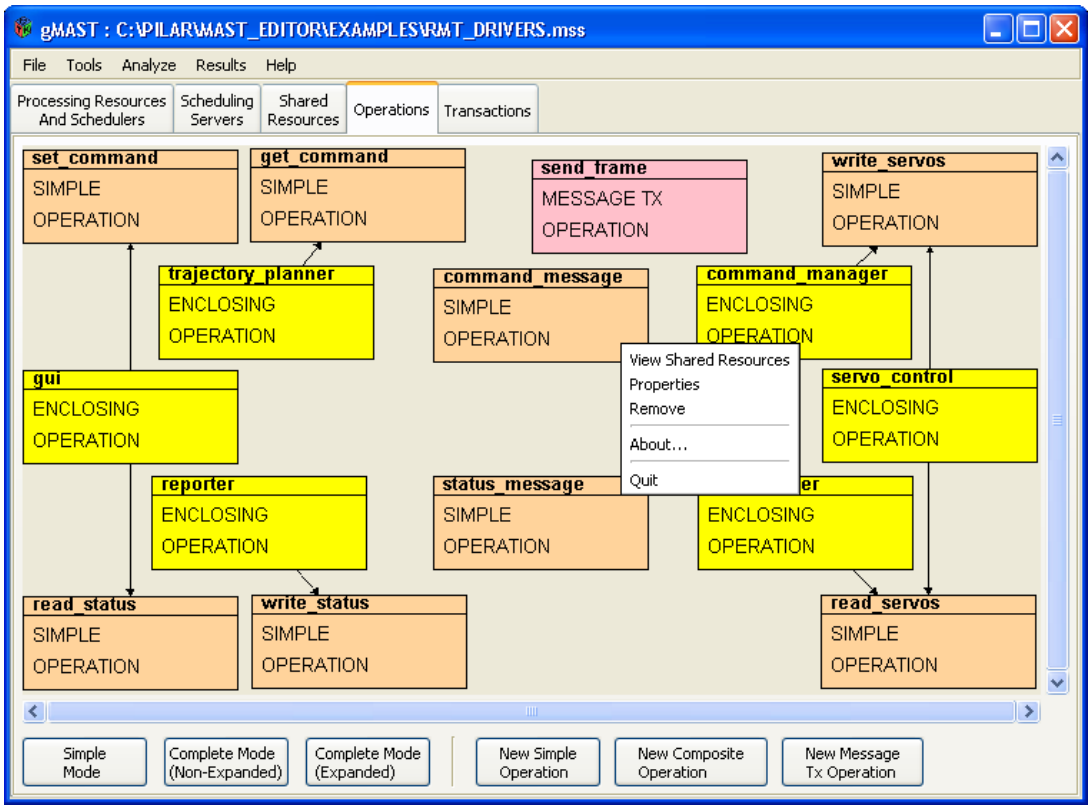

Figura 3.19: Capa de Operaciones

Para elaborar el diagrama hemos dividido las operaciones en tres grupos: **operaciones simples** *(simple operations)*, **operaciones compuestas** *(composite*

operations) y **operaciones de transmisión de mensaje** *(message transmission operations)*. Las operaciones simples representan un trozo de código sencillo o un mensaje y se dibujan con una caja de color marrón en la que se especifican sus atributos. Las operaciones compuestas representan una operación que contiene una secuencia ordenada de otras operaciones, simples o compuestas. Aparecen en el dibujo como una caja de color amarillo con sus propiedades y se asocian a las operaciones simples que las componen por medio de flechas. Las operaciones de transmisión de mensaje representan un mensaje a transmitir a través de una red y se dibujan con una caja de color sepia. Los atributos de cada uno de los tipos de operación se comentarán a lo largo de este capítulo.

También en este caso la **barra de herramientas** se dividide en dos grupos: el primero contiene los botones de definición del nivel de detalle del diagrama *(Simple Mode, Complete Mode Non Expanded* y *Complete Mode Expanded*) y, el segundo contiene dos botones para la creación de nuevas operaciones. El comportamiento de los tres primeros ya se ha explicado en apartados anteriores, así que comentaremos sólo los botones del segundo grupo:

- **Nueva Operación Simple** *(New Simple Operation)*: Permite crear una nueva operación simple. Al pulsarlo se muestra una ventana que contiene las propiedades de la operación simple (ver figura 3.21).
- **Nueva Operación Compuesta** *(New Composite Operation)*: Permite crear una nueva operación compuesta. Al pulsarlo aparece una ventana con las propiedades de las operaciones compuestas (ver figura 3.23).
- **Nueva Operación de Transmisión de Mensaje** *(New Message Tx Operation)*: Permite crear una nueva operación de transmisión de mensaje. Al pulsarlo aparece una ventana con las propiedades de este tipo de operación (ver figura 3.25).

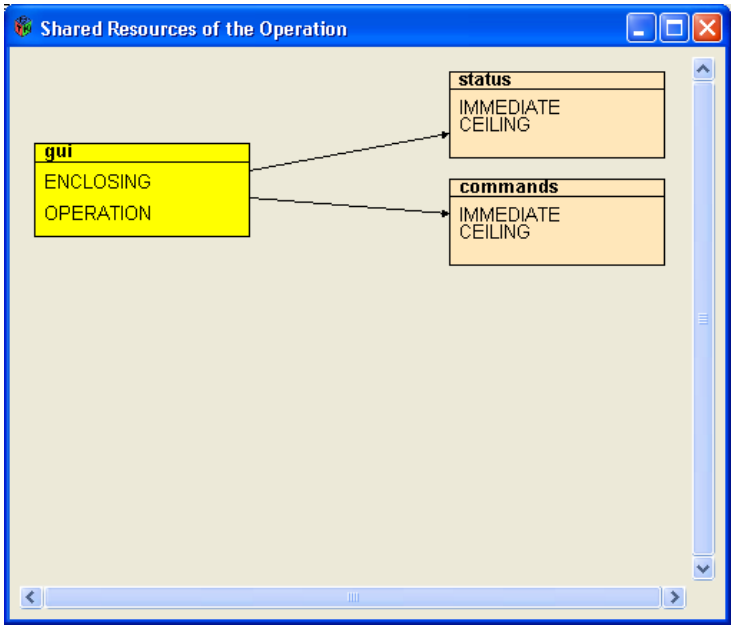

Figura 3.20: Ventana auxiliar con los recursos compartidos de la operación

A la hora de eliminar, consultar o cambiar las propiedades de una operación, se proporciona la funcionalidad de un **menú emergente** con diferentes opciones que permiten dichas acciones, como se explicó en el apartado 3.3.2. Si bien la consulta de

propiedades se puede hacer colocando el puntero del ratón encima de la operación elegida y haciendo doble click con el botón izquierdo.

Para las **operaciones simples** y **compuestas** el **menú emergente** contiene una nueva opción llamada *View Shared Resources* (ver figura 3.19). Con ella se muestra una ventana en la que aparece un diagrama auxiliar con los Recursos Compartidos asociados a la operación (ver figura 3.20).

Como complemento a esta capa se ha visto que aparecen algunas ventanas auxiliares invocadas desde la barra de herramientas o desde los elementos del diagrama de operaciones. Una de estas ventanas es la de la figura 3.21, "Diálogo de Propiedades de una Operación Simple". Esta ventana contiene un diálogo que permite configurar todos los atributos de una operación simple, los cuales explicamos a continuación (entre paréntesis y en cursiva aparece la terminología empleada en la ventana):

- Tipo *(Operation Type)*: Indica el tipo de operación, en este caso solo hay definido un tipo: Simple.
- Nombre *(Operation Name)*: Cadena de caracteres que identifica la operación.
- Parámetros de planificación sobreescritos *(Overridden Scheduling Parameters)*: Representa un nivel de prioridad superior al nivel de prioridad normal al cual la operación se ejecutaría.

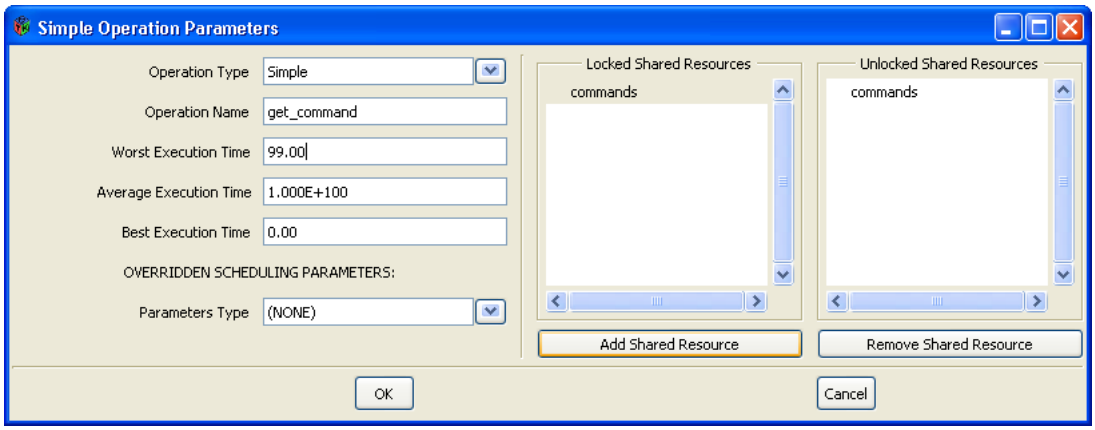

Figura 3.21: Diálogo de Propiedades de una Operación Simple

- Tiempos de ejecución (peor, medio y mejor) *(Execution Time (Worst, Average Best))*: Se expresa en unidades de tiempo normalizadas. Para los mensajes, representa el tiempo de transmisión.
- Recursos compartidos a bloquear (*Shared Resources To Lock*): Lista de recursos compartidos que se han de bloquear antes de ejecutar la operación.
- Recursos compartidos a liberar (*Shared Resources To Unlock*): Lista de recursos compartidos que se han de liberar después de ejecutar la operación.

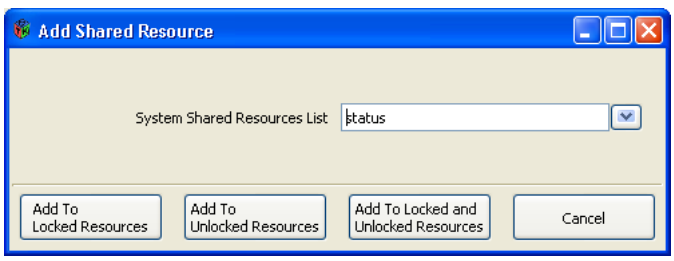

Figura 3.22: Diálogo para Añadir Recursos Compartidos a una Operación

La función de los botones *OK* y *Cancel* es la de validar o no los cambios realizados en las propiedades de la operación. Si la operación ya existe, pulsando *OK* se actualizan sus propiedades, mientras que pulsando *Cancel* ésta permanece sin cambios, si no existe, pulsando *OK* se crea la operación y se añade al sistema, mientras que si se pulsa *Cancel* no se realiza ningún cambio en el sistema.

Existe la posibilidad de, una vez creada una operación simple (sin recursos compartidos asociados), cambiar su tipo a englobante (*enclosing)*. Para ello solo hay que seleccionar el valor *enclosing* en la lista desplegable del campo *Operation type* del diálogo de la figura 3.21.

Los dos botones *Add Shared Resource* y *Remove Shared Resource* que aparecen en la ventana permiten modificar las listas de recursos compartidos de la operación.

El botón *Remove Shared Resource* permite eliminar el recurso seleccionado por el usuario de cualquiera de las listas de recursos compartidos.

Por su parte, el botón *Add Shared Resource* permite añadir un recurso compartido a la lista de recursos a bloquear o liberar de la operación simple, cuando se pulsa se muestra una nueva ventana como la de la figura 3.22, en ella aparece una lista con los recursos compartidos que previamente han sido añadidos al sistema en la Capa de Recursos Compartidos y una barra de herramientas con varios botones:

- **Añadir a Recursos a Bloquear** *(Add To Locked Resources)*: Al pulsarlo se añade el recurso de sistema seleccionado a la lista de recursos a bloquear de la operación simple.
- **Añadir a Recursos a Liberar** *(Add To Unlocked Resources)*: Al pulsarlo se añade el recurso de sistema seleccionado a la lista de recursos a liberar de la operación simple.
- **Añadir a Recursos a Bloquear y a Liberar** *(Add To Locked and Unlocked Resources)*: Al pulsarlo se añade el recurso de sistema seleccionado tanto a la lista de recursos a bloquear como a la lista de recursos a liberar de la operación simple.
- **Cancelar** *(Cancel)*: No se modifican las listas de recursos compartidos.

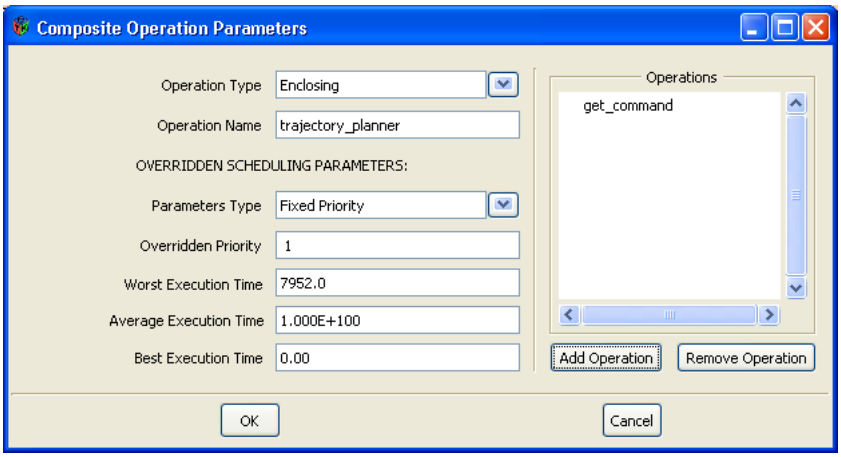

Figura 3.23: Diálogo de Propiedades de una Operación Compuesta

En la figura 3.24, “Diálogo para Añadir Operaciones a una Operación Compuesta”, se muestra otra de las ventanas auxiliares de esta capa. Esta ventana permite configurar

las propiedades de las operaciones compuestas. A continuación se explican los atributos correspondientes a cada clase (entre paréntesis y en cursiva aparece la terminología empleada en la ventana):

- Tipo *(Operation Type)*: Actualmente hay definidos los siguientes tipos:
  - Compuesta (*Composite*): Representa una operación compuesta de una secuencia ordenada de otras operaciones, simples o compuestas. No es necesario especificar el tiempo de ejecución ya que es la suma de los tiempos de ejecución de las operaciones que la componen. Sus atributos son:
    - Nombre *(Operation Name)*: Cadena de caracteres que identifica la operación.
    - Parámetros de planificación sobreescritos *(Overridden Scheduling Parameters)*: Representa un nivel de prioridad superior al nivel de prioridad normal al cual la operación se ejecutaría.
    - Lista de operaciones (*Operations*): Lista de operaciones que componen la operación.
    - Parámetros de planificación sobreescritos *(Overridden Scheduling Parameters)*: Representa un nivel de prioridad superior al nivel de prioridad normal al cual la operación se ejecutaría.
  - Englobante (*Enclosing*): Como la operación compuesta, representa una operación que incluye otras operaciones como parte de su ejecución, pero en este caso el tiempo de ejecución ha de ser especificado, ya que no es la suma de los tiempos de ejecución de las operaciones que la componen, esto se debe a que se pueden ejecutar otros trozos de código adicionalmente. Las operaciones englobadas necesitan ser consideradas con el fin de calcular los tiempos de bloqueo asociados al uso de sus recursos compartidos. Tiene los mismos atributos que la operación compuesta más los siguientes atributos adicionales:
    - Tiempos de ejecución (peor, medio y mejor) *(Execution Time (Worst, Average Best))*: Se expresa en unidades de tiempo normalizadas.

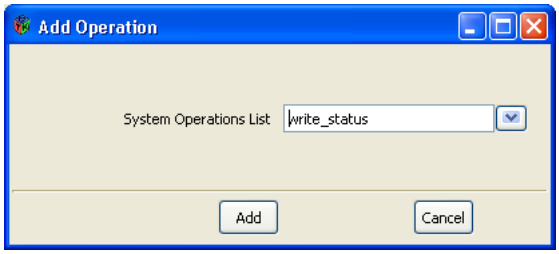

Figura 3.24: Diálogo para Añadir Operaciones a una Operación Compuesta

Los botones *OK* y *Cancel* tienen el mismo comportamiento que el comentado para los de la 3.21, “Diálogo de Propiedades de una Operación Simple”.

En la parte derecha de la ventana aparecen los botones *Add Operation* y *Remove Operation*, con los que se pueden añadir o borrar operaciones de la lista de operaciones.

El botón *Remove Operation* permite eliminar la operación seleccionada por el usuario de la lista de operaciones.

Al pulsar el botón *Add Operation* aparece una nueva ventana como la de la figura 3.24, en ella se muestra una lista con las operaciones (simples, compuestas y englobantes) que previamente han sido añadidas al sistema y una barra de herramientas con los botones:

- **Añadir** *(Add)*: Al pulsarlo se añade la operación del sistema seleccionada, a la lista de operaciones de la operación compuesta.
- **Cancelar** *(Cancel)*: Al pulsarlo no se modifica la lista de operaciones.

La ventana de la figura 3.25, "Diálogo de Propiedades de una Operación Transmisión de Mensaje" permite configurar todos los atributos de una operación de transmisión de mensajes, los cuales explicamos a continuación (entre paréntesis y en cursiva aparece la terminología empleada en la ventana):

- Tipo *(Operation Type)*: Indica el tipo de operación, en este caso solo hay definido un tipo: Message Transmission.
- Nombre *(Operation Name)*: Cadena de caracteres que identifica la operación.
- Parámetros de planificación sobreescritos *(Overridden Scheduling Parameters)*: Representa un nivel de prioridad superior al nivel de prioridad normal al cual la operación se ejecutaría
- Tamaños de paquete (máximo, medio y mínimo) *(Message Size (Max, Avg, Min))*: El tiempo de transmisión se calcula dividiendo el tamaño del paquete por el *Throughput* y el *Speed Factor* de la red correspondiente.

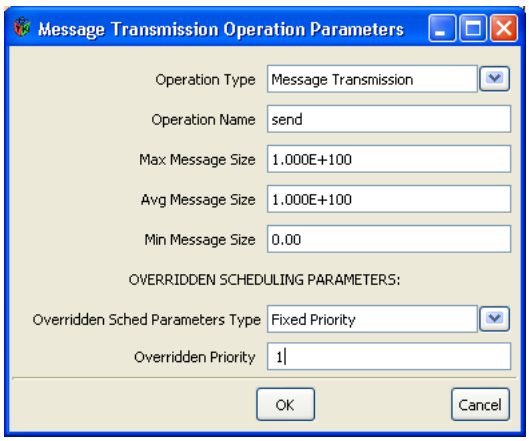

Figura 3.25: Diálogo de Propiedades de una Operación Transmisión de Mensaje

Los botones *OK* y *Cancel* tienen el mismo comportamiento que el comentado para los de la 3.21, "Diálogo de Propiedades de una Operación Simple".

3.3.6. Capa de Transacciones

En el apartado 1.4, "Modelo MAST para Sistemas de Tiempo Real", se comentó que un sistema de tiempo real se modela en MAST como un conjunto de **transacciones** *(transactions)*. Cada una de las transacciones se activa por uno o más **eventos externos** *(external events)* y representa un conjunto de acciones que serán ejecutadas en el sistema. Estas acciones generan **eventos internos** *(internal events)* que activan otras acciones dentro de una misma transacción.

La Capa de Transacciones se utiliza para configurar y representar gráficamente las distintas transacciones que contiene el sistema modelado. Como se puede observar en la

figura 3.26, la capa consta de dos elementos: una zona de dibujo en la que aparece el **Diagrama de las Transacciones** *(Transactions Diagram)* y una barra de herramientas.

En el diagrama, las transacciones se representan mediante cajas de color azul. Cada una de ellas tiene un grafo asociado que representa el flujo de **eventos** entre los **manejadores de eventos** *(event handlers)*, ambos elementos son representados como cajas dentro del grafo relacionados mediante flechas (ver figura 3.26). Atendiendo al modo de funcionamiento de los manejadores de eventos, podemos clasificarlos en dos grupos: **actividades** *(activities),* que representan la ejecución de una operación por un servidor de planificación, y **manejadores estructurales** *(structural handlers),* que sólo manipulan los eventos y no consumen recursos o tiempo de ejecución. Ambos tipos aparecen en el diagrama representados por cajas de color azul claro en las que se muestra el tipo específico de manejador (actividad, actividad temporizada de sistema, concentrador, barrera, etc.), con la diferencia que en el caso de las actividades también aparecen el nombre de la operación y del servidor de planificación asociados a la actividad.

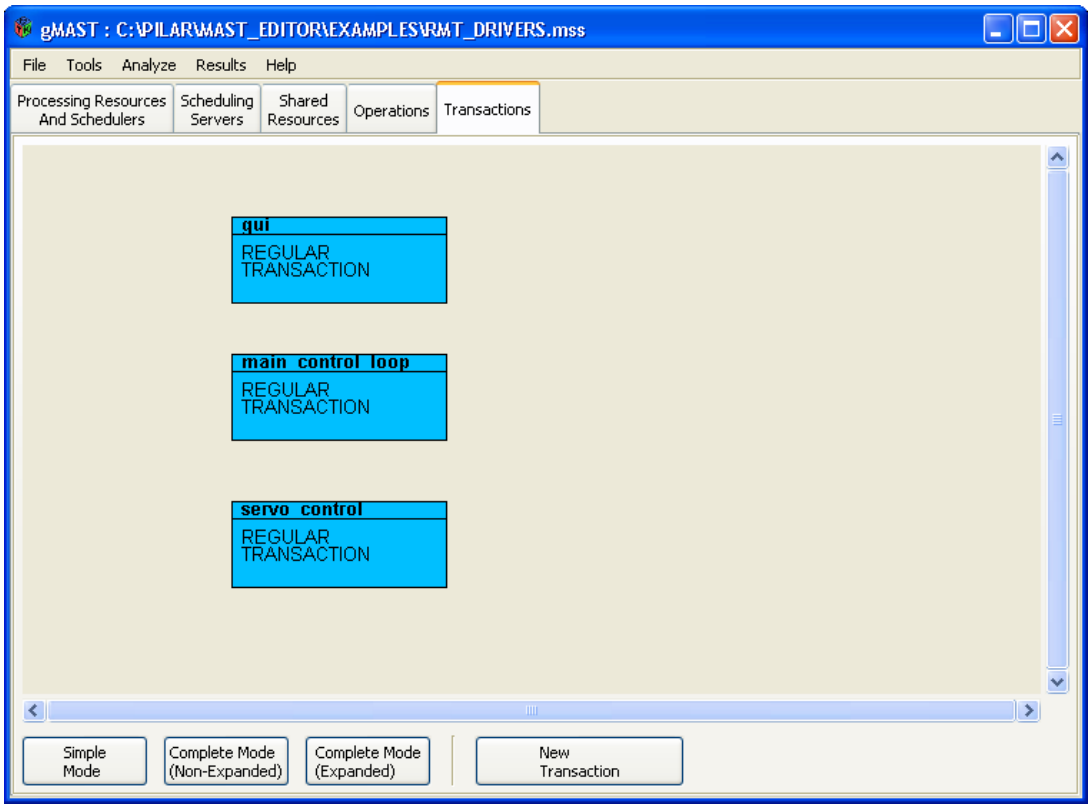

Figura 3.26: Capa de Transacciones

Análogamente a las otras capas, la **barra de herramientas** se dividide en dos partes: primeramente, están los botones *Simple Mode, Complete Mode (Non Expanded)* y *Complete Mode (Expanded)* con los que se determina el modo de visualización del diagrama y, por otro lado, está el botón para la creación de nuevas transacciones. El comportamiento de los tres primeros ya se explicó en detalle el apartado 3.3.2, "Capa de Recursos de Procesado y Planificadores". Por su parte, el botón **Nueva Transacción** *(New Transaction)* permite crear una nueva transacción. Al pulsarlo se muestra una ventana que contiene un diálogo con las propiedades de la transacción como se muestra en la figura 3.27.

Para eliminar, consultar o cambiar las propiedades de una transacción, el usuario puede utilizar el **menú emergente** cuyo comportamiento se explicó en el apartado 3.3.2. También se puede consultar sus propiedades colocando el puntero del ratón encima de la transacción y haciendo doble click con el botón izquierdo.

A continuación explicaremos detalladamente cada una de las ventanas auxiliares que aparecen en relación con esta capa. La primera de ellas es la de la figura 3.27, "Diálogo de Propiedades de una Transacción", que sirve para configurar los atributos de las transacciones que seguidamente se comentan (entre paréntesis y en cursiva aparece la terminología empleada en la ventana):

- Tipo *(Transaction Type)*: Sólo hay una clase de transacción definida llamada regular *(Regular)*.
- Nombre *(Transaction Name)*: Cadena de caracteres que sirve como identificador de la transacción.
- Diagrama *(Transaction´s Diagram)*: Diagrama donde se representa el grafo asociado a la transacción mostrando los manejadores y los eventos.

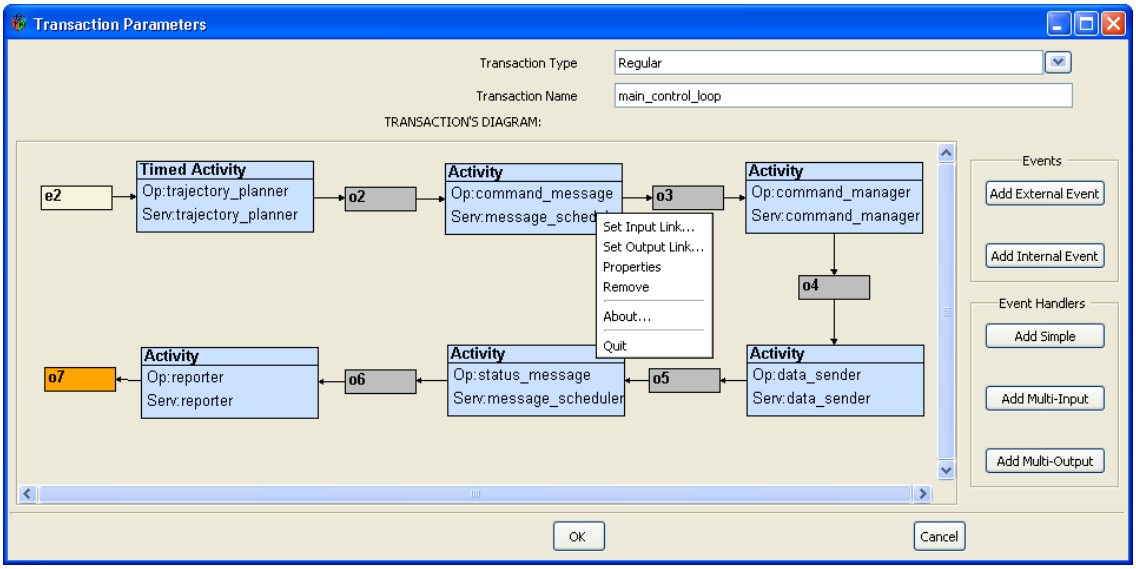

Figura 3.27: Diálogo de Propiedades de una Transacción

La función de los botones *OK* y *Cancel* que aparecen en la parte inferior es la de validar o no los cambios realizados en las propiedades de la transacción. Si la transacción ya existe, pulsando *OK* se actualizan sus propiedades, mientras que pulsando *Cancel* ésta permanece sin cambios, si no existe, pulsando *OK* se crea la transacción y se añade al sistema, mientras que si se pulsa *Cancel* no se realiza ningún cambio en el sistema.

Cuando se pulsa el botón *Add External Event* aparece una nueva ventana como la de la figura 3.28, en ella se pueden configurar los atributos de un evento externo. Los **eventos externos** modelan las interacciones del sistema con componentes o dispositivos externos a través de interrupciones, señales, etc., o con dispositivos de temporización hardware. Tienen un doble papel en el modelo: por una parte, establecen los patrones de llegada de las actividades al sistema, por otra parte, proporcionan referencias para definir requerimientos temporales globales. Se representan con cajas de color marfil. Sus propiedades son las siguientes (entre paréntesis aparece la terminología empleada en la ventana):

- Nombre *(Event Name)*: Cadena de caracteres que identifica el evento.

- Tipo *(External Event Type)*: Existen varios tipos de evento definidos para representar los diferentes patrones de llegada de las actividades:
  - Periódico (*Periodic*): Representa una cadena de eventos generados períodicamente. Sus atributos adicionales son:
    - Período *(Period)*: Período de tiempo en el que se producen las llegadas.
    - Máxima fluctuación *(Maximum Jitter)*: Indica la máxima cantidad de tiempo a añadir al tiempo de activación de cada evento.
    - Fase *(Phase)*: Instante de la primera activación en caso de que no tenga fluctuación (después de ese tiempo los siguientes eventos son períodicos, posiblemente con fluctuación).
  - Unitario (*Singular*): Representa un evento generado solo una vez. Sólo tiene un atributo adicional:
    - Fase *(Phase)*: Instante de la primera activación.
  - Esporádico (*Sporadic*): Representa una cadena de eventos aperiódicos que tienen un tiempo entre llegadas mínimo. Sus atributos adicionales son:
    - Tiempo entre llegadas mínimo *(Min Interarrival Time)*.
    - Tiempo entre llegadas promedio *(Average Interarrival Time)*.
    - Función de distribución *(Distribution Function)*: Función de distribución de los eventos aperiódicos. Puede ser uniforme *(Uniform)* o poisoniana *(Poisson)*.
  - No acotado (*Unbounded*): Representa una cadena de eventos para la cual no es posible establecer una cota superior del número de eventos que pueden llegar en un intervalo dado. Sus atributos adicionales son:
    - Tiempo entre llegadas promedio *(Average Interarrival Time)*
    - Función de distribución *(Distribution Function)*.

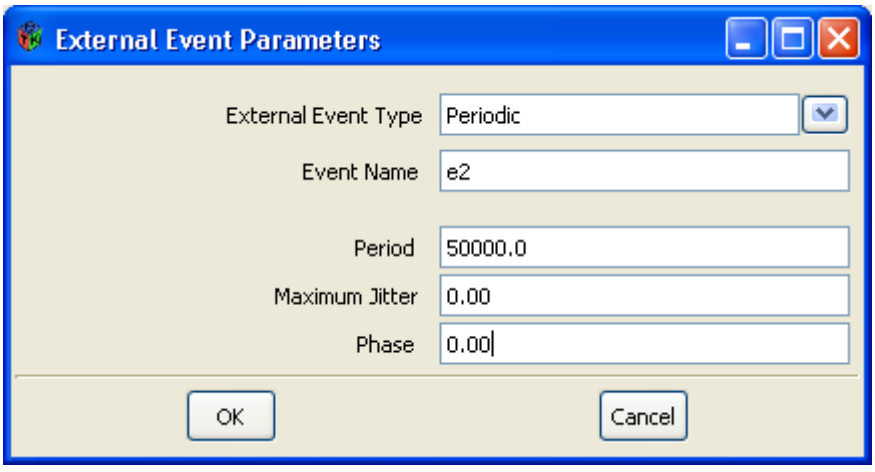

Figura 3.28: Diálogo de Propiedades de un Evento Externo

- A ráfagas (*Bursty*): Representa una cadena de eventos aperiódicos que tienen una cota superior del número de eventos que pueden llegar en un intervalo dado. Dentro de este intervalo, los eventos pueden llegar a ráfagas. Tiene los siguientes atributos adicionales:

- Intervalo de acotación *(Bound Interval)*: Intervalo de tiempo en el cual la cantidad de eventos está acotada.
- Máximo número de llegadas *(Max Arrivals)*: Indica el número máximo de eventos que pueden llegar en el intervalo de acotación.
- Tiempo entre llegadas promedio (*Average Interarrival Time)*.
- Función de distribución (*Distribution Function)*.

Si se pulsa el botón *OK* el evento se añade a la transacción y se dibuja en el diagrama, si por el contrario se pulsa *Cancel* entonces no se realiza ningún cambio.

Por otra parte, cuando se pulsa el botón *Add Internal Event* aparece una nueva ventana como la de la figura 3.29, en ella se pueden configurar las propiedades de un **evento interno** que es aquel generado por un manejador de eventos. Se representan con cajas de color gris (si no tienen requerimientos temporales) o naranja (si tienen requerimientos temporales). Sus propiedades son:

- Tipo *(Internal Event Type)*: Sólo hay una clase definida: regular *(Regular)*.
- Nombre *(Event Name)*: Cadena de caracteres que sirve como identificador del evento.

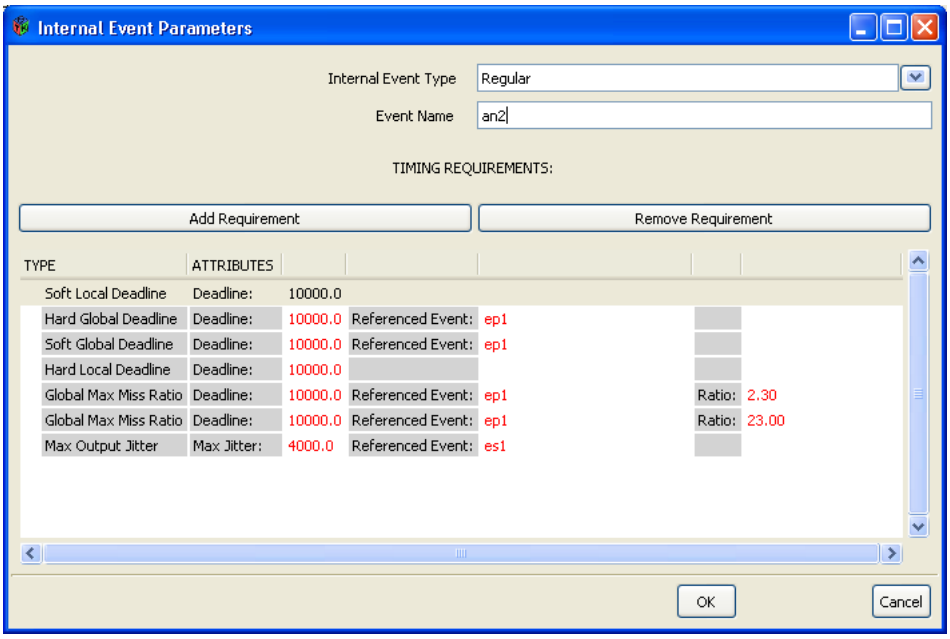

Figura 3.29: Diálogo de Propiedades de un Evento Interno

- Requerimientos temporales *(Timing Requirements Type)*: Los eventos internos pueden tener asociados requerimientos temporales impuestos en la generación del evento. Pueden ser de diferente tipo:
  - Plazo global estricto *(Hard Global Deadline)*: Representa el tiempo máximo permitido para la generación del evento asociado. En este caso, el plazo ha de cumplirse siempre y se referencia respecto a la llegada de un determinado evento. Sus atributos son:
    - Plazo *(Deadline)*: Se expresa en unidades de tiempo
    - Evento de referencia *(Referenced Event*): Nombre del evento al que se refiere.

- Plazo global no estricto *(Soft Global Deadline)*: Es similar al anterior sólo que en este caso el plazo se cumple en promedio. Tiene los mismos atributos que el plazo global estricto.
- Plazo local estricto *(Hard Local Deadline)*: El plazo se referencia respecto a la llegada del evento que activó la propia actividad. En este caso, el plazo se ha de cumplir siempre. Sólo tiene un atributo:
    - Plazo *(Deadline)*: Expresado en unidades de tiempo
- Plazo local no estricto (*Soft Local Deadline)*: Es análogo al anterior, salvo que en esta ocasión el plazo se cumple en promedio. Tiene un sólo atributo: el plazo *(Deadline)*.
- Fluctuación de salida máxima *(Max Output Jitter)*: Representa un requerimiento para limitar la fluctuación con la cual un evento interno es generado. Sus atributos son:
    - Fluctuación de salida máxima *(Max Output Jitter)*: Se calcula como la diferencia entre la respuesta temporal del peor y del mejor caso de la actividad que genera el evento asociado.
    - Evento de referencia *(Referenced Event*): Nombre del evento al que se refiere.
- Tasa de pérdida máxima global *(Global Max Miss Ratio)*: Representa un tipo de requerimiento no estricto en el cual el plazo no puede ser perdido más alla de una determinada tasa. En este caso, se referencia respecto a la llegada de un determinado evento. Sus atributos son:
    - Plazo *(Deadline)*.
    - Tasa *(Ratio)*: Porcentaje de pérdida de plazos máxima.
    - Evento de referencia *(Referenced Event*).
- Tasa de pérdida máxima local *(Local Max Miss Ratio)*: Es similar al anterior tipo, sólo que en este caso el plazo se referencia respecto a la llegada del evento que activó la actividad. Sus atributos son:
    - Plazo *(Deadline)*.
    - Tasa *(Ratio)*: Porcentaje de pérdida de plazos máxima.
- Compuestos *(Composite):* El evento tiene varios requerimientos impuestos al mismo tiempo, consta de una lista de requerimientos temporales simples.

Al pulsar el botón *Add Requirement* se muestra un diálogo auxiliar que nos permite elegir el tipo de requerimiento que queremos añadir a la lista (ver figura 3.30), si se pulsa *OK* en dicho diálogo se añade una nueva línea con los atributos correspondientes. Para editar los valores de los campos solo hay que hacer doble click sobre la celda asociada e introducir el valor. En el caso del atributo *Referenced Event*, al hacer doble click aparece una ventana auxiliar con la lista de eventos de la transacción, que permite elegir el evento deseado (ver figura 3.31). Al pulsar el botón *OK* en esa ventana se añade el nombre del evento referenciado en la fila del requerimiento temporal seleccionado inicialmente.

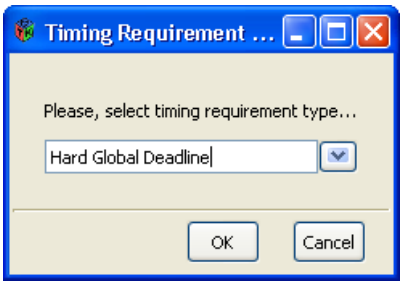

Figura 3.30: Diálogo de Selección de tipo de Requerimiento Temporal

Por otra parte, si pulsamos el botón *Remove Requirement* en el diálogo de propiedades del evento interno, se borra de la lista el requerimiento temporal que el usuario haya seleccionado en la pantalla.

Pulsando el botón *OK* del mismo diálogo el evento se añade a la transacción y se dibuja en el diagrama, si por el contrario se pulsa *Cancel* entonces se descartan los cambios.

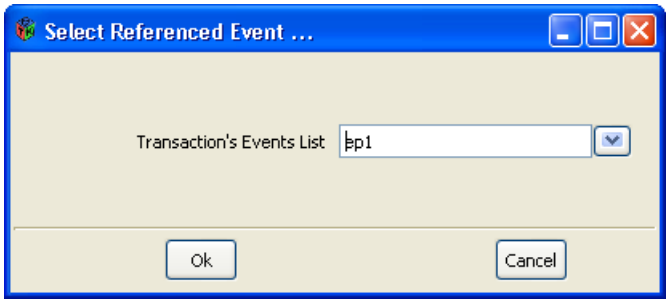

Figura 3.31: Diálogo auxiliar de la lista de eventos de una transacción

Finalmente comentaremos la función de los botones *Add Simple, Add Multi-Input y Add Multi-Output* de la figura 3.27, "Diálogo de Propiedades de una Transacción". Dichos botones sirven para añadir **manejadores de eventos** a la transacción. Los manejadores de eventos representan acciones activadas por la llegada de uno o más eventos, y que a su vez pueden generar uno o más eventos a su salida. Por consiguiente, atendiendo al número de eventos presentes a la entrada y a la salida, se pueden clasificar los manejadores en tres grupos:

- Manejador Simple *(Simple Event Handler)*: A la entrada sólamente llega un evento y genera un único evento a su salida.
- Manejador de Múltiples Entradas *(Multi-Input Event Handler)*: A la entrada llegan varios eventos y genera un único evento a su salida.
- Manejador de Múltiples Salidas *(Multi-Ouput Event Handler)*: A la entrada llega un sólo evento y genera varios eventos a su salida.

A partir de esta clasificación, se han diseñado tres nuevas ventanas de diálogo asociadas a los tres tipos de manejadores descritos anteriormente. Cada una de estas ventanas sirve para determinar las propiedades de los distintos tipos de manejadores y se activan al pulsar los botones *Add Simple, Add Multi-Input* y *Add Multi-Output* contenidos en la ventana de propiedades de la transacción.

Cuando se pulsa el botón *Add Simple* aparece una ventana como la de la figura 3.32, en ella se pueden configurar las propiedades de un **manejador de eventos simple**, cuyos atributos son los siguientes (entre paréntesis aparece la terminología empleada en la ventana de diálogo):

- Tipo *(Single Event Handler Type)*: Dentro de los manejadores simples existen varios tipos definidos:
  - Actividad *(Activity):* Representa una instancia de una operación para ser ejecutada por un servidor de planificación. Sus atributos adicionales son:
    - Operación de la actividad *(Activity Operation)*: Nombre de la operación a ejecutar.
    - Servidor de la actividad *(Activity Server)*: Nombre del servidor de planificación que ejecuta la operación.
  - Actividad temporizada de sistema *(System Timed Activity)*: Representa una actividad activada por el temporizador del sistema y por tanto sujeta a los *overheads* asociados con él. Tiene los mismos atributos que la anterior
  - Divisor de velocidad *(Rate Divisor)*: Genera un evento de salida cuando el número de eventos de llegada es igual al factor de división *(Rate Factor)*. Sólo tiene un atributo adicional:
    - Factor de división *(Rate Factor)*: Número de eventos que han de llegar para que se genere el evento de salida.
  - Retraso Local (*Delay)*: Genera su evento de salida después de que haya transcurrido un cierto intervalo de tiempo desde la llegada del evento de entrada. Sus atributos adicionales son:
    - Intervalo máximo de retraso *(Delay Max Interval)*: Intervalo de tiempo máximo usado para generar el evento de salida.
    - Intervalo mínimo de retardo *(Delay Min Interval)*: Intervalo de tiempo mínimo usado para generar el evento de salida.

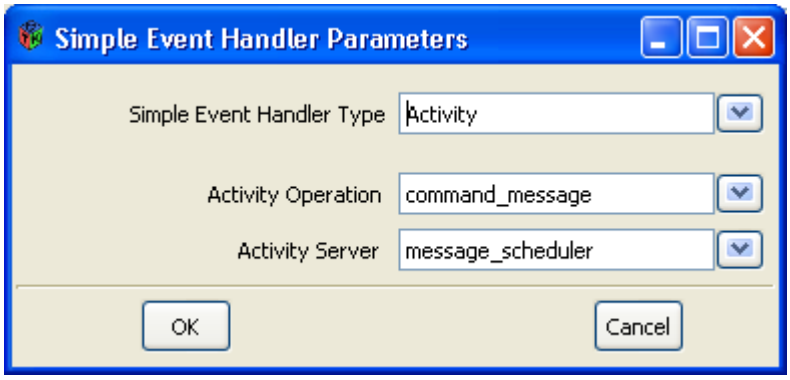

Figura 3.32: Diálogo de Propiedades de un Manejador de Eventos Simple

- Retraso Global *(Offset):* Es similar al anterior, excepto que el intervalo de tiempo es relativo a la llegada de algún evento previo. Si el intervalo de tiempo ya ha pasado cuando el evento de entrada llega, el evento de salida es generado inmediatamente. Sus atributos son los mismos del anterior incluyendo además:

- Evento de referencia *(Referenced Event)*: Nombre del evento tomado como referencia.

Al pulsar el botón *OK* de la figura 3.32 el manejador se añade a la transacción y se dibuja en el diagrama, si por el contrario se pulsa *Cancel* entonces no hay modificaciones.

El botón *Add Multi-Input* activa la ventana de la figura 3.33. En ella se muestran las propiedades de un **manejador de eventos de múltiples entradas**. Sus atributos se listan a continuación (entre paréntesis aparece la terminología empleada en la ventana):

- Tipo *(Input Event Handler Type)*: Existen varios tipos:
  - Concentrador *(Concentrator)*: Manejador que genera su evento de salida cuando llega cualquiera de sus eventos de entrada.
  - Barrera *(Barrier)*: Genera su evento de salida cuando todos sus eventos de entrada han llegado. Para poder realizar el análisis del peor caso es necesario que todos los eventos de llegada sean periódicos con el mismo período.

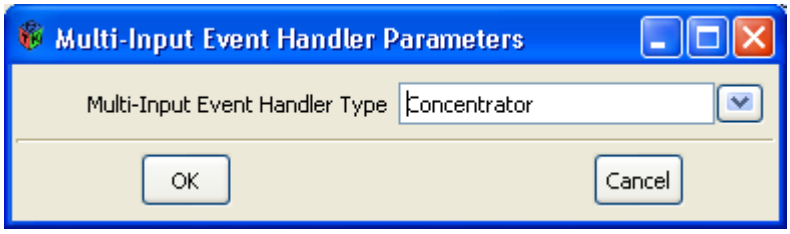

Figura 3.33: Diálogo de Propiedades de un Manejador de Eventos con varios eventos de entrada y un único evento de salida

Si se pulsa el botón *OK* de esa ventana se añade el manejador a la transacción y se dibuja en el grafo, mientras que pulsando *Cancel* se descartan los cambios.

De forma análoga a los dos casos anteriores, con el botón *Add Multi-Output* aparece una ventana como la de la figura 3.34, en ella se listan las propiedades de un **manejador de eventos de múltiples salidas**. Este tipo de manejadores tiene los siguientes atributos (entre paréntesis se muestra la terminología empleada en la ventana):

- Tipo *(Output Event Handler Type)*: Existen varios tipos:
  - Servidor de reparto *(Delivery Server):* Es un manejador que genera un evento en solo una de sus salidas cada vez que llega un evento de entrada. La salida se determina en el momento de generación del evento. Sólo tiene un atributo adicional:
    - Política de reparto *(Delivery Policy)*: Política utilizada para determinar el camino de salida. Puede ser de escaneo *(Scan)* (la salida se elije de forma cíclica) o aleatoria *(Random)*.
  - Servidor de petición *(Query Server):* Este manejador genera un evento en solo una de sus salidas cada vez que llega un evento de entrada. La salida se elige en el momento en que el evento se despacha por una de las actividades conectadas a un evento de salida. Tiene un atributo adicional:
    - Política de petición *(Request Policy)*: Política que determina el camino de salida cuando hay varias peticiones pendientes desde las actividades conectadas. Puede ser de escaneo *(Scan)* (la salida se

elije de forma cíclica), de prioridad *(Priority)* (la actividad de prioridad más alta gana), FIFO o LIFO.

- Multiplicador *(Multicast):* Manejador que genera un evento en cada una de sus salidas cada vez que llega un evento.

Dentro de este diálogo aparecen los botones *OK* y *Cancel*. El primero de ellos sirve para añadir el manejador y dibujarle en el diagrama de la transacción, mientras que pulsando el segundo no se realiza ninguna acción.

Para el caso de los manejadores de eventos el menú emergente presenta opciones adicionales: *Set Input Link...* y *Set Output Link...*(figura 3.27) para manejadores simples, *Add Input Link...* y *Set Output Link...* para manejadores de múltiples entradas y *Set Input Link...* y *Add Output Link...* para manejadores de múltiples entradas.

Figura 3.34: Diálogo de Propiedades de un Manejador de Eventos con un único evento de entrada y varios eventos de salida

Las opciones *Set Input Link...*y *Add Input Link...* sirven para configurar los enlaces de entrada a los manejadores mostrando una ventana que incluye la lista de eventos de la transacción sobre la que elegir el evento deseado. Al pulsar el botón *OK* de esa ventana se establece la relación manejador-evento y, dentro del grafo, se dibuja una flecha desde el evento al manejador.

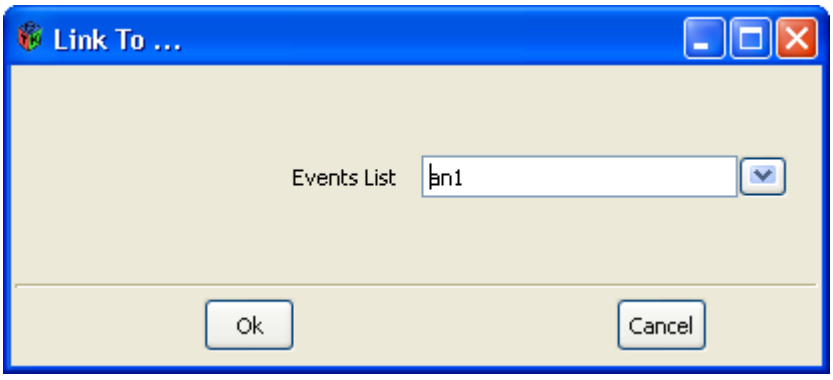

Figura 3.35: Diálogo auxiliar de configuración de enlace de salida motrando la lista de eventos de una transacción

Las opciones *Set Output Link...*y *Add Output Link...* tienen un cometido análogo a la anteriores solo que en este caso establecen enlaces de salida de los manejadores. También muestran una ventana con la lista de eventos de la transacción de forma similar a la vista anteriormente, en la que al pulsar *OK* se establece el enlace de salida y se dibuja una flecha desde el manejador al evento.

3.4. Operaciones con ficheros

Previamente a la utilización de las herramientas de análisis incluidas en MAST es necesario generar la descripción MAST del sistema modelado, en la cual se define la estructura y disposición de todos los elementos que componen el sistema (recursos de procesado, servidores de planificación, operaciones, etc.) y se determinan las relaciones existentes entre ellos. La descripción MAST se especifica en formato ASCII y para confeccionarla es preciso seguir unos patrones o plantillas definidos de antemano por los programadores del software y que sirven como pautas para la correcta especificación de la misma en formato de texto. Una vez elaborada la descripción ASCII, ésta es convertida por un parser en una estructura de datos Ada que se utiliza como entrada de las herramientas de análisis. Por este motivo resulta fundamental que la descripción se confeccione siguiendo el formato establecido y así evitar que se produzcan errores en la invocación de las herramientas.

Una de las principales funcionalidades del editor gráfico es la de generar de forma automática la descripción del sistema en formato ASCII a partir de los parámetros configurados por el usuario en las capas y diálogos que componen la interfaz. Para ello se ha diseñado una herramienta que además de generar un archivo de texto con la descripción ASCII del sistema, también genera de forma paralela otro archivo de texto en el cual se almacenan los parámetros referentes a la representación gráfica del modelo en las diferentes capas y diagramas del editor, esto es, el tipo, subtipo, nombre y coordenadas. En el apartado 4.3.6, “Ficheros del Editor Gráfico” se explica detalladamente cómo se almacenan dichos parámetros y el formato de archivo utilizado.

En la parte superior de la ventana principal se incluye un menú desplegable llamado **File** con diferentes opciones para la lectura o escritura simultánea de ambos archivos (ver figura 3.2):

- ***New***: Con esta opción se crea un nuevo modelo y se borra el contenido de todas las capas de la interfaz, dando al usuario la posibilidad de salvar el modelo editado anteriormente.

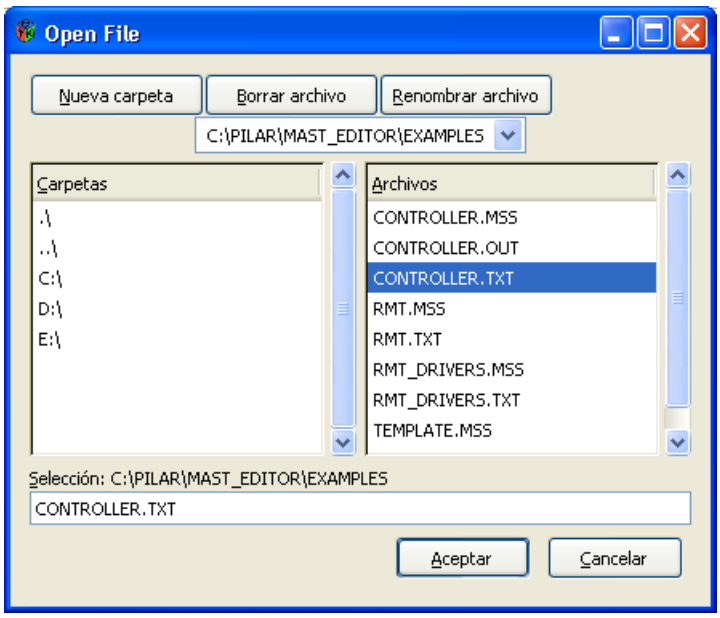

Figura 3.36: Ventana de apertura de archivo

- ***Open...***: Esta opción permite abrir simultáneamente los dos archivos asociados a un modelo ya existente. El usuario sólo ha de seleccionar el nombre del archivo que contiene la descripción ASCII en la ventana de la figura 3.36 para que se cargen los atributos y diagramas correspondientes al modelo. El otro archivo se abre da manera automática ya que su nombre coincide con el de la descripción MAST, aunque tiene diferente extensión.

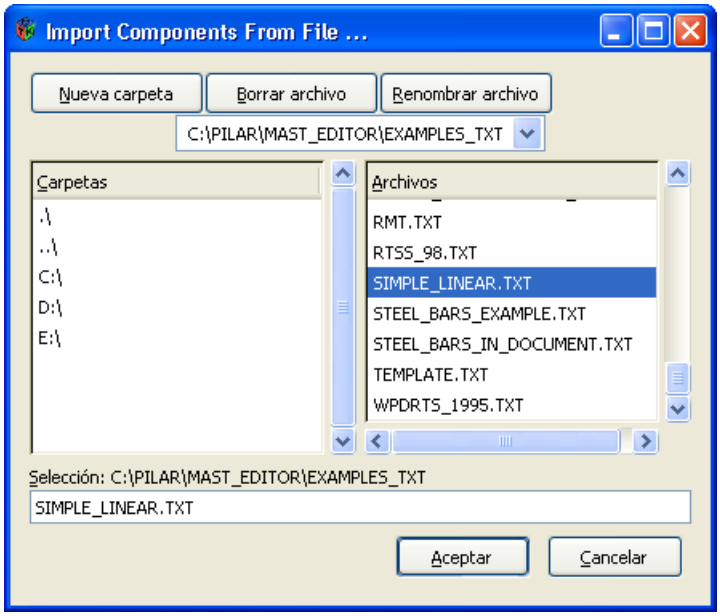

Figura 3.37: Ventana de importación de componentes desde archivo

- ***Import...***: Esta opción permite importar componentes de otro modelo al sistema editado. Para ello el usuario sólo ha de seleccionar el nombre del archivo que contiene la descripción ASCII con los componentes que quiere añadir al sistema (figura 3.37).

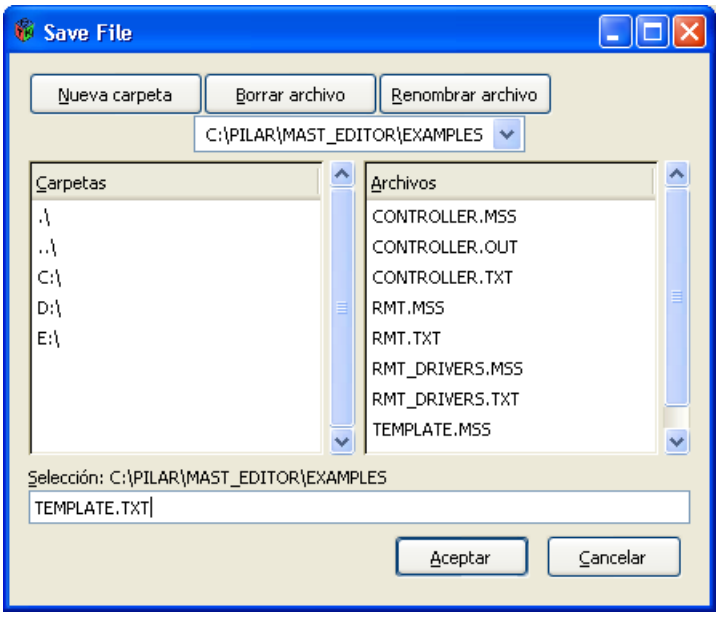

Figura 3.38: Ventana de guardado de archivo

- ***Save***: Permite generar la descripción ASCII del modelo de forma automática. Guarda en un archivo de texto la descripción ASCII y simultáneamente salva en otro archivo de texto los parámetros referentes a la representación gráfica del sistema.
- ***Save As...***: Es similar a la anterior opción, sólo que lleva asociada una ventana para la selección del nombre con el que se quiere salvar los archivos (ver figura 3.38).

Las ventanas de diálogo de selección de archivo permiten navegar por la estructura de directorios y escoger el nombre del archivo de origen que contiene la descripción MAST. Si se elige un archivo que no se corresponde con el formato adecuado o falta el archivo de descripción ASCII, se visualizará un mensaje de error indicando la incidencia. En el caso de que sólo se disponga del archivo de descripción ASCII, se podrá visualizar la representación del modelo MAST en los diferentes diagramas del editor sin ningún problema.

3.5. Herramientas *(Tools)*

Este menú integrado en la interfaz gráfica contiene un solo elemento llamado *Create Simple Transaction...* que nos permite lanzar un asistente para la creación de una transacción simple.

3.5.1. Asistente para la creación de transacciones simples

Este asistente se lanza pulsando sobre la opción *Create Simple Transaction...* del menú *Tools* y consta de varias pantallas de configuración que guían al usuario en el proceso de creación de la transacción simple.

Una transacción simple es aquella que se compone de un evento de entrada, una actividad y un evento de salida.

El proceso de creación de la transacción se puede cancelar en cualquier momento pulsando cualquiera de los botones *Cancel* de las pantallas del asistente. En este caso el sistema permanecerá sin cambios.

Los botones *Next* y *Back* de las ventanas permiten al usuario avanzar o retroceder en el proceso de creación. Esto permite redefinir o cambiar los parámetros de la transacción todas las veces que se precise, antes de añadirla al sistema.

El propio asistente se encarga de verificar la validez de los parámetros introducidos por el usuario en cada pantalla y, en el caso de que no sean correctos, muestra en pantalla el correspondiente mensaje de error.

3.5.1.1. Ventana de Bienvenida

Al lanzarse el asistente se muestra una ventana inicial que da la bienvenida al usuario en el proceso de creación de la transacción simple (figura 3.39).

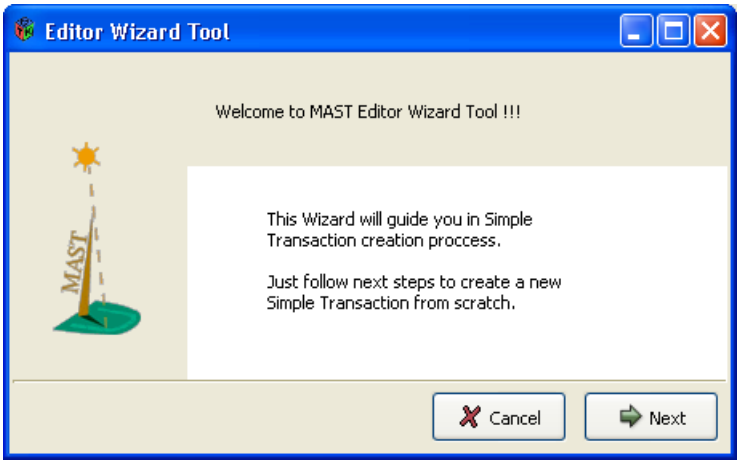

Figura 3.39: Ventana de bienvenida del asistente

3.5.1.2. Primer Paso: Especificación del nombre de la transacción

Pulsando el botón *Next* de la ventana de bienvenida aparece una nueva ventana que permite especificar el nombre de la nueva transacción simple que se quiere añadir al sistema (figura 3.40).

Si se pulsa el botón *Next* de esta pantalla se seguirá con el proceso, si se pulsa el botón *Back* se volverá a la pantalla de bienvenida.

El botón *Cancel* cancela todo el proceso, si bien antes se muestra una pantalla que pide al usuario que confirme la salida del asistente.

Figura 3.40: Primer Paso: Especificación del nombre de la transacción

3.5.1.3. Segundo Paso: Configuración del Evento de Entrada

Pulsando el botón *Next* de la anterior ventana se muestra una nueva ventana (figura 3.41) que permite configurar los parámetros del evento externo que compone el evento de entrada de la transacción simple. Los atributos comunes a especificar son: tipo *(type)* y nombre *(event name)* y, en función del tipo de evento externo elegido, se mostrarán los correspondientes parámetros. Estos atributos ya fueron explicados en el apartado 3.3.6, “Capa de Transacciones”.

Si se pulsa el botón *Next* de esta pantalla se seguirá con el proceso, si se pulsa el botón *Back* se volverá a la pantalla anterior.

Figura 3.41: Segundo Paso: Configuración del Evento de Entrada

3.5.1.4. Tercer Paso: Configuración de la Actividad

Pulsando el botón *Next* de la anterior ventana se muestra una nueva ventana (figura 3.42) que permite configurar los parámetros de la actividad de la transacción simple.

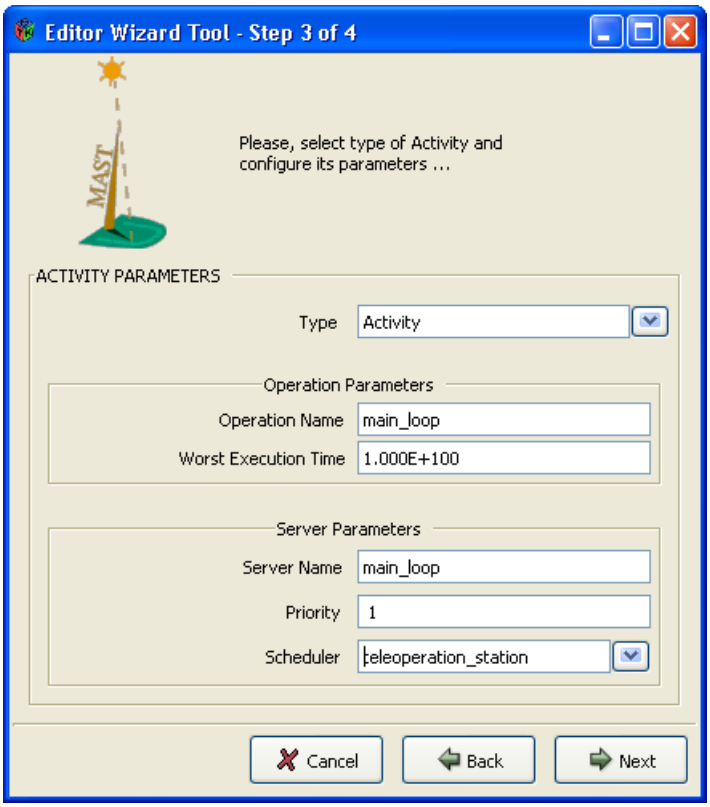

Figura 3.42: Tercer Paso: Configuración de la Actividad

Los atributos a especificar son: tipo *(type)*, nombre de la operación *(operation name)*, peor tiempo de ejecución *(worst execution time),* nombre del servidor de planificación *(server name),* prioridad *(priority)* y planificador *(scheduler)*.

3.5.1.5. Cuarto Paso: Configuración del Evento de Salida

Pulsando el botón *Next* de la anterior ventana aparece una nueva ventana (figura 3.43) que permite configurar los parámetros los parámetros del evento interno que compone el evento de salida de la transacción simple. Los atributos a especificar son: nombre *(event name)* y plazo *(deadline)*.

Si se pulsa el botón *Next* de esta pantalla se seguirá con el proceso, si se pulsa el botón *Back* se volverá a la pantalla anterior.

El botón *Cancel* cancela todo el proceso, si bien antes se muestra una pantalla que pide al usuario que confirme la salida del asistente. Si se confirma dicha acción el sistema permanecerá sin cambios.

Figura 3.43: Cuarto Paso: Configuración del Evento de Salida

3.5.1.6. Finalización del asistente

Una vez realizados todos los pasos anteriores, si se pulsa el botón *Next* de la anterior ventana aparece una nueva ventana (figura 3.44) que notifica al usuario de que ya se dispone de la información necesaria para crear la transacción.

Si se pulsa el botón *Apply* el asistente creará la transacción, el servidor y la operación especificados en las anteriores pantallas y los añadirá al sistema. Si se pulsa el botón *Back* se volverá a la pantalla anterior.

El botón *Cancel* cancela todo el proceso, si bien antes se muestra una pantalla que pide al usuario que confirme la salida del asistente. Si se confirma dicha acción el sistema permanecerá sin cambios.

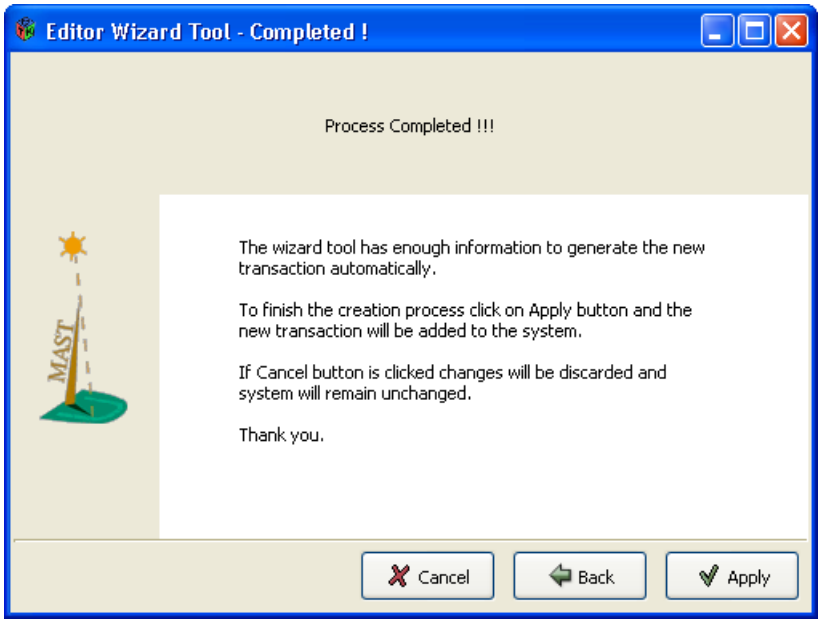

Figura 3.44: Ventana de finalización del asistente

3.6. Cómo realizar el modelado del sistema

En este apartado se describe cuál es la secuencia lógica de pasos a seguir para modelar un sistema de tiempo real con el editor gráfico de MAST en 7 puntos, que son:

1. Configurar los parámetros asociados a los recursos de procesado y los planificadores primarios: Se han de asignar los valores de los atributos de los procesadores (incluyendo los temporizadores en aquellos procesadores que lleven asociado uno), de las redes (incluyendo los controladores de red y sus correspondientes servidores y operaciones) y de los planificadores primarios asociados (especificando la política de planificación).
2. Determinar las propiedades de los servidores de planificación y los planificadores secundarios: Es preciso definir la política de planificación asignada a cada planificador secundario y los parámetros de planificación del servidor asociado.
3. Especificar los recursos compartidos del sistema: Para ello se ha de determinar el tipo y nombre de cada recurso.
4. Configurar las operaciones: Primeramente se especifican las operaciones simples (incluyendo los recursos compartidos que éstas utilizan), después las operaciones compuestas (determinando las operaciones que incluyen) y las de transmisión de mensajes.
5. Configurar las transacciones y sus componentes: Definiendo los eventos (externos e internos) y manejadores incluídos en cada transacción y configurando las relaciones existentes entre ellos.
6. Generar el archivo con la descripción ASCII del sistema: Utilizando el menú habilitado para tal fin dentro de la ventana principal del editor.

Es preciso realizar los pasos anteriores de forma cuidadosa y precisa, ya que un fallo en cualquiera de ellos puede dar lugar a errores en la simulación y análisis del sistema,

generando resultados que pueden distar mucho de la realidad de los sistemas que se modelan.

No siempre es necesario seguir todos los pasos. Puede que en ciertas situaciones no se requiera realizar algunos de ellos, como en el caso en que se disponga de un modelo elaborado previamente cuya configuración ha sido guardada en un archivo y que puede servir como referente en la creación de un nuevo modelo.

4. Implementación del Editor

4.1. Introducción

Una vez definidas la funcionalidad, las facilidades y opciones del programa en el capítulo anterior, en este capítulo se realiza una descripción de la implementación de la interfaz, explicando la estructura de programación y el proceso seguido en la elaboración de los archivos y paquetes incluidos en ella.

En todo momento se ha tenido en cuenta, a la hora de la programación, que el objetivo final era crear una interfaz de usuario portable para plataformas Windows y Linux y que además fuese fácilmente extensible en el futuro. Este último es uno de los principales motivos por los que se optó por el lenguaje Ada 95, ya que es un lenguaje de programación orientada al objeto que presenta facilidades como la abstracción de tipos, la utilización de paquetes genéricos y el tratamiento de errores con excepciones, entre otras, que permiten la programación de software fiable, reutilizable y extensible. Por otra parte, para lograr que la interfaz fuera portable se utilizaron la herramientas GtkAda [20] y las librerías gtk+, ya que permiten crear interfaces gráficas de usuario para múltiples entornos, incluidos Windows y Linux.

El diseño e implementación de este editor gráfico para MAST, se puede dividir en tres partes como se muestra en la figura 4.1. Por un lado, se realizó el diseño y programación de los paquetes que se corresponden con las ventanas de la interfaz y cuyo código está relacionado con la composición y el comportamiento de las mismas. Estos paquetes se generaron automáticamente mediante un constructor de interfaces de usuario llamado Glade integrado en el entorno Gtk/GtkAda. Por otro lado, se diseñaron y programaron los paquetes asociados a los objetos manejados internamente por el editor siguiendo un esquema paralelo al de los paquetes de la librería MAST existentes. Finalmente, se elaboró el código de los paquetes relacionados con la creación, lectura y escritura de archivos y con el control de cambios del archivo editado. Los detalles de la implementación de todos los paquetes se comentarán dentro del apartado 4.3.

La descripción de la programación se limitará a proporcionar una visión global de la estructura del código sobre el que se apoya el programa. Se evitará entrar a comentar el código en profundidad ya que, además de que esta acción tomaría una extensión no aconsejable en la redacción de un Trabajo de Investigación, en ciertos casos aportaría gran redundancia, debido a que muchos de los módulos comparten un perfil común al haber utilizado técnicas de programación orientada a objetos.

También se comentará el formato de los archivos que guardan los parámetros y propiedades correspondientes a la representación gráfica de los objetos en las diferentes pantallas y diagramas del editor. Dichos archivos son creados y manejados internamente por el editor gráfico que los utiliza para mostar los diagramas del modelo en pantalla y por esta razón es importante que la lectura y escritura en dichos archivos se haga de forma

cuidadosa. El último apartado del capítulo se dedica a explicar en detalle la forma de acceso a estos archivos, así como su formato y extensión.

4.2. Arquitectura del software de la interfaz

La arquitectura de la aplicación se muestra en la figura 4.1. Como se puede observar el software consta de varios paquetes Ada que se pueden agrupar en tres tipos:

- **Paquetes Glade:** Este tipo de paquetes han sido creados utilizando la herramienta Glade que es un constructor de interfaces para gtk+ (ver apartado 4.3). Su código determina la estructura y comportamiento de las ventanas que componen la interfaz y las relaciones que existen entre ellas.
- **Paquetes Mast_Editor:** Estos paquetes contienen el código utilizado para mostrar o recoger los valores de los atributos del modelo de las ventanas y para dibujar los elementos en los diagramas del editor.
- **Paquetes de Manejo de Archivos y paquetes de Control:** La principal función de estos paquetes es la de controlar la creación y el acceso de los archivos manejados por el editor, realizar el control de cambios del archivo editado y de los asistentes del menú de herramientas.

La mayoría de estos paquetes están relacionados entre sí ya que, en muchos casos, dentro del cuerpo de los procedimientos o funciones de un paquete se realizan llamadas a funciones y procedimientos de otros paquetes. Las flechas que aparecen en el diagrama de la figura 4.1 indican las relaciones entre los distintos tipos de paquetes de la interfaz, incluyendo también los paquetes de la librería MAST. Se puede comprobar que los módulos internos hacen uso de los procedimientos y funciones y de algunas variables de otros módulos internos de forma bidireccional. Además, los *Paquetes Mast_Editor* y los *Paquetes de Manejo de Archivos* hacen llamadas a procedimientos y funciones de paquetes externos a la interfaz e incluso instanciaciones a paquetes genéricos incluidos dentro de la librería MAST, por eso se representa la relación unidireccionalmente ya que los módulos de dicha librería son totalmente independientes de los de la interfaz.

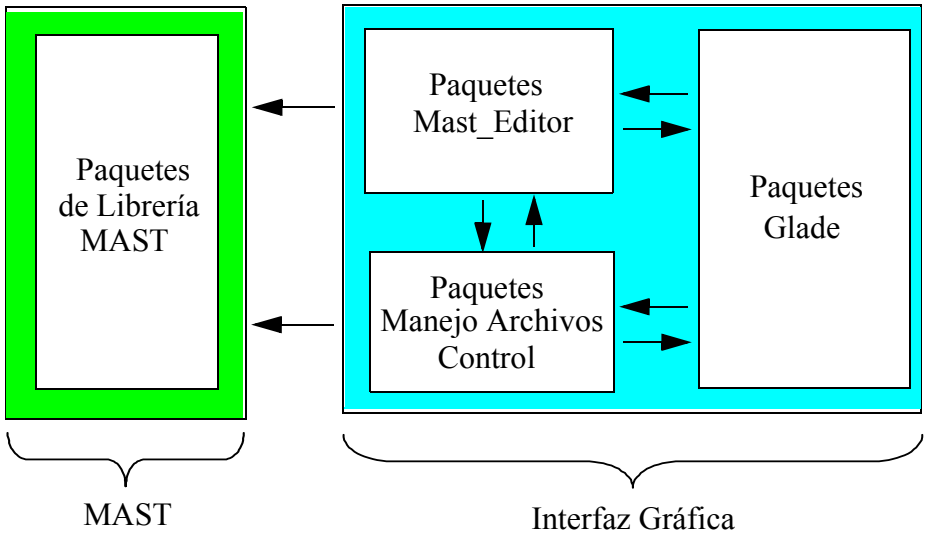

Figura 4.1: Arquitectura y dependencias de paquetes de la Interfaz

La estructura y especificación de los diferentes módulos incluídos en la interfaz, así como de los archivos asociados a los mismos será descrita en el apartado 4.3, “Detalles de la implementación”.

4.3. Detalles de la implementación

En los sucesivos apartados se describirán las distintas facetas de la implementación de la interfaz gráfica. Se explicará la estructura de datos empleada y la arquitectura de los paquetes, atendiendo a la clasificación mostrada en la figura 4.1. Primero se comentarán en detalle los paquetes generados por las herramientas GtkAda, posteriormente se explicará la jerarquía de objetos y paquetes creada para la representación gráfica del modelo MAST y, finalmente, se describirá cómo se realiza el manejo de archivos.

Para referencia del lenguaje Ada y de la programación orientada a objetos recomendamos “Ada 95: The Craft of Object-Oriented Programming” [3], así como [4] y [17].

4.3.1. Paquetes Glade

Para elaborar el código correspondiente a las ventanas, diálogos y demás componentes gráficos de la interfaz se utilizó el software GtkAda [19][20], un conjunto de herramientas gráficas de alto nivel, basado en las librerías gtk+ [16], que permite crear interfaces de usuario portables para múltiples plataformas. Una de las razones por las que se optó por las herramientas GtkAda es que llevan integrado el constructor de interfaces gráficas de usuario Glade. Este constructor facilita la labor de programación de los paquetes Ada asociados a los componentes gráficos de la interfaz, ya que una vez diseñados cada uno de los diálogos, ventanas, menús, etc., genera el código Ada de forma automática. De esta manera, sólo es necesario modificar los procedimientos o funciones asociados a aquellos componentes en los que el código generado por defecto no es válido para obtener el comportamiento deseado.

Dentro de la aplicación Glade cada interfaz creada se guarda en un archivo XML llamado archivo de proyecto (*project file*) en el cual se definen, entre otras cosas, todos los componentes de alto nivel (*high-level widgets*) que constituyen la interfaz (ventanas, menús, etc.). Una vez diseñada la interfaz, las herramientas GtkAda utilizan el archivo de proyecto como argumento de entrada para generar los archivos Ada correspondientes.

En nuestro caso, primero creamos un nuevo proyecto en Glade llamado “*gmasteditor*” y en él diseñamos los diálogos, ventanas y menús del editor gráfico. En la figura 4.2 se muestra la ventana con los componentes de alto nivel incluidos en el proyecto. Seguidamente se comenta la función asociada a cada uno de ellos:

- **Mast_Editor_Window:** Diseño de la ventana principal de la aplicación.
- **Open_File_Selection:** Diseño del cuadro de diálogo para la selección de un archivo que el usuario desea abrir.
- **Save_File_Selection:** Es similar al anterior y se corresponde con el diseño del cuadro de diálogo para la selección de un archivo que el usuario desea guardar.
- **Processor_Dialog:** Diseño del diálogo que permite configurar las propiedades de los procesadores del modelo MAST.

Figura 4.2: Componentes de alto nivel del proyecto Glade

- **Network_Dialog:** Diseño del diálogo que permite configurar las propiedades de las redes del modelo MAST.
- **Driver_Dialog:** Diseño del diálogo auxiliar que permite configurar las propiedades de los controladores de una red.
- **Add_Operation_Dialog:** Diseño del diálogo auxiliar que permite definir las operaciones incluídas en la operación compuesta.

- **Add_Shared_Dialog:** Diseño del diálogo auxiliar que permite definir los recursos compartidos de una operación simple.
- **Item_Menu:** Corresponde al diseño del menú emergente mostrado cuando el usuario pulsa un botón del ratón en la ventana principal.
- **Add_New_Op_To_Driver_Dialog:** Diseño del diálogo auxiliar que permite añadir una operación a un controlador de red.
- **Add_New_Server_To_Driver_Dialog:** Diseño del diálogo auxiliar que permite añadir un servidor de planificación a un controlador de red.
- **Sop_Dialog:** Diseño del diálogo que permite configurar las propiedades de las operaciones simples.
- **Sched_Server_Dialog:** Diseño del diálogo que permite configurar las propiedades de los servidores de planificación del modelo.
- **Editor_Error_Window:** Diseño de la ventana utilizada para indicar que se ha producido un error en la aplicación.
- **Prime_Sched_Dialog:** Diseño del diálogo que permite configurar las propiedades de los planificadores primarios.
- **Import_File_Selection:** Diseño del cuadro de diálogo para la selección de un archivo desde donde se importan nuevos elementos del modelo.
- **External_Dialog:** Diseño del diálogo que permite configurar las propiedades de los eventos externos.
- **Internal_Dialog:** Diseño del diálogo que permite configurar las propiedades de los eventos internos.
- **Trans_Dialog:** Diseño del diálogo que permite configurar las propiedades de las transacciones.
- **SEH_Dialog:** Diseño del diálogo que permite configurar las propiedades de los manejadores de eventos simples.
- **MIEH_Dialog:** Diseño del diálogo que permite configurar las propiedades de los manejadores de eventos de múltiples entradas.
- **MOEH_Dialog:** Diseño del diálogo que permite configurar las propiedades de los manejadores de eventos de múltiples salidas.
- **Add_Link_Dialog:** Diseño del diálogo que permite configurar los enlaces entre los eventos y manejadores de las transacciones.
- **Save_Changes_Dialog:** Diseño del diálogo de confirmación del salvado de los cambios de archivo.
- **Select_Req_Type_Dialog:** Diseño del diálogo que permite seleccionar el tipo de requerimiento temporal.
- **Shared_Resource_Dialog:** Corresponde al diseño del diálogo que permite configurar las propiedades de los recursos compartidos.
- **Aux_Window:** Diseño de la ventana en la que se visualizan los diagramas auxiliares.
- **Select_Ref_Event_Dialog:** Diseño del diálogo que permite seleccionar el nombre del evento de referencia de un requerimiento temporal.
- **Wizard_Welcome_Dialog:** Corresponde al diseño del diálogo de bienvenida del asistente de creación de transacciones.

- **Wizard_Transaction_Dialog:** Corresponde al diseño del diálogo que permite especificar el nombre de la transacción simple en el asistente de creación de transacciones simples.
- **Wizard_Input_Dialog:** Diseño del diálogo que permite configurar el evento de entrada de la transacción simple en el asistente de creación de transacciones simples.
- **Wizard_Activity_Dialog:** Diseño del diálogo que permite configurar la actividad de la transacción simple en el asistente de creación de transacciones simples.
- **Wizard_Output_Dialog:** Diseño del diálogo que permite configurar el evento de salida de la transacción simple en el asistente de creación de transacciones simples.
- **Wizard_Completed_Dialog:** Diseño del diálogo de finalización del asistente de creación de transacciones.
- **Cop_Dialog:** Diseño del diálogo que permite configurar las propiedades de las operaciones compuestas.
- **Message_Tx_Dialog:** Diseño del diálogo que permite configurar las propiedades de las operaciones de transmisiones de mensajes.
- **Cop_Dialog:** Diseño del diálogo que permite configurar las propiedades de las operaciones compuestas.
- **Second_Sched_Dialog:** Diseño del diálogo que permite configurar las propiedades de los planificadores secundarios.
- **Timer_Dialog:** Diseño del diálogo auxiliar que permite configurar las propiedades de los temporizadores.
- **Log_Window:** Diseño del diálogo auxiliar que muestra las incidencias acontecidas durante la ejecución de la aplicación.

Después de diseñar los componentes de alto nivel generamos el código Ada por medio de la opción *Construir*. Esto da como resultado una serie de archivos Ada cuya estructura explicaremos a continuación.

Por cada uno de los componentes de alto nivel que aparecen en la figura 4.2 se generan cuatro archivos (el nombre de los archivos viene determinado por el nombre con el que se designa al componente dentro del proyecto Glade):

- ***<nombre_componente>_pkg.ads***
- ***<nombre_componente>_pkg.adb***
- ***<nombre_componente>_pkg-callbacks.ads***
- ***<nombre_componente>_pkg-callbacks.adb***

Los dos primeros archivos se correspoden a los paquetes que contienen las llamadas GtkAda necesarias para crear e inicializar un componente. Dentro del cuerpo del paquete (en el archivo *<nombre_componente>_pkg.adb*) se especifica el tipo de componente (ventana, diálogo, menú, etc.), se determina su estructura y los elementos que lo componen (tablas, cajas, botones, etiquetas, etc.) y se definen las señales emitidas/ conectadas al componente. Seguidamente se comentará el funcionamiento de estas señales.

Antes de comentar la estructura de los demás archivos es necesario explicar cómo se realiza el manejo de eventos producidos en la interfaz, como puede ser la pulsación de un botón en una ventana. En GtkAda la interacción entre la interfaz y la aplicación se hace por medio de señales (*signals*), de hecho, la mayoría de las acciones que realiza el usuario sobre la aplicación gráfica provocan la emisión de una señal. Podemos decir que una señal no es más que un mensaje que un objeto quiere enviar y que está relacionada con ciertos eventos que se producen durante la existencia de un componente. Además, dependiendo del tipo de componente, éste puede tener asociadas varias señales (p. ej. un botón) o carecer de ellas (como es el caso de una etiqueta). Para que la aplicación responda a los eventos definidos para un determinado componente se ha de conectar la correspondiente señal a un procedimiento especial llamado manejador (*handler* o *callback*). Este manejador será invocado cada vez que la señal se emita, haciendo que la aplicación lleve a cabo las acciones necesarias para atender dicha señal.

Los ficheros denominados *<nombre_componente>_pkg-callbacks.ads* y *<nombre_componente>_pkg-callbacks.adb* contienen los manejadores (*callbacks*) conectados a las señales emitidas por el componente (o por alguno de sus elementos). Dentro de los manejadores se realizan una o varias llamadas GtkAda para que la aplicación realice las operaciones pertinentes (destruir una ventana, cambiar una etiqueta, ocultar una tabla, etc.).

Como se ha podido observar, los archivos asociados a los componentes de alto nivel comparten una estructura común si bien el código que contienen es muy diferente. Debido a que el número de archivos de este tipo es elevado (al tener 39 componentes de alto nivel, Glade generó 156 archivos) no vamos a comentar en profundidad el código de cada uno de ellos ya que esto conllevaría una redacción demasiado extensa para este Trabajo. Así pues diremos que, básicamente, el código que contienen se compone de llamadas GtkAda que se encargan de crear el objeto siguiendo el diseño realizado en Glade y de conectar las señales emitidas por el objeto a los manejadores correspondientes para obtener el comportamiento deseado en la aplicación gráfica.

Existen dos archivos más que requieren especial atención: el primero de ellos se llama *gmasteditor.adb* y contiene el código de control de la aplicación y de inicialización y creación de la ventana principal, el otro archivo se denomina *callbacks_mast_editor.ads* y contiene las principales instanciaciones de la aplicación a los paquetes genéricos GtkAda que se encargan del manejo de señales (*Gtk.Handlers*).

4.3.2. Paquetes Mast_Editor

El código de estos paquetes es utilizado para representar gráficamente el modelo MAST dentro de los diagramas contenidos en las solapas de la ventana principal del editor, así como para mostrar y recoger los valores de los atributos de los elementos utilizando las ventanas auxiliares. En la creación de estos paquetes se han utilizando técnicas de programación orientada a objetos, partiendo de uno de los paquetes incluidos en la librería GtkAda y dando lugar a una jerarquía de paquetes que sigue un esquema similar al existente en los paquetes de la librería MAST (ver figura 4.3).

La visualización en la pantalla de los elementos que componen el modelo MAST de un sistema a través de diferentes diagramas constituye una de las funcionalidades más importantes de la interfaz gráfica. Para implementar esa funcionalidad se utilizó el paquete *Gtkada.Canvas* [19] de la librería GtkAda, que proporciona un área de dibujo o *canvas* interactivo en el cual se pueden colocar objetos, moverlos con el ratón,

relacionarlos mediante flechas, etc. Dichas características le hacen apropiado para utilizarlo en la representación de diagramas y ese fue el principal motivo para la elección de ese paquete como base para la programación de los paquetes Mast_Editor.

Cada uno de los diagramas de la interfaz lleva asociado un canvas en el que aparecen diferentes elementos gráficos. Todos los objetos colocados en los canvas se derivan del tipo abstracto *Canvas_Item_Record* declarado en el paquete *Gtkada.Canvas* y, por consiguiente, "heredan" todas las *operaciones primitivas* definidas para ese tipo, dichas operaciones no fueron modificadas salvo en dos casos: los procedimientos *Draw* y *On_Button_Click*. El primer procedimiento se sobreescibió para determinar el aspecto que presentarían los objetos en la pantalla mientras que el segundo se redefinió para especificar el comportamiento de los objetos ante los eventos del ratón. Estas modificaciones se comentarán posteriormente en este apartado.

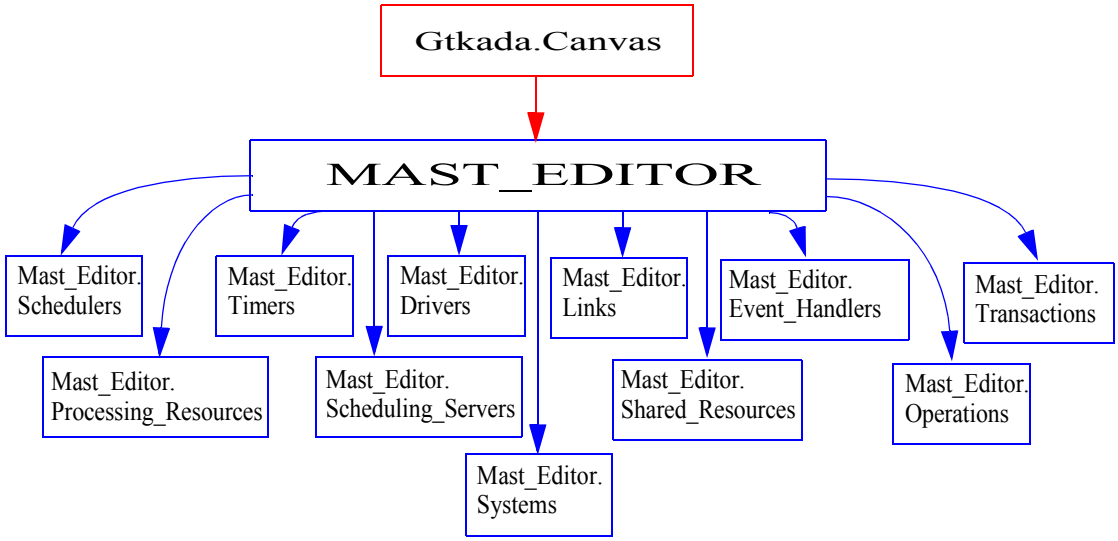

Figura 4.3: Jerarquía y dependencias de paquetes Mast_Editor

La figura 4.3 muestra la jerarquía de los *paquetes Mast_Editor*. Para una mejor comprensión de la figura y de la jerarquía pasaremos a enumerar primero y, en los distintos subapartados, a describir cada uno de los paquetes programados:

- **Mast_Editor:** Paquete en que se define el tipo de datos abstracto "padre" que será extendido en los paquetes "hijos" para representar los diferentes elementos del modelo MAST.
- **Mast_Editor.Processing_Resources:** Paquete para la representación y configuración de los recursos de procesado.
- **Mast_Editor.Timers:** Paquete para la representación y configuración de los temporizadores.
- **Mast_Editor.Drivers:** Paquete para la representación y configuración de los controladores de red.
- **Mast_Editor.Schedulers:** Paquete para la representación y configuración de los planificadores.
- **Mast_Editor.Scheduling_Servers:** Paquete para la representación y configuración de los servidores de planificación.

- **Mast_Editor.Shared_Resources:** Paquete para la representación y configuración de los recursos compartidos.
- **Mast_Editor.Operations:** Paquete para la representación y configuración de las operaciones.
- **Mast_Editor.Transactions:** Paquete para la representación y configuración de las transacciones.
- **Mast_Editor.Event_Handlers:** Paquete para la representación y configuración de los manejadores de eventos.
- **Mast_Editor.Links:** Paquete para la representación y configuración de los eventos.
- **Mast_Editor.Systems:** Paquete para estructurar y almacenar toda la información referente a la representación gráfica del modelo.

4.3.2.1. Mast_Editor

Este paquete es muy importante ya que es el paquete "padre" del que derivan los demás. En él se declara un tipo de datos abstracto que servirá como tipo de partida para representar en pantalla los elementos de modelo. Para este tipo se definen algunos campos y algunas operaciones primitivas que serán "heredados" por las extensiones que se realicen en los paquetes "hijos". El paquete está compuesto por dos tipos de datos, dos funciones y un procedimiento:

```
package Mast_Editor is

   type ME_Object is abstract new Canvas_Item_Record with record
      Name : Var_Strings.Var_String;
      Canvas_Name : Var_Strings.Var_String;
      Color_Name : Var_Strings.Var_String;
      W, H, X_Coord, Y_Coord : Gint;
   end record;

   function Name (Item : in ME_Object)
                 return Var_String is abstract;

   procedure Print
     (File : Ada.Text_IO.File_Type;
      Item : in out ME_Object;
      Indentation : Positive;
      Finalize : Boolean := False) is abstract;

   type ME_Object_Ref is access all ME_Object'Class;

   function Name (Item_Ref : in  ME_Object_Ref)
                 return Var_String is abstract;

end Mast_Editor;
```

ME_Object es el tipo de datos abstracto derivado del tipo *Canvas_Item_Record* (definido en el paquete Gtkada.Canvas) que posteriormente se extenderá para representar gráficamente los objetos en los diferentes canvas de la aplicación. Sus campos son:

- **Name, Canvas_Name, Color_Name:** Son cadenas de caracteres que indican el nombre del objeto, el nombre del canvas al que pertenece el objeto y el nombre del color con el que se representa dicho objeto.
- **W, H:** Son enteros que indican el ancho y el alto del objeto, respectivamente.

- **X_Coord, Y_Coord:** Son enteros que indican la coordenada "x" e "y", respectivamente, del punto donde está colocado el objeto en el canvas.

El tipo *ME_Object_Ref* es un puntero al tipo *ME_Object* o a cualquiera de sus tipos derivados.

Las funciones abstractas *Name* devuelven el nombre del objeto tomando como argumento el objeto o un puntero al objeto. El procedimiento *Print* se utilizará para guardar en archivo el objeto y sus propiedades. Al ser abstractos, estas funciones y este procedimiento serán redefinidos posteriormente para tipos no abstractos derivados del tipo *ME_Object.*

4.3.2.2. Mast_Editor.Processing_Resources

Este paquete incluye las funciones y procedimientos programados para representar y configurar los Recursos de Procesado (procesadores y redes). Dentro de él se declara un nuevo tipo abstracto derivado del tipo *ME_Object*:

```
type ME_Processing_Resource is abstract new ME_Object with record
   Res : Mast.Processing_Resources.Processing_Resource_Ref;
end record;
```

El campo *Res* es un puntero al tipo *Processing_Resource* del paquete *Mast.Processing_Resources* así el objeto gráfico queda asociado al tipo de dato que va a representar en la pantalla. Para *ME_Processing_Resource*, se redefinen la función *Name* y el procedimiento *Print*, ambos heredados de *ME_Object*:

```
function Name (Item : in ME_Processing_Resource)
              return Var_String;

procedure Print
  (File : Ada.Text_IO.File_Type;
   Item : in out ME_Processing_Resource;
   Indentation : Positive;
   Finalize : Boolean := False);
```

Se definen dos nuevos procedimientos abstractos llamados *Write_Parameters* y *Read_Parameters* cuya función es escribir y leer, respectivamente, los atributos de los recursos de procesado utilizando las ventanas auxiliares:

```
procedure Write_Parameters (Item : access ME_Processing_Resource;
   Dialog : access Gtk_Dialog_Record'Class) is abstract;

procedure Read_Parameters (Item : access ME_Processing_Resource;
   Dialog : access Gtk_Dialog_Record'Class) is abstract;
```

Además se declara un nuevo tipo llamado *ME_Processing_Resource_Ref* que es un puntero a *ME_Processing_Resource* o a sus tipos derivados:

```
type ME_Processing_Resource_Ref is access all ME_Processing_Resource'Class;
```

Para este tipo, se redefine la función *Name*, heredada de *ME_Object_Ref* y se realiza una instanciación al paquete genérico *Named_Lists* de la librería MAST, que permite crear una estructura de datos dinámica en la cual los punteros se van a ordenar por nombre en una lista. Dicha lista se guardará utilizando el procedimiento *Print*:

```
function Name (Item_Ref : in ME_Processing_Resource_Ref)
               return Var_String;

package Lists is new Named_Lists
  (Element => ME_Processing_Resource_Ref,
   Name    => Name);

procedure Print
  (File : Ada.Text_IO.File_Type;
   The_List : in out Lists.List;
   Indentation: Positive);
```

En el paquete se crean dos tipos abstractos derivados de *ME_Processing_Resource* sin añadir ningún campo:

```
type ME_Processor is abstract new ME_Processing_Resource with null record;
type ME_Network is abstract new ME_Processing_Resource with null record;
```

El primero será usado para representar los procesadores mientras que el segundo se utilizará para representar las redes en la pantalla. A partir de estos dos últimos se declaran los tipos *ME_Regular_Processor* y *ME_Packet_Based_Network* que servirán para dibujar los procesadores y las redes basadas en paquetes, respectivamente:

```
type ME_Regular_Processor is new ME_Processor with null record;
type ME_Packet_Based_Network is new ME_Network with null record;
```

Como puede verse estos tipos ya no son abstractos como los anteriores y para ellos se redefinen algunos de los procedimientos heredados:

```
procedure Write_Parameters (Item : access ME_Regular_Processor;
                           Dialog : access Gtk_Dialog_Record'Class);

procedure Read_Parameters (Item : access ME_Regular_Processor;
                          Dialog : access Gtk_Dialog_Record'Class);

procedure Draw (Item : access ME_Regular_Processor;
                Canvas : access Interactive_Canvas_Record'Class;
                GC : Gdk.GC.Gdk_GC;
                Xdest, Ydest : Gint);

procedure On_Button_Click (Item : access ME_Regular_Processor;
                          Event : Gdk.Event.Gdk_Event_Button);

procedure Print
  (File : Ada.Text_IO.File_Type;
   Item : in out ME_Regular_Processor;
   Indentation : Positive;
   Finalize : Boolean := False);
```

Write_Parameters y *Read_Parameters* realizan la escritura y lectura de los atributos de un procesador utilizando la ventana de diálogo diseñada para los procesadores.

Draw se encarga de dibujar el procesador dentro del canvas pasado como argumento. *On_Button_Click* determina la acción a realizar si se pulsa el ratón sobre el objeto: si se hace doble click con el botón izquierdo aparece el cuadro de diálogo con las propiedades del procesador, mientras que si se pulsa una vez el botón derecho aparece un menú emergente con diferentes opciones.

El procedimiento *Print* guarda en un fichero los atributos gráficos del objeto: nombre, canvas, coordenadas, etc. (ver apartado 4.3.6, “Ficheros del Editor Gráfico”).

De forma análoga a la vista para *ME_Regular_Processor*, para el tipo *ME_Packet_Based_Network* se sobreescriben los mismos procedimientos. La única diferencia es que en lugar de mostrarse el diálogo de los procesadores se muestra el diálogo de las redes y que el procedimiento *Draw* dibuja una red basada en paquetes.

Finalmente, se define el procedimiento *Run* cuya función es colocar el canvas y la barra de herramientas de los recursos de procesado dentro de un marco (*frame*). El único argumento que recibe el procedimiento es precisamente el marco, incluido en alguna de las solapas de la ventana principal de la interfaz:

```
procedure Run
  (Frame : access Gtk.Frame.Gtk_Frame_Record'Class);
```

4.3.2.3. Mast_Editor.Timers

Este paquete implementa las funciones y procedimientos utilizadas para dibujar y configurar los Temporizadores. Su estructura es similar a la del anterior paquete. Inicalmente se crea el tipo *ME_Timer* extendiendo el tipo *ME_Object* con los campos *Tim* (que es un puntero al tipo *System_Timer* (perteneciente al paquete *Mast.Timers*) y *Proc* (que es un puntero al tipo *Processing_Resource* (perteneciente al paquete *Mast.Processing_Resources*) y se sobreescriben la función *Name* y el procedimiento *Print* heredados de *ME_Object*:

```
type ME_Timer is abstract new ME_Object with record
   Tim : Mast.Timers.System_Timer_Ref;
   Proc: Mast.Processing_Resources.Processing_Resource_Ref;
end record;

function Name (Item : in ME_Timer) return Var_String;

procedure Print (File : Ada.Text_IO.File_Type;
                 Item : in out ME_Timer;
                 Indentation : Positive;
                 Finalize : Boolean := False);
```

Se definen dos procedimientos abstractos llamados *Write_Parameters* y *Read_Parameters* para recojer y mostrar, respectivamente, los atributos de los temporizadores en el cuadro de diálogo auxiliar:

```
procedure Write_Parameters (Item : access ME_Timer;
   Dialog : access Gtk_Dialog_Record'Class) is abstract;

procedure Read_Parameters (Item : access ME_Timer;
   Dialog : access Gtk_Dialog_Record'Class) is abstract;
```

Seguidamente veremos la definición del tipo *ME_Timer_Ref* que es un puntero a *ME_Timer* o a cualquiera de sus tipos derivados. Para él se redefine la función *Name*, heredada de *ME_Object_Ref* y se realiza una instanciación al paquete *Named_Lists* para crear una lista en la que los punteros se ordenan por nombre. El procedimiento *Print* salva la lista en un fichero.

```
type ME_Timer_Ref is access all ME_Timer'Class;

function Name (Item_Ref : in ME_Timer_Ref) return Var_String;

package Lists is new Named_Lists
  (Element => ME_Timer_Ref,
```

```
      Name    => Name);

   procedure Print (File : Ada.Text_IO.File_Type;
                    The_List : in out Lists.List;
                    Indentation: Positive);
```

El tipo *ME_System_Timer* se deriva de *ME_Timer* y representa un temporizador del sistema:

```
type ME_System_Timer is new ME_Timer with null record;
```

De forma análoga a la vista para el paquete *Mast_Editor.Processing_Resources*, para este tipo se sobreescriben cinco procedimientos: *Write_Parameters, Read_Parameters, Draw*, *On_Button_Click* y *Print*. La función de estos procedimientos ya se explicó en el apartado 4.3.2.2.

4.3.2.4. Mast_Editor.Drivers

Este paquete implementa las funciones y procedimientos utilizadas para dibujar y configurar los Controladores de red. Su estructura es muy similar a la del anterior paquete. Inicalmente se crea el tipo *ME_Driver* extendiendo el tipo *ME_Object* con los campos *Driv* (que es un puntero al tipo *Driver* (perteneciente al paquete *Mast.Drivers*), *Proc* y *Net* (que son punteros al tipo *Processing_Resource* (perteneciente al paquete *Mast.Processing_Resources*) y se sobreescriben la función *Name* y el procedimiento *Print* heredados de *ME_Object*:

```
type ME_Driver is abstract new ME_Object with record
   Driv : Mast.Drivers.Driver_Ref;
   Proc : Mast.Processing_Resources.Processing_Resource_Ref;
   Net  : Mast.Processing_Resources.Processing_Resource_Ref;
end record;

function Name (Item : in ME_Driver) return Var_String;

procedure Print (File : Ada.Text_IO.File_Type;
                 Item : in out ME_Driver;
                 Indentation : Positive;
                 Finalize : Boolean := False);
```

Se definen dos procedimientos abstractos llamados *Write_Parameters* y *Read_Parameters* para recojer y mostrar, respectivamente, los atributos de los controladores en el cuadro de diálogo auxiliar:

```
procedure Write_Parameters (Item : access ME_Driver;
   Dialog : access Gtk_Dialog_Record'Class) is abstract;

procedure Read_Parameters (Item : access ME_Driver;
    Dialog : access Gtk_Dialog_Record'Class) is abstract;
```

Seguidamente veremos la definición del tipo *ME_Driver_Ref* que es un puntero a *ME_Driver* o a cualquiera de sus tipos derivados. Para él se redefine la función *Name*, heredada de *ME_Object_Ref* y se realiza una instanciación al paquete *Named_Lists* para crear una lista en la que los punteros se ordenan por nombre. El procedimiento *Print* salva la lista en un fichero.

```
type ME_Driver_Ref is access all ME_Driver'Class;

function Name (Item_Ref : in ME_Driver_Ref) return Var_String;
```

```
package Lists is new Named_Lists
  (Element => ME_Driver_Ref,
   Name    => Name);

procedure Print (File : Ada.Text_IO.File_Type;
                 The_List : in out Lists.List;
                 Indentation: Positive);
```

El tipo *ME_Packet_Driver* se deriva de *ME_Driver* y representa un controlador de red:

```
type ME_Packet_Driver is new ME_Driver with null record;
```

De forma análoga a la vista para el paquete *Mast_Editor.Processing_Resources*, para este tipo se sobreescriben cinco procedimientos: *Write_Parameters, Read_Parameters, Draw*, *On_Button_Click* y *Print*. La función de estos procedimientos ya se explicó en el apartado 4.3.2.2.

4.3.2.5. Mast_Editor.Schedulers

Las funciones y procedimientos de este paquete se encargan de representar y configurar los Planificadores. Primeramente se declara un nuevo tipo abstracto derivado del tipo *ME_Object*, añadiendo el campo *Sche* que es un puntero al tipo *Scheduler* del paquete *Mast.Schedulers*, con ello objeto queda relacionado al tipo de dato que va a representar:

```
type ME_Scheduler is abstract new ME_Object with record
   Sche : Mast.Schedulers.Scheduler_Ref;
end record;
```

Los planificadores se representan en dos capas diferentes: Recursos de Procesado y Servidores de Planificación, para identificar a qué capa pertenece el objeto gráfico se utiliza el campo *Canvas_Name* (heredado de *ME_Object)* que contiene el nombre del canvas en donde se representa el objeto.

Para *ME_Schedduler*, se sobreescriben la función *Name* y el procedimiento *Print* heredados de *ME_Object*:

```
function Name (Item : in ME_Scheduler) return Var_String;

procedure Print (File : Ada.Text_IO.File_Type;
                 Item : in out ME_Scheduler;
                 Indentation : Positive;
                 Finalize : Boolean := False);
```

Los procedimientos abstractos *Write_Parameters* y *Read_Parameters* serán utilizados para escribir y leer, respectivamente, los atributos de los servidores utilizando la ventana de propiedades de los planificadores:

```
procedure Write_Parameters (Item : access ME_Scheduler;
   Dialog : access Gtk_Dialog_Record'Class) is abstract;

procedure Read_Parameters (Item : access ME_Scheduler;
   Dialog : access Gtk_Dialog_Record'Class) is abstract;
```

También se declara un nuevo tipo llamado *ME_Scheduler_Ref* que es un puntero a *ME_Scheduler* o a cualquiera de sus tipos derivados:

```
type ME_Scheduler_Ref is access all ME_Scheduler'Class;
```

Al igual que en el anterior paquete, para este tipo se redefine la función *Name*, heredada de *ME_Object_Ref* y se realiza una instanciación al paquete genérico *Named_Lists* de la librería MAST, que permite crear una estructura de datos dinámica en la cual los punteros se van a ordenar por nombre en una lista, la cual se guardará utilizando el procedimiento *Print*:

```
function Name (Item_Ref : in ME_Scheduler_Ref) return Var_String;

package Lists is new Named_Lists
  (Element => ME_Scheduler_Ref,
   Name    => Name);

procedure Print (File : Ada.Text_IO.File_Type;
                 The_List : in out Lists.List;
                 Indentation: Positive);
```

Tomando a *ME_Scheduler* como tipo de partida se crean los tipos *ME_Primary_Scheduler* y *ME_Secondary_Scheduler* que servirán para dibujar los planificadores primarios y secundarios respectivamente:

```
type ME_Primary_Scheduler is new ME_Scheduler with null record;
type ME_Secondary_Scheduler is new ME_Scheduler with null record;
```

Como puede verse estos tipos ya no son abstractos como los anteriores y para ellos se redefinen algunos de los procedimientos heredados:

```
procedure Write_Parameters (Item : access ME_Primary_Scheduler;
                           Dialog : access Gtk_Dialog_Record'Class);

procedure Read_Parameters (Item : access ME_Primary_Scheduler;
                          Dialog : access Gtk_Dialog_Record'Class);

procedure Draw (Item : access ME_Primary_Scheduler;
                Canvas : access Interactive_Canvas_Record'Class;
                GC : Gdk.GC.Gdk_GC;
                Xdest, Ydest : Gint);

procedure On_Button_Click (Item : access ME_Primary_Scheduler;
                          Event : Gdk.Event.Gdk_Event_Button);

procedure Show_Primary_Dialog (Widget : access Gtk_Button_Record'Class);

procedure Print (File : Ada.Text_IO.File_Type;
                 Item : in out ME_Primary_Scheduler;
                 Indentation : Positive;
                 Finalize : Boolean := False);
```

Write_Parameters y *Read_Parameters* realizan la escritura y lectura de los atributos de un planificador utilizando la ventana de diálogo diseñada para los planificadores primarios.

Draw se encarga de dibujar el planificador dentro del canvas pasado como argumento. *On_Button_Click* determina la acción a realizar si se pulsa el ratón sobre el objeto: si se hace doble click con el botón izquierdo aparece el cuadro de diálogo con las propiedades del planificador, mientras que si se pulsa una vez el botón derecho aparece un menú emergente con diferentes opciones.

El procedimiento *Print* guarda en un fichero los atributos gráficos del objeto: nombre, canvas, coordenadas, etc. (ver apartado 4.3.6, “Ficheros del Editor Gráfico”).

De forma análoga a la vista para *ME_Primary_Scheduler*, para el tipo *ME_Secondary_Scheduler* se sobreescriben los mismos procedimientos. La única diferencia es que en lugar de mostrarse el diálogo de los planificadores primarios se muestra el diálogo de los secundarios y que el procedimiento *Draw* dibuja un planificador secundario.

4.3.2.6. Mast_Editor.Scheduling_Servers

Las funciones y procedimientos de este paquete se encargan de representar y configurar los Servidores de Planificación. Primeramente se declara un nuevo tipo abstracto derivado del tipo *ME_Object*, añadiendo el campo *Ser* que es un puntero al tipo *Scheduling_Server* del paquete *Mast.Scheduling_Servers*, con ello objeto queda relacionado al tipo de dato que va a representar:

```
type ME_Scheduling_Server is abstract new ME_Object with record
   Ser : Mast.Scheduling_Servers.Scheduling_Server_Ref;
end record;
```

Para *ME_Scheduling_Server*, se sobreescriben la función *Name* y el procedimiento *Print* heredados de *ME_Object*:

```
function Name (Item : in ME_Scheduling_Server)
              return Var_String;

procedure Print
  (File : Ada.Text_IO.File_Type;
   Item : in out ME_Scheduling_Server;
   Indentation : Positive;
   Finalize : Boolean := False);
```

Los procedimientos abstractos *Write_Parameters* y *Read_Parameters* serán utilizados para escribir y leer, respectivamente, los atributos de los servidores utilizando la ventana de propiedades de los servidores de planificación:

```
procedure Write_Parameters (Item : access ME_Scheduling_Server;
  Dialog : access Gtk_Dialog_Record'Class) is abstract;

procedure Read_Parameters (Item : access ME_Scheduling_Server;
  Dialog : access Gtk_Dialog_Record'Class) is abstract;
```

También se declara un nuevo tipo llamado *ME_Scheduling_Server_Ref* que es un puntero a *ME_Scheduling_Server* o a cualquiera de sus tipos derivados:

```
type ME_Scheduling_Server_Ref is access all ME_Scheduling_Server'Class;
```

Al igual que en el anterior paquete, para este tipo se redefine la función *Name*, heredada de *ME_Object_Ref* y se realiza una instanciación al paquete genérico *Named_Lists* de la librería MAST, que permite crear una estructura de datos dinámica en la cual los punteros se van a ordenar por nombre en una lista, la cual se guardará utilizando el procedimiento *Print*:

```
function Name (Item_Ref : in ME_Scheduling_Server_Ref)
              return Var_String;

package Lists is new Named_Lists
```

```
   (Element => ME_Scheduling_Server_Ref,
    Name    => Name);

procedure Print
   (File : Ada.Text_IO.File_Type;
    The_List : in out Lists.List;
    Indentation: Positive);
```

Tomando a *ME_Scheduling_Server* como tipo abstracto de partida se crea el tipo *ME_Server* que servirá para dibujar los servidores y se redefinen algunos de los procedimientos heredados:

```
type ME_Server is new ME_Scheduling_Server with null record;

procedure Write_Parameters (Item : access ME_Server;
                           Dialog : access Gtk_Dialog_Record'Class);

procedure Read_Parameters (Item : access ME_Server;
                          Dialog : access Gtk_Dialog_Record'Class);

procedure Draw (Item : access ME_Server;
                Canvas : access Interactive_Canvas_Record'Class;
                GC : Gdk.GC.Gdk_GC;
                Xdest, Ydest : Gint);

procedure On_Button_Click (Item : access ME_Server;
                          Event : Gdk.Event.Gdk_Event_Button);

procedure Print (File : Ada.Text_IO.File_Type;
                 Item : in out ME_Server;
                 Indentation : Positive;
                 Finalize : Boolean := False);
```

Los dos primeros procedimientos llevan a cabo la escritura y lectura de los atributos de un servidor utilizando la ventana de diálogo de los servidores.

Draw dibuja el servidor en el canvas, mientras que *On_Button_Click* muestra el diálogo de propiedades o el menú emergente de la misma manera a como se explicó en el anterior apartado. Por otra parte, *Print* guarda en un fichero los atributos gráficos del objeto.

Existe otro procedimiento llamado *Run* cuya función es colocar el canvas y la barra de herramientas de los servidores de planificación en un marco de la ventana principal:

```
procedure Run
   (Frame : access Gtk.Frame.Gtk_Frame_Record'Class);
```

4.3.2.7. Mast_Editor.Shared_Resources

Este paquete implementa las funciones y procedimientos utilizados para dibujar y configurar los Recursos Compartidos. Su estructura es similar a la de los anteriores paquetes. Inicalmente se crea el tipo *ME_Shared_Resource* extendiendo el tipo *ME_Object* con el campo *Share,* que es un puntero al tipo *Shared_Resource* (perteneciente al paquete *Mast.Shared_Resources*) y se sobreescriben la función *Name* y el procedimiento *Print* heredados de *ME_Object*:

```
type ME_Shared_Resource is abstract new ME_Object with record
   Share : Mast.Shared_Resources.Shared_Resource_Ref;
end record;
```

```
function Name (Item : in ME_Shared_Resource)
              return Var_String;

procedure Print
  (File : Ada.Text_IO.File_Type;
   Item : in out ME_Shared_Resource;
   Indentation : Positive;
   Finalize : Boolean := False);
```

Se definen dos procedimientos abstractos llamados *Write_Parameters* y *Read_Parameters* para recojer y mostrar, respectivamente, los atributos de los recursos compartidos en el cuadro de diálogo auxiliar:

```
procedure Write_Parameters (Item : access ME_Shared_Resource;
  Dialog : access Gtk_Dialog_Record'Class) is abstract;

procedure Read_Parameters (Item : access ME_Shared_Resource;
   Dialog : access Gtk_Dialog_Record'Class) is abstract;
```

Seguidamente veremos la definición del tipo *ME_Shared_Resource_Ref* que es un puntero a *ME_Shared_Resource* o a cualquiera de sus tipos derivados. Para él se redefine la función *Name*, heredada de *ME_Object_Ref* y se realiza una instanciación al paquete *Named_Lists* para crear una lista en la que los punteros se ordenan por nombre. El procedimiento *Print* salva la lista en un fichero.

```
type ME_Shared_Resource_Ref is access all ME_Shared_Resource'Class;

function Name (Item_Ref : in ME_Shared_Resource_Ref)
              return Var_String;

package Lists is new Named_Lists
  (Element => ME_Shared_Resource_Ref,
   Name    => Name);

procedure Print
  (File : Ada.Text_IO.File_Type;
   The_List : in out Lists.List;
   Indentation: Positive);
```

Los tipos *ME_Priority_Inheritance_Resource*, *ME_Immediate_Ceiling_Resource* y *ME_SRP_Resource* se derivan de *ME_Shared_Resource* y representan a los recursos de herencia de prioridad, de techo de prioridad y de Stack Resource Protocol respectivamente:

```
type ME_Priority_Inheritance_Resource is new ME_Shared_Resource with
   null record;
type ME_Immediate_Ceiling_Resource is new ME_Shared_Resource with
   null record;
type ME_SRP_Resource is new ME_Shared_Resource with null record;
```

De forma análoga a la vista para el paquete *Mast_Editor.Processing_Resources*, para dichos tipos se sobreescriben cinco procedimientos: *Write_Parameters, Read_Parameters, Draw*, *On_Button_Click* y *Print*. La función de estos procedimientos ya se explicó en el apartado 4.3.2.2.

Este paquete también contiene otro procedimiento *Run* que colocar el canvas y la barra de herramientas de los recursos compartidos en uno de los marcos que contiene la ventana principal:

```
procedure Run
  (Frame : access Gtk.Frame.Gtk_Frame_Record'Class);
```

4.3.2.8. Mast_Editor.Operations

La función de este paquete es configurar y representar las Operaciones (simples, compuestas y de transmisión de paquetes). La estructura que presenta es muy parecida a la del anterior paquete. Dentro de él se declara un nuevo tipo abstracto derivado del tipo *ME_Object*:

```
type ME_Operation is abstract new ME_Object with record
   Op : Mast.Operations.Operation_Ref;
end record;
```

Op es un puntero al tipo *Operation* (del paquete *Mast.Operations)* que es el tipo de dato que va a representar. Para *ME_Operation*, se redefinen la función *Name* y el procedimiento *Print* heredados de *ME_Object*:

```
function Name (Item : in ME_Operation)
              return Var_String;

procedure Print
  (File : Ada.Text_IO.File_Type;
   Item : in out ME_Operation;
   Indentation : Positive;
   Finalize : Boolean := False);
```

A continuación se crean los procedimientos abstractos *Write_Parameters* y *Read_Parameters* para recojer y mostrar, respectivamente, los atributos de las operaciones en las ventanas auxiliares:

```
procedure Write_Parameters (Item : access ME_Operation;
  Dialog : access Gtk_Dialog_Record'Class) is abstract;

procedure Read_Parameters (Item : access ME_Operation;
  Dialog : access Gtk_Dialog_Record'Class) is abstract;
```

Posteriormente se crea el tipo *ME_Operation_Ref*, un puntero a *ME_Operation* o a cualquiera de sus tipos derivados. Para él se redefine la función *Name*, heredada de *ME_Object_Ref* y se realiza otra instanciación al paquete *Named_Lists* para crear una lista de punteros ordenados por nombre. El procedimiento *Print* se encargará de salvar la lista en un fichero.

```
type ME_Operation_Ref is access all ME_Operation'Class;

function Name (Item_Ref : in ME_Operation_Ref)
              return Var_String;

package Lists is new Named_Lists
  (Element => ME_Operation_Ref,
   Name    => Name);

procedure Print
  (File : Ada.Text_IO.File_Type;
   The_List : in out Lists.List;
   Indentation: Positive);
```

Los tipos *ME_Simple_Operation*, *ME_Composite_Operation* y *ME_Message_Transmission_Operation* se derivan de *ME_Operation.* El primero representa las operaciones simples, el segundo representa las operaciones compuestas y el tercero las operaciones de transmisión de mensajes.

```
type ME_Simple_Operation is new ME_Operation with null record;
type ME_Composite_Operation is new ME_Operation with null record;
type Me_Message_Transmission_Operation is new Me_Operation with null
   record;
```

Al igual que en el paquete anterior, para dichos tipos se sobreescriben cinco procedimientos: *Write_Parameters, Read_Parameters, Draw*, *On_Button_Click* y *Print*. Estos procedimientos se explicaron en detalle anteriormente.

El procedimiento *Run* coloca el canvas y la barra de herramientas de las operaciones un marco de la ventana principal:

```
procedure Run
  (Frame : access Gtk.Frame.Gtk_Frame_Record'Class);
```

4.3.2.9. Mast_Editor.Transactions

Las funciones y procedimientos de este paquete se encargan de representar y configurar las Transacciones. Primeramente se declara un nuevo tipo abstracto derivado del tipo *ME_Object*, añadiendo el campo *Tran* (que es un puntero al tipo *Transaction* del paquete *Mast.Transactions*) y *Dialog* (que es un puntero al diálogo asociado a cada transacción):

```
type ME_Transaction is abstract new ME_Object with record
   Tran : Mast.Transactions.Transaction_Ref;
   Dialog : Trans_Dialog_Pkg.Trans_Dialog_Access;
end record;
```

Para *ME_Transaction*, se sobreescriben la función *Name* y el procedimiento *Print* heredados de *ME_Object*:

```
function Name (Item : in ME_Transaction)
              return Var_String;

procedure Print
  (File : Ada.Text_IO.File_Type;
   Item : in out ME_Transaction;
   Indentation : Positive;
   Finalize : Boolean := False);
```

Los procedimientos abstractos *Write_Parameters* y *Read_Parameters* serán utilizados para escribir y leer, respectivamente, los atributos de las transacciones utilizando la ventana de propiedades de la transacción asociada:

```
procedure Write_Parameters (Item : access ME_Transaction) is abstract;

procedure Read_Parameters (Item : access ME_Transaction) is abstract;
```

También se declara un nuevo tipo llamado *ME_Transaction_Ref* que es un puntero a *ME_Transaction* o a cualquiera de sus tipos derivados:

```
type ME_Transaction_Ref is access all ME_Transaction'Class;
```

Al igual que en el anterior paquete, para este tipo se redefine la función *Name*, heredada de *ME_Object_Ref* y se realiza una instanciación al paquete genérico *Named_Lists* de la librería MAST, que permite crear una estructura de datos dinámica en la cual los punteros se van a ordenar por nombre en una lista, la cual se guardará utilizando el procedimiento *Print*:

```
function Name (Item_Ref : in ME_Transaction_Ref)
              return Var_String;

package Lists is new Named_Lists
  (Element => ME_Transaction_Ref,
   Name    => Name);

procedure Print
  (File : Ada.Text_IO.File_Type;
   The_List : in out Lists.List;
   Indentation: Positive);
```

Tomando a *ME_Transaction* como tipo abstracto de partida se crea el tipo *ME_Regular_Transacion* que servirá para dibujar las transacciones regulares y se redefinen algunos de los procedimientos heredados:

```
type ME_Regular_Transaction is new ME_Transaction with null record;

procedure Write_Parameters (Item : access ME_Regular_Transaction);

procedure Read_Parameters (Item : access ME_Regular_Transaction);

procedure Draw (Item : access ME_Regular_Transaction;
                Canvas : access Interactive_Canvas_Record'Class;
                GC : Gdk.GC.Gdk_GC;
                Xdest, Ydest : Gint);

procedure On_Button_Click (Item : access ME_Regular_Transaction;
                           Event : Gdk.Event.Gdk_Event_Button);

procedure Print
  (File : Ada.Text_IO.File_Type;
   Item : in out ME_Regular_Transaction;
   Indentation : Positive;
   Finalize : Boolean := False);
```

Los dos primeros procedimientos llevan a cabo la escritura y lectura de los atributos de una transacción utilizando la ventana de diálogo de las transacciones.

Draw dibuja la tranasacción en el canvas, mientras que *On_Button_Click* muestra el diálogo de propiedades o el menú emergente de la misma manera a como se explicó en el anterior apartado. Por otra parte, *Print* guarda en un fichero los atributos gráficos del objeto.

Existe otro procedimiento llamado *Run* cuya función es colocar el canvas y la barra de herramientas de las transacciones en un marco de la ventana principal:

```
procedure Run
  (Frame : access Gtk.Frame.Gtk_Frame_Record'Class);
```

4.3.2.10. Mast_Editor.Links

Las funciones y procedimientos de este paquete se encargan de representar y configurar los Eventos de las transacciones. Primeramente se declara un nuevo tipo abstracto derivado del tipo *ME_Object*, añadiendo el campo *Lin* (que es un puntero al tipo *Link* del paquete *Mast.Graphs*) y *ME_Tran* (que es un puntero al tipo *ME_Transaction* e indica la transacción a la que está asociado):

```
type ME_Link is abstract new ME_Object with record
   Lin : Mast.Graphs.Link_Ref;
   ME_Tran : Mast_Editor.Transactions.ME_Transaction_Ref;
end record;
```

Para *ME_Link*, se sobreescriben la función *Name* y el procedimiento *Print* heredados de *ME_Object*:

```
function Name (Item : in ME_Link) return Var_String;

procedure Print (File : Ada.Text_IO.File_Type;
                 Item : in out ME_Link;
                 Indentation : Positive;
                 Finalize : Boolean := False);
```

Los procedimientos abstractos *Write_Parameters* y *Read_Parameters* serán utilizados para escribir y leer, respectivamente, los atributos de los eventos:

```
procedure Write_Parameters (Item : access ME_Link;
   Dialog : access Gtk_Dialog_Record'Class) is abstract;

procedure Read_Parameters (Item : access ME_Link;
   Dialog : access Gtk_Dialog_Record'Class) is abstract;
```

También se declara un nuevo tipo llamado *ME_Link_Ref* que es un puntero a *ME_Link* o a cualquiera de sus tipos derivados:

```
type ME_Link_Ref is access all ME_Link'Class;
```

Al igual que en el anterior paquete, para este tipo se redefine la función *Name*, heredada de *ME_Object_Ref* y se realiza una instanciación al paquete genérico *Named_Lists* de la librería MAST, que permite crear una estructura de datos dinámica en la cual los punteros se van a ordenar por nombre en una lista, la cual se guardará utilizando el procedimiento *Print*:

```
function Name (Item_Ref : in ME_Link_Ref) return Var_String;

package Lists is new Named_Lists
  (Element => ME_Link_Ref,
   Name    => Name);

procedure Print (File : Ada.Text_IO.File_Type;
                 The_List : in out Lists.List;
                 Indentation: Positive);
```

Tomando a *ME_Link* como tipo de partida se crean los tipos *ME_Internal_Link* y *ME_External_Link* que servirán para dibujar los eventos internos y externos respectivamente:

```
type ME_Internal_Link is new ME_Link with null record;
type ME_External_Link is new ME_Link with null record;
```

Como puede verse estos tipos ya no son abstractos como los anteriores y para ellos se redefinen algunos de los procedimientos heredados:

```
procedure Write_Parameters (Item : access ME_Internal_Link;
                            Dialog : access Gtk_Dialog_Record'Class);

procedure Read_Parameters (Item : access ME_Internal_Link;
                           Dialog : access Gtk_Dialog_Record'Class);

procedure Draw (Item : access ME_Internal_Link;
                Canvas : access Interactive_Canvas_Record'Class;
                GC : Gdk.GC.Gdk_GC;
                Xdest, Ydest : Gint);

procedure On_Button_Click (Item : access ME_Internal_Link;
                           Event : Gdk.Event.Gdk_Event_Button);

procedure Show_Internal_Dialog (Widget : access Gtk_Widget_Record'Class;
                                Res : Me_Transaction_Ref);

procedure Print (File : Ada.Text_IO.File_Type;
                 Item : in out ME_Internal_Link;
                 Indentation : Positive;
                 Finalize : Boolean := False);
```

Write_Parameters y *Read_Parameters* realizan la escritura y lectura de los atributos de un evento interno utilizando la ventana de diálogo diseñada para los eventos internos.

Draw se encarga de dibujar el evento dentro del canvas pasado como argumento. *On_Button_Click* determina la acción a realizar si se pulsa el ratón sobre el objeto: si se hace doble click con el botón izquierdo aparece el cuadro de diálogo con las propiedades del evento, mientras que si se pulsa una vez el botón derecho aparece un menú emergente con diferentes opciones.

El procedimiento *Print* guarda en un fichero los atributos gráficos del objeto: nombre, canvas, coordenadas, etc. (ver apartado 4.3.6, “Ficheros del Editor Gráfico”).

De forma análoga a la vista para *ME_Internal_Link*, para el tipo *ME_External_Link* se sobreescriben los mismos procedimientos, la única diferencia es que en lugar de mostrarse el diálogo de los eventos internos se muestra el diálogo de los externos y que el procedimiento *Draw* dibuja un evento externo.

4.3.2.11. Mast_Editor.Event_Handlers

Este paquete implementa las funciones y procedimientos utilizadas para dibujar y configurar los Manejadores de Eventos. Su estructura es similar a la de los anteriores paquetes. Inicalmente se crea el tipo *ME_Event_Handler* extendiendo el tipo *ME_Object* con el campo *Han* (que es un puntero al tipo *Event_Handler* del paquete *Mast.Graphs*), *Me_Tran* (que es un puntero al objeto *ME_Transaction* al que está asociado) y *Id* (que es un entero que se correponde con la posición que ocupa el manejador en la lista de manejadores de la transacción). Además, se sobreescriben la función *Name* y el procedimiento *Print* heredados de *ME_Object*:

```
type ME_Event_Handler is abstract new ME_Object with record
   Han : Mast.Graphs.Event_Handler_Ref;
   ME_Tran : Mast_Editor.Transactions.ME_Transaction_Ref;
   Id : Natural := 1;
end record;
```

```
function Name (Item : in ME_Event_Handler) return Var_String;

procedure Print (File : Ada.Text_IO.File_Type;
                 Item : in out ME_Event_Handler;
                 Indentation : Positive;
                 Finalize : Boolean := False);
```

Se definen dos procedimientos abstractos llamados *Write_Parameters* y *Read_Parameters* para recojer y mostrar, respectivamente, los atributos de los recursos compartidos en el cuadro de diálogo auxiliar:

```
procedure Write_Parameters (Item : access ME_Event_Handler;
   Dialog : access Gtk_Dialog_Record'Class) is abstract;

procedure Read_Parameters (Item : access ME_Event_Handler;
   Dialog : access Gtk_Dialog_Record'Class) is abstract;
```

Seguidamente veremos la definición del tipo *ME_Event_Handler_Ref* que es un puntero a *ME_Event_Handler* o a cualquiera de sus tipos derivados. Para él se redefine la función *Name*, heredada de *ME_Object_Ref* y se realiza una instanciación al paquete *Named_Lists* para crear una lista en la que los punteros se ordenan por nombre. El procedimiento *Print* salva la lista en un fichero.

```
type ME_Event_Handler_Ref is access all ME_Event_Handler'Class;

function Name (Item_Ref : in ME_Event_Handler_Ref) return Var_String;

package Lists is new Named_Lists
  (Element => ME_Event_Handler_Ref,
   Name    => Name);

procedure Print (File : Ada.Text_IO.File_Type;
                 The_List : in out Lists.List;
                 Indentation: Positive);
```

Los tipos *ME_Simple_Event_Handler*, *ME_Multi_Input_Event_Handler* y *ME_Multi_Output_Event_Handler* se derivan de *ME_Event_Handler* y representan a los manejadores de eventos simples, de múltiples entradas y de múltiples salidas:

```
type ME_Simple_Event_Handler is new ME_Event_Handler with null record;
type ME_Multi_Input_Event_Handler is new ME_Event_Handler with null record;
type ME_Multi_Output_Event_Handler is new ME_Event_Handler with null
   record;
```

De forma análoga a la vista para el paquete *Mast_Editor.Processing_Resources*, para dichos tipos se sobreescriben cinco procedimientos: *Write_Parameters, Read_Parameters, Draw*, *On_Button_Click* y *Print*. La función de estos procedimientos ya se explicó en el apartado 4.3.2.2.

4.3.2.12. Mast_Editor.Systems

Este paquete proporciona una estructura en la que se almacena la información de los objetos utilizados para la representación gráfica del sistema y una función que se encarga de guardar en un archivo los atributos de los objetos:

```
type ME_System is tagged record
   Model_Name           : Var_String:=Null_Var_string;
   Model_Date           : Date:="                    ";
```

```
   Me_Processing_Resources : MAST_Editor.Processing_Resources.Lists.List;
   Me_Shared_Resources     : MAST_Editor.Shared_Resources.Lists.List;
   Me_Operations           : MAST_Editor.Operations.Lists.List;
   Me_Transactions         : MAST_Editor.Transactions.Lists.List;
   Me_Scheduling_Servers   : MAST_Editor.Scheduling_Servers.Lists.List;
   Me_Schedulers           : MAST_Editor.Schedulers.Lists.List;
   Me_Timers               : MAST_Editor.Timers.Lists.List;
   Me_Drivers              : MAST_Editor.Drivers.Lists.List;
   Me_Schedulers_In_Server_Canvas : MAST_Editor.Schedulers.Lists.List;
   Me_Servers_In_Proc_Canvas : MAST_Editor.Scheduling_Servers.Lists.List;
   Me_Links                : MAST_Editor.Links.Lists.List;
   Me_Event_Handlers       : MAST_Editor.Event_Handlers.Lists.List;
end record;

procedure Print (File : Ada.Text_IO.File_Type;
                 The_System : in out Me_System;
                 Indentation : Positive:=1);
```

El tipo *ME_System* se compone de varias listas que contienen los punteros a los diferentes objetos utilizados para la visualización del sistema en pantalla.

4.3.3. Paquetes de Manejo de Archivos

En este apartado se comentarán los procedimientos que utiliza el editor para crear y acceder a los archivos en los que se guarda la información del modelo MAST. Cada modelo, identificado por su nombre, lleva asociados dos archivos de texto: el primero, llamado ***<nombre_modelo>.txt***, contiene la descripción ASCII del modelo mientras que el segundo, denominado ***<nombre_modelo>.mss***, contiene los parámetros asociados a la representación gráfica del modelo. Estos archivos se comentarán más detalladamente en el apartado 4.3.6, “Ficheros del Editor Gráfico”.

El paquete ***Editor_Actions*** incluye los procedimientos que se encargan de crear, leer y escribir los ficheros asociados al modelo MAST. A continuación se explicarán los tipos, variables y procedimientos definidos en el paquete.

Inicialmente, se declaran dos nuevos tipos:

```
type Me_Arguments is array (1..6) of Var_String;
type Me_Arguments_Ref is access Me_Arguments;
```

Me_Arguments es un array de seis elementos del tipo *Var_String* que se va ha utilizar en la lectura de los archivos ***<nombre_modelo>.mss***. Como se verá en el apartado 4.3.6, en estos archivos cada línea corresponde a uno de los objetos dibujados en el editor y contiene una cadena de caracteres en la que se incluyen los atributos asociados a los objetos: tipo, subtipo, nombre, canvas, coordenada “x” y coordenada “y”. El tipo *Me_Arguments* se utilizará para extraer los valores de esos atributos y asignarlos a los correspondientes campos del objeto.

Además se declaran cuatro variables globales:

```
Current_Filename : String_Access;
Editor_System : Mast_Editor.Systems.Me_System;
The_System : Mast.Systems.System;
Delimiter : constant String := ",";
```

Current_Filename es un puntero al string cuyo valor se corresponde con el nombre del fichero editado. *Editor_System* es la variable utilizada para guardar las listas de los diferentes objetos dibujados en los diagramas del editor. *The_System* contiene varias listas compuestas por los elementos del modelo MAST y sus respectivos atributos. *Delimiter* es

una constante carácter utilizada para denominar a ciertos objetos cuyo nombre se compone de otros dos nombres separados por este delimitador.

Los procedimientos incluídos en el paquete *Editor_Actions* son los siguientes:

```
function Id_Name_Is_Valid (Id : String) return Boolean;

procedure Clear_Canvas (Canvas : access Interactive_Canvas_Record'Class);

procedure Clear_All_Canvas;

procedure Add_Canvas_Link(Canvas : access Interactive_Canvas_Record'Class;
                          Item1 : access Canvas_Item_Record'Class;
                          Item2 : access Canvas_Item_Record'Class);

procedure Remove_Links (Canvas : access Interactive_Canvas_Record'Class;
                        Item1 : access Canvas_Item_Record'Class;
                        Item2 : access Canvas_Item_Record'Class;
                        Outgoing : Boolean := True);

procedure Remove_Old_Links(Canvas: access Interactive_Canvas_Record'Class;
                           Item : access Canvas_Item_Record'Class;
                           Outgoing : Boolean := True);

procedure New_File;

procedure Set_Window_Title;

procedure Load_System_Font
   (Font_Normal : in out Pango.Font.Pango_Font_Description;
   Font_Bold : in out Pango.Font.Pango_Font_Description);

procedure Reset_Editor_System
    (The_Editor_System: in out Mast_Editor.Systems.Me_System);

procedure Reset_Mast_System (The_Mast_System: in out Mast.Systems.System);

function Get_Number_Of_Items
    (Canvas : access Interactive_Canvas_Record'Class) return Glib.Guint;

procedure Show_Left_Top_Item
    (Canvas : access Interactive_Canvas_Record'Class);

procedure Show_All_Left_Top_Items;

procedure Show_Central_Item
    (Canvas : access Interactive_Canvas_Record'Class);

procedure Show_All_Central_Items;

function Number_Of_Input_Links
    (Canvas : access Interactive_Canvas_Record'Class;
     Item : access Canvas_Item_Record'Class) return Guint;

function Number_Of_Output_Links
    (Canvas : access Interactive_Canvas_Record'Class;
     Item : access Canvas_Item_Record'Class) return Guint;

function Search_Next_Zero_Input_Item
    (Canvas : access Interactive_Canvas_Record'Class) return Canvas_Item;

procedure Move_Item_To_Index_Position
    (Canvas : access Interactive_Canvas_Record'Class;
```

```
    Item : access Canvas_Item_Record'Class;
    Column_Index : in Integer;
    Row_Index : in Integer);

procedure Draw_Input_Items
    (Canvas : access Interactive_Canvas_Record'Class;
    Item : access Canvas_Item_Record'Class;
    Row_Index : in Integer;
    Column_Index : in Integer;
    Position_Matrix : in out Matrix;
    Max_Row_Index : in out Integer);

procedure Draw_Output_Items
    (Canvas : access Interactive_Canvas_Record'Class;
    Item : access Canvas_Item_Record'Class;
    Row_Index : in Integer;
    Column_Index : in Integer;
    Position_Matrix : in out Matrix;
    Max_Row_Index : in out Integer);

procedure Show_TXT_Graphs
    (Canvas : access Interactive_Canvas_Record'Class);

procedure Show_TXT_Items
    (Canvas : access Interactive_Canvas_Record'Class);

procedure Put_TXT_Item (Canvas : access Interactive_Canvas_Record'Class;
    Item : Mast_Editor.Me_Object_Ref);

procedure Put_Item_In_Canvas (Item : Mast_Editor.Me_Object_Ref);

procedure Draw_TXT_File_System
    (TXT_File_System : Mast.Systems.System := The_System;
    MSS_File_Exists : Boolean := False);

procedure Read_Editor_System (Filename : String;
                              Importing_System : Boolean := False);

procedure Extract_Delimiter (Composite_Name : in Var_String;
                             First_Name : out Var_String;
                             Second_Name : out Var_String);

procedure Read_Common_Params (Item : Mast_Editor.Me_Object_Ref;
                              Arguments : Me_Arguments_Ref);

procedure Save_Editor_System (Filename : String;
                              Saving_As :Boolean := False);

function Has_System return Boolean;

function Has_Editor_System return Boolean;

procedure Import_System (Filename : String;
                         System_Imported : in out MAST.Systems.System);

procedure Adjust_System_Lists (System_Imported : MAST.Systems.System);

procedure Read_System (Filename : String);

procedure Save_System (Filename : String);

function Name_Of_System return String;
```

```
No_System : exception;
```

Las funciones de cada uno de estos procedimientos se comentan a continuación:

- *Free*: libera la memoria ocupada por el nombre del fichero editado, para ello se hace una instanciación al paquete genérico *Unchecked_Deallocation*. Este procedimiento es utilizado cada vez que se abre o renombra un fichero.
- *Id_Name_Is_Valid*: comprueba la validez de un identificador según la nomenclatura MAST.
- *Clear_Canvas*: borra todos los elementos que aparecen en un canvas.
- *Clear_All_Canvas*: borra todos los canvas de la interfaz.
- *Add_Canvas_Link*: dibuja un enlace entre dos elementos de un canvas.
- *Remove_Links*: borra los enlaces entre dos elementos de un canvas.
- *Remove_Old_Links*: borra los enlaces de un elemento del canvas.
- *New_File*: asigna el valor nulo el nombre del archivo editado, inicializa las variables globales *Editor_System* y *The_System* y borra todos los canvas. Este procedimiento es invocado cada vez que se crea un modelo nuevo.
- *Set_Window_Title*: muestra el nombre del archivo editado en la parte superior de la ventana principal.
- *Load_System_Font*: carga la fuente utilizada por defecto en la plataforma (sistema operativo) sobre la que se ejecuta la aplicación.
- *Reset_Editor_System*: inicializa la variable global *Editor_System*.
- *Reset_Mast_System*: inicializa la variable global *The_System*.
- *Get_Number_Of_Items*: devuelve el número de elementos de un canvas.
- *Remove_Old_Links*: borra los enlaces de un elemento del canvas.
- *Show_Left_Top_Item*: muestra el elemento situado en la esquina superior izquierda del canvas.
- *Show_All_Left_Top_Item*s: realiza un “scroll” en todos los canvas mostrando el elemento situado en la esquina superior izquierda de cada uno de ellos.
- *Show_Central_Item*: muestra el elemento más centrico del canvas.
- *Show_All_Central_Item*s: realiza un “scroll” en todos los canvas mostrando el elemento más centrico de cada uno de ellos.
- *Number_Of_Input_Links*: devuelve el número de enlaces de entrada de un elemento del canvas.
- *Number_Of_Output_Links*: devuelve el número de enlaces de salida de un elemento del canvas.
- *Search_Next_Zero_Input_Item*: devuelve el siguiente elemento de la lista de objetos del canvas sin enlaces de entrada.
- *Move_Item_To_Index_Position*: desplaza el elemento del canvas a la posición indicada por los indices, realizando un escalado para obtener las coordenadas dentro de la matriz de posiciones del canvas.
- *Draw_Input_Items*: dibuja los enlaces de entrada del elemento del canvas situado en la posición especificada.

- *Draw_Output_Items*: dibuja los enlaces de salida del elemento del canvas situado en la posición especificada.
- *Show_TXT_Graphs*: dibuja todos los elementos del grafo de una transacción cuando no existe archivo *.mss.
- *Show_TXT_Items*: dibuja todos los elementos en el canvas de una determinada capa cuando no existe archivo *.mss.
- *Put_TXT_Item*: dibuja un elemento en el canvas especificado cuando no existe archivo *.mss.
- *Put_Item_In_Canvas*: dibuja un elemento en el canvas que especifica el canpo *Canvas* del objeto.
- *Draw_TXT_File_System*: dibuja todos los elementos del sistema cuando no existe archivo *.mss.
- *Read_Editor_System*: abre el fichero <*Filename*>.mss, lee línea a línea los argumentos asociados a los objetos usados en la representación gráfica del modelo, dibuja dichos objetos en los correspondientes canvas y actualiza la variable *Editor_System*.
- *Extract_Delimiter*: descompone un string en dos extrayendo de él el carácter especificado en la constante *Delimiter*.
- *Read_Common_Params*: extrae de cada línea del archivo *.mss el nombre, el canvas y las coordenadas de cada objeto a visualizar y asigna los valores a los respectivos campos definidos para el tipo *Me_Object*.
- *Save_Editor_System*: guarda en el archivo <*Filename*>.mss el valor de la variable global *Editor_System*.
- *Has_System*: verifica si hay sistema cargado.
- *Has_Editor_System*: verifica si hay sistema de elementos gráficos cargado.
- *Import_System*: abre el fichero <*Filename*>.txt, lee la descripción ASCII utilizando un parser incluído en las herramientas MAST e importa sus elementos al sistema editado.
- *Adjust_System_Lists*: realiza un ajuste en las listas del sistema editado una vez que se han importado nuevos elementos desde otro archivo, añadiendo dichos elementos a las listas del sistema correpondientes.
- *Read_System*: abre el fichero <*Filename*>.txt, lee la descripción ASCII utilizando un parser incluído en las herramientas MAST y actualiza la variable *The_System*.
- *Save_System*: guarda en el archivo <*Filename*>.txt el valor de la variable global *The_System*.
- *Name_Of_System*: devuelve el nombre del sistema.

4.3.4. Paquetes de Control de Cambios

El paquete ***Change_Control*** incluye los procedimientos que se encargan de realizar el control de cambios en la aplicación. Permite conocer si se ha hecho algún cambio en el archivo editado y si ha sido guardado en disco o no. A continuación se muestra la especificación del paquete:

```
package Change_Control is
```

```
   procedure Changes_Made;

   procedure Saved;

   function Saved_Changes return Boolean;

end Change_Control;
```

Como puede verse el paquete solo contiene dos procedimientos y una función:

- *Changes_Made*: especifica que se han hecho cambios en el archivo editado.
- *Saved*: indica que el fichero ha sido guardado en disco.
- *Saved_Changes*: permite saber si los cambios ha sido guardados o no.

4.3.5. Paquetes de Control de Herramientas

El paquete ***Simple_Transaction_Wizard_Control*** incluye los procedimientos que se encargan de realizar el control del asistente para la creación de transacciones simples. Permite controlar el proceso de creación de la transacción usando las diferentes ventanas de diálogo que componen el asistente (*wizard*). A continuación se muestra la especificación del paquete:

```
package Simple_Transaction_Wizard_Control is

   type Previous_Window is (Welcome_Win, Transaction_Win,
                            Input_Event_Win, Activity_Win,
                            Output_Event_Win, Complete_Win);

   type Simple_Transaction_Wizard_Record is record

      Previous_Win : Previous_Window;
      Wizard_Welcome_Dialog : Wizard_Welcome_Dialog_Access;
      Wizard_Transaction_Dialog : Wizard_Transaction_Dialog_Access;
      Wizard_Input_Dialog : Wizard_Input_Dialog_Access;
      Wizard_Activity_Dialog : Wizard_Activity_Dialog_Access;
      Wizard_Output_Dialog : Wizard_Output_Dialog_Access;
      Wizard_Completed_Dialog : Wizard_Completed_Dialog_Access;

   end record;

   type Simple_Transaction_Wizard_Access is access all
      Simple_Transaction_Wizard_Record;

   procedure Create_Simple_Transaction_Wizard;

end Simple_Transaction_Wizard_Control;
```

En el paquete se especifican tres nuevos tipos:

- *Previous_Window*: tipo enumerado que se utiliza para saber cuál fue la ventana del asistente anterior a la actual (sirve para controlar internamente en qué paso del proceso de creación se está).
- *Simple_Transaction_Wizard_Record*: tipo *record* compuesto por el indicador de ventana previa y por todas las ventanas que incluye el asistente.
- *Simple_Transaction_Wizard_Access*: puntero al tipo compuesto comentado anteriormente.

Como puede verse el paquete solo contiene un procedimiento:

- *Create_Simple_Transaction_Wizard*: crea la estructura de datos que se utilizará para controlar la creación de la transacción y la configuración de sus elementos y parámetros utilizando las pantallas del asistente.

4.3.6. Ficheros del Editor Gráfico

Cada vez que el usuario utiliza el editor gráfico para crear el modelo MAST de un sistema de tiempo real o modificar un modelo ya existente, se crean o se accede a dos ficheros de texto en los que se almacena la información asociada al modelo. Seguidamente se describe la función de ambos ficheros, cuyo nombre viene determinado por el nombre del modelo editado:

- El primer fichero contiene la especificación del modelo MAST mediante una descripción ASCII que sirve como entrada de las herramientas de análisis. Esta descripción se corresponde con un listado de los elementos que componen el modelo (procesadores, redes, controladores, recursos compartidos, operaciones, transacciones, etc.) y sus respectivos atributos (nombre, tipo, prioridades, tiempos, etc.). El archivo se denomina ***<nombre_modelo>.txt***.
- El segundo fichero contiene una lista con los atributos de los objetos utilizados en la representación gráfica del modelo: tipo, nombre, canvas y coordenadas. El nombre del archivo es ***<nombre_modelo>.mss***.

De esta manera, cuando se abre o se guarda un modelo utilizando el editor, la aplicación ha de acceder a ambos archivos de forma paralela y así poder realizar la lectura o escritura de los datos referentes al modelo (en el fichero **.txt*) y a su representación gráfica (en el fichero **.mss*). Es muy importante que el acceso a los ficheros se haga cuidadosamente, ya que si se producen errores de lectura o escritura esto puede tener como consecuencia una incorrecta representación del modelo o que incluso sea imposible poder visualizarlo en pantalla.

En este apartado no vamos a explicar las características de los ficheros *<nombre_modelo>.txt* ya que estos detalles quedan fuera del ámbito del presente trabajo. Sólo comentaremos que la aplicación realiza el acceso a dichos ficheros mediante un parser contenido dentro de las herramientas MAST y que su formato contiene la descripción ASCII del modelo definida en [1].

A continuación describiremos la estructura de los ficheros *<nombre_modelo>.mss*, cuyo diseño y programación fue una de las partes más importantes en la implementación del editor.

En el momento de elaborar el formato de estos ficheros se tuvo presente que en ellos se iban a almacenar sólamente los parámetros relativos a la representación gráfica del modelo, de hecho sólo se guardarían unos pocos atributos comunes definidos para los objetos representados, lo cual no implicaba una gran cantidad de información. Así pues, se optó por utilizar un formato de texto sencillo, en el cual cada línea corresponde a uno de los objetos dibujados y está compuesta por los valores de los atributos definidos para el objeto separados por un espacio. El formato de la línea es el siguiente:

> **[tipo] [subtipo] [nombre] [canvas] [coordenada_x] [coordenada_y]**

Como puede apreciarse, cada línea contiene una cadena de caracteres formada por seis *Var_String* separados entre sí por un espacio en blanco que se corresponden a los seis

atributos asociados a los objetos: tipo, subtipo, nombre, canvas, coordenada "x" y coordenada "y".

De esta manera, cada archivo **.mss* contiene un listado de los objetos representados y sus atributos compuesto por una serie de líneas de texto de longitud variable. A continuación se incluye un extracto del fichero **controller.mss** a modo de ejemplo:

```
ME_Processing_Resource Me_Regular_Processor cpu_1 Proc_Res_Canvas 139 184
ME_Scheduler Me_Primary_Scheduler cpu_1 Proc_Res_Canvas 109 73
ME_Scheduler Me_Primary_Scheduler cpu_1 Sched_Server_Canvas 271 182
ME_Scheduling_Server Me_Server control_task Sched_Server_Canvas 520 40
ME_Scheduling_Server Me_Server planning_task Sched_Server_Canvas 360 40
ME_Scheduling_Server Me_Server status_task Sched_Server_Canvas 200 40
ME_Scheduling_Server Me_Server emergency Sched_Server_Canvas 40 40
ME_Shared_Resource Me_Immediate_Ceiling_Resource data_server Shared_Res_Canvas 112 173
ME_Shared_Resource Me_Immediate_Ceiling_Resource comm_server Shared_Res_Canvas 111 55
ME_Operation Me_Simple_Operation read Operation_Canvas 547 190
ME_Operation Me_Simple_Operation write Operation_Canvas 207 183
ME_Operation Me_Simple_Operation send Operation_Canvas 377 185
ME_Operation Me_Simple_Operation receive Operation_Canvas 26 185
ME_Operation Me_Enclosing_Operation control Operation_Canvas 446 78
ME_Operation Me_Enclosing_Operation planning Operation_Canvas 223 77
ME_Operation Me_Enclosing_Operation status Operation_Canvas 28 82
ME_Operation Me_Simple_Operation emergency Operation_Canvas 275 275
ME_Transaction Me_Regular_Transaction control_task Transaction_Canvas 374 172
ME_Transaction Me_Regular_Transaction planning_task Transaction_Canvas 371 77
ME_Transaction Me_Regular_Transaction status_task Transaction_Canvas 108 142
ME_Transaction Me_Regular_Transaction emergency Transaction_Canvas 102 46
ME_Link Me_External_Link e1,control_task control_task 127 125
ME_Link Me_Internal_Link o1,control_task control_task 371 122
ME_Link Me_External_Link e2,planning_task planning_task 87 104
ME_Link Me_Internal_Link o2,planning_task planning_task 365 104
ME_Link Me_External_Link e3,status_task status_task 73 112
ME_Link Me_Internal_Link o3,status_task status_task 340 112
ME_Link Me_External_Link e4,emergency emergency 42 131
ME_Link Me_Internal_Link o4,emergency emergency 313 130
ME_Event_Handler Me_Simple_Event_Handler 1,control_task control_task 203 109
ME_Event_Handler Me_Simple_Event_Handler 1,planning_task planning_task 183 90
ME_Event_Handler Me_Simple_Event_Handler 1,status_task status_task 166 98
ME_Event_Handler Me_Simple_Event_Handler 1,emergency emergency 137 115
```

En este ejemplo se puede observar como en cada una de las líneas se especifica el tipo de objeto a representar en pantalla, el subtipo, su nombre, el nombre del color con el que se dibuja el objeto, las coordenadas de la posición que ocupa el objeto en el canvas y sus dimensiones.

Para realizar la lectura y escritura en este tipo de ficheros se programaron diversos procedimientos: *Read_Editor_System* y *Read_Common_Params* (incluidos en el paquete *Editor_Actions*) realizan la lectura, mientras que *Save_Editor_System* (del paquete *Editor_Actions*) y los diferentes *Print* incluidos en los paquetes *Mast_Editor* son los encargados de realizar la escritura. Estos procedimientos ya fueron comentados en los apartados 4.3.2 y 4.3.3.

5. Ejemplos de Modelado MAST

5.1. Introducción

En este capítulo comentaremos algunos ejemplos de sistemas de tiempo real que han sido modelados utilizando la metodología MAST. Estos ejemplos han sido incluidos a modo orientativo para ilustrar el proceso seguido en la elaboración del modelo MAST a partir de las características estructurales y temporales de los sistemas.

Para caracterizar estos sistemas utilizando el editor gráfico se han seguido los pasos ya explicados en el apartado 3.6, “Cómo realizar el modelado del sistema”.

A continuación dedicaremos los siguientes apartados a comentar el proceso seguido en el modelado de dos sistemas de tiempo real: el primero es un sistema monoprocesador, el segundo es un sistema con transacciones lineales.

5.2. Ejemplo de Sistema Monoprocesador: CASEVA

CASEVA [1] es un robot diseñado para la soldadura automática de uniones entre piezas que no tienen simetría axial. Este robot contiene un controlador empotrado que usa un computador (un HP 743rt) basado en el bus VME ejecutando el sistema operativo de tiempo real HP-RT. El software de la aplicación es concurrente y está escrito en el lenguaje Ada. Las principales características de las tareas realizadas se muestran en la figura 5.1, donde se especifican los plazos (*T*), tiempos de ejecución (*C*) y las prioridades (*Prio*).

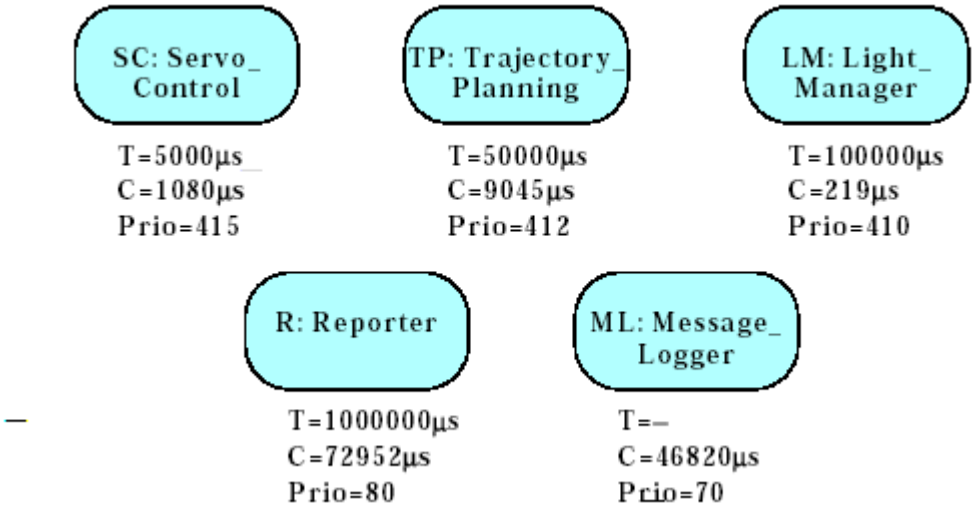

Figura 5.1: Características básicas de las tareas del controlador CASEVA

La comunicación y sincronización entre las diferentes tareas es asíncrona y se basa en recursos compartidos implementados utilizando objetos protegidos Ada. En este apartado se presenta una visión simplificada de los recursos compartidos y de las operaciones asociadas para no hacer la descripción demasiado extensa. La tabla I muestra las características de los objetos protegidos y operaciones simplificados.

Recurso Compartido	Operación	WCET (μs)	Utilizado por
Servo_Data	Read_New_Point New_Point	87 54	SC TP
Arm	Read_Axis_Positions Control_Servos	135 99	SC, R SC
Lights	Turn_On Turn_Off Time_Lights	74 71 119	TP TP LM
Alarms	Read_All Set	78 59	SC, TP, R SC, TP
Error_Log	Notify_Error Get_Error_From_Queue	85 79	TP ML

Tabla I: Características de los recursos y operaciones del controlador CASEVA

Para construir el modelo de este sistema monoprocesador primero se configuraron los elementos de la Capa de Recursos de Procesado y Planificadores. En este caso se trata de un procesador con un temporizador del tipo reloj despertador y un planificador con política de prioridades fijas (ver figura 5.2).

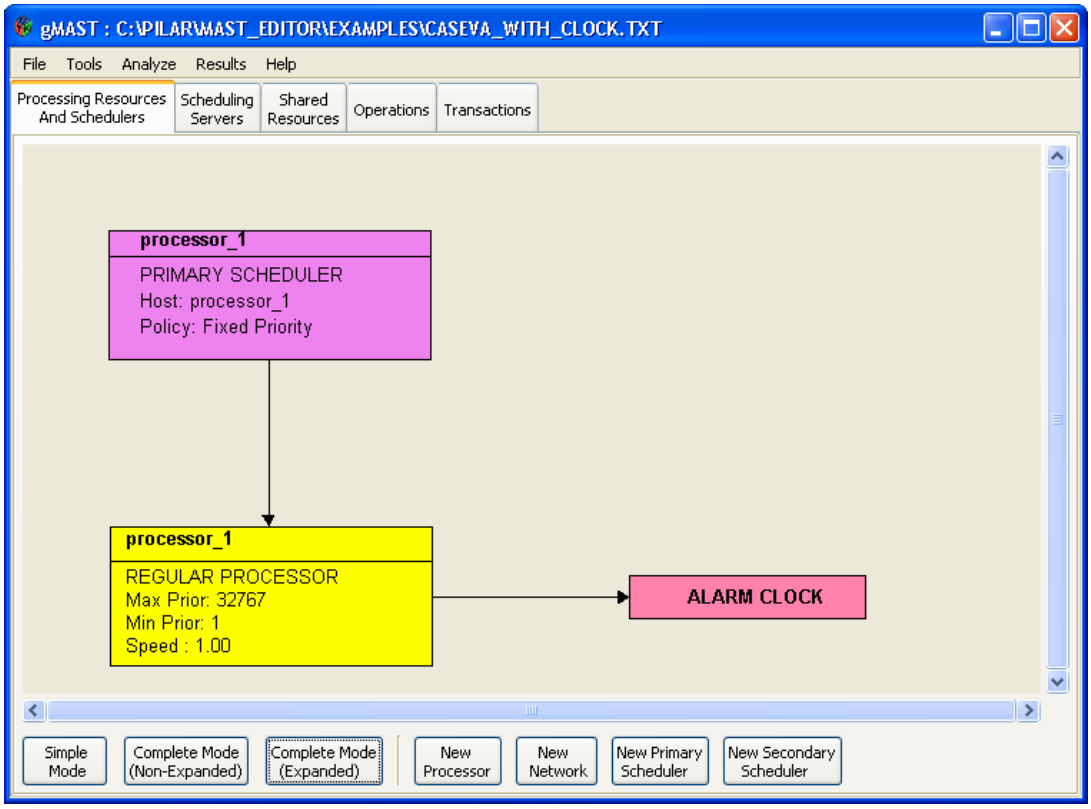

Figura 5.2: Diagrama de Recursos de Procesado y Planificadores del CASEVA

Seguidamente se configuraron los servidores de planificación incluidos en la figura 5.1 con sus correspondientes prioridades, como se muestra en la figura 5.3.

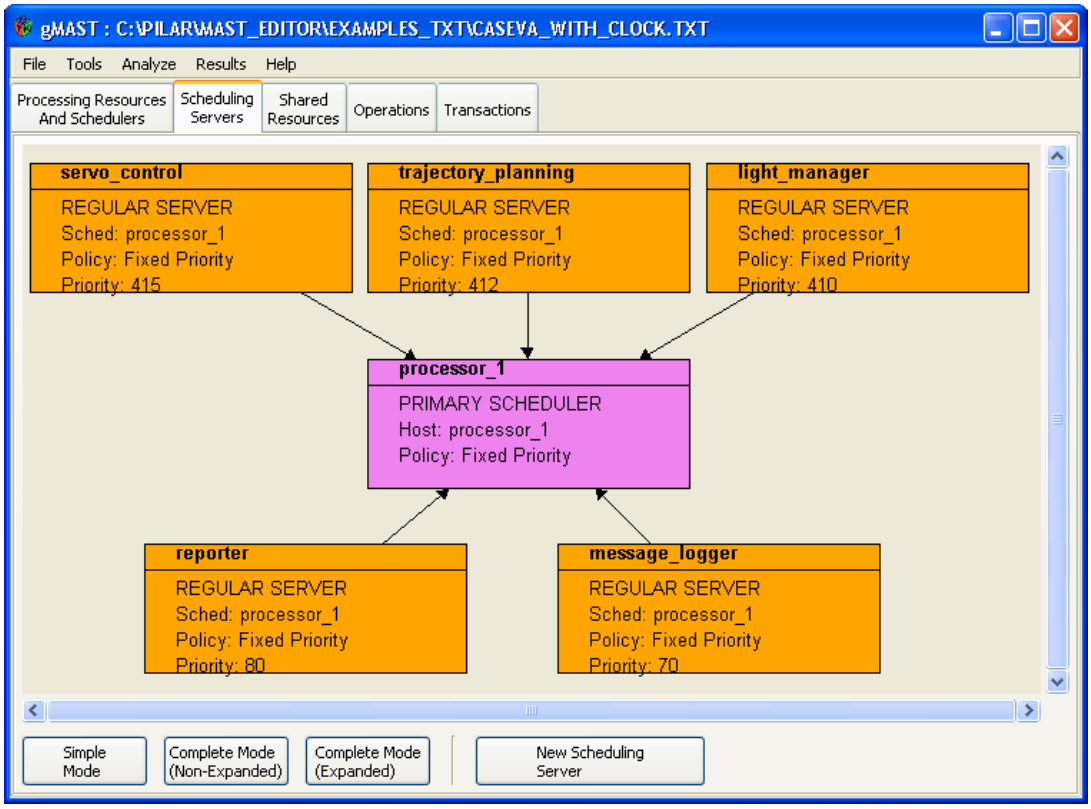

Figura 5.3: Diagrama de Servidores de Planificación del controlador CASEVA

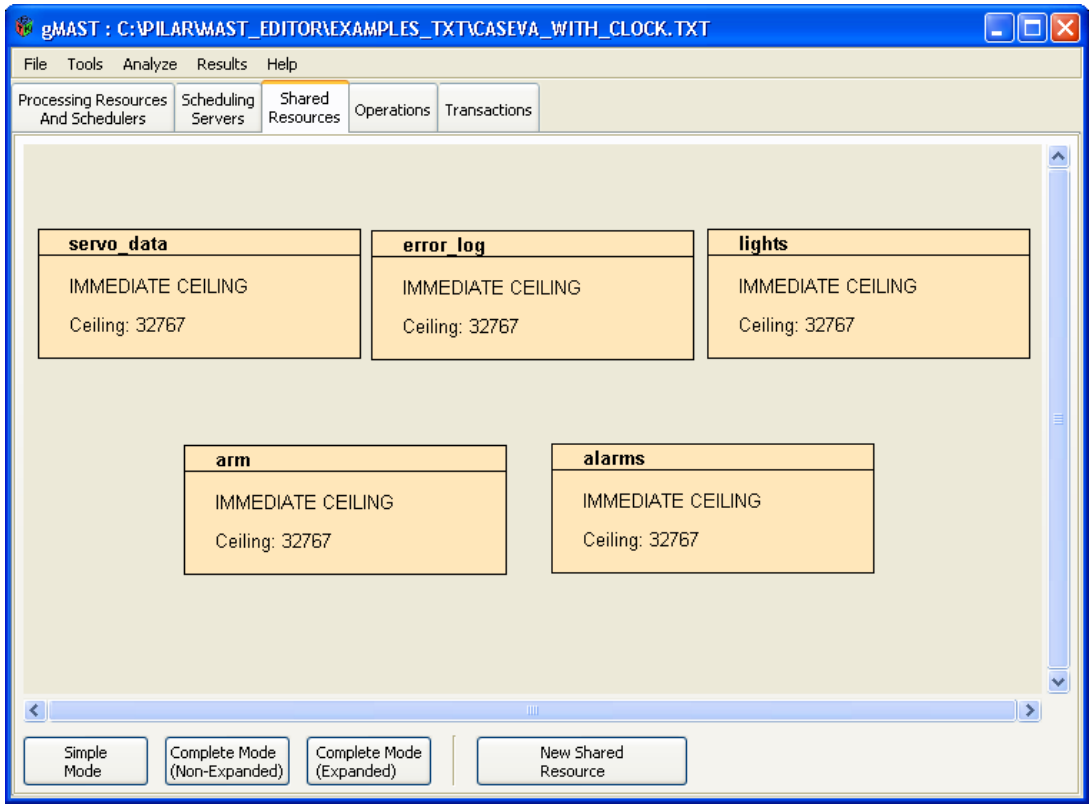

Figura 5.4: Diagrama de Recursos Compartidos del controlador CASEVA

A continuación se añadieron los recursos compartidos y las operaciones definidos en la tabla I en las correspondientes capas (ver figuras 5.4 y 5.5).

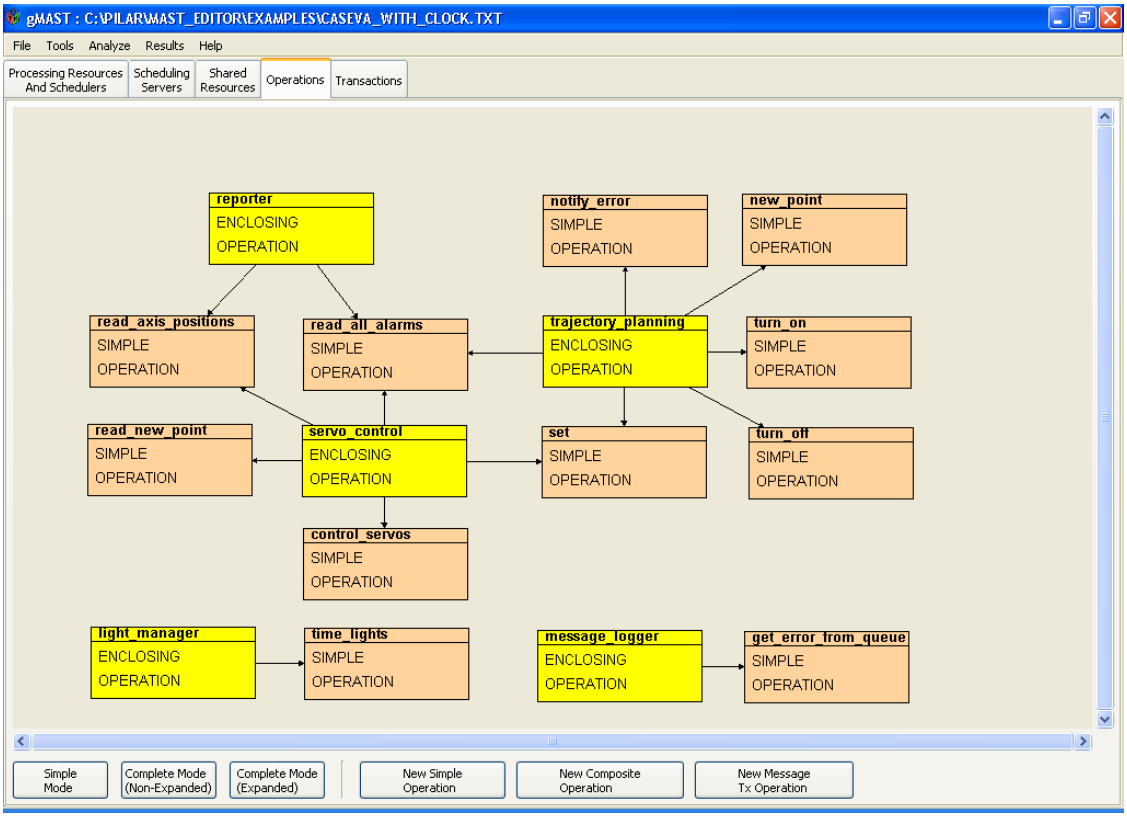

Figura 5.5: Diagrama de Operaciones del controlador CASEVA

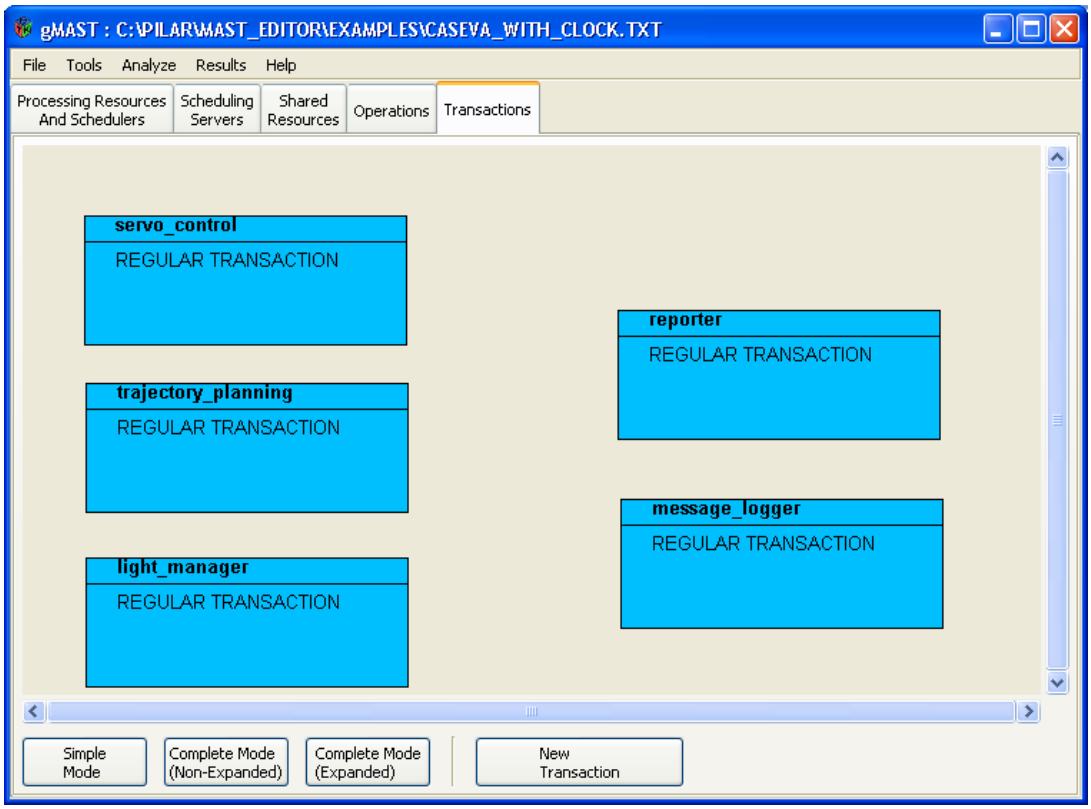

Figura 5.6: Diagrama de Transacciones del controlador CASEVA

Finalmente, se configurarían las cinco transacciones del sistema (figura 5.6). Todas ellas se componen de una sola actividad que ejecuta una operación en un servidor de planificación del procesador. Las actividades de las cuatro primeras transacciones (*servo_control, trajectory_ planning, light_manager, reporter)* están temporizadas y tienen impuestos plazos para generar los eventos de salida. El aspecto del grafo de estas transacciones se muestra en la figura 5.7 (al tener requerimientos temporales el evento de salida se pinta de color naranja).

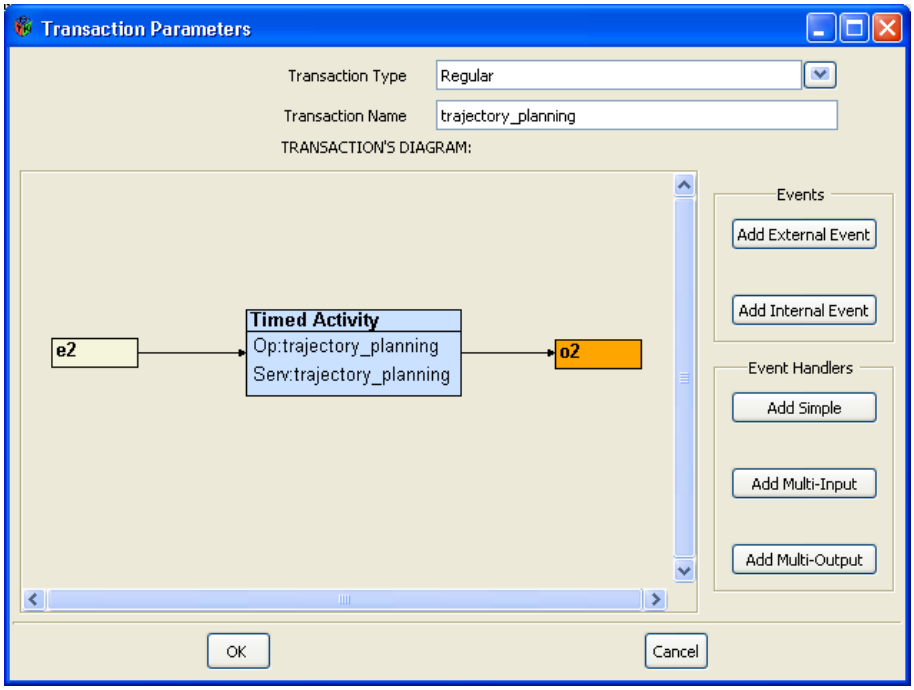

Figura 5.7: Diagrama de la Transacción *trajectory_planning* del CASEVA

La otra transacción (*message_logger)* carece de requerimientos temporales y su diagrama es el de la figura 5.8 (al no tener requerimientos temporales, el evento de salida se pinta de color gris).

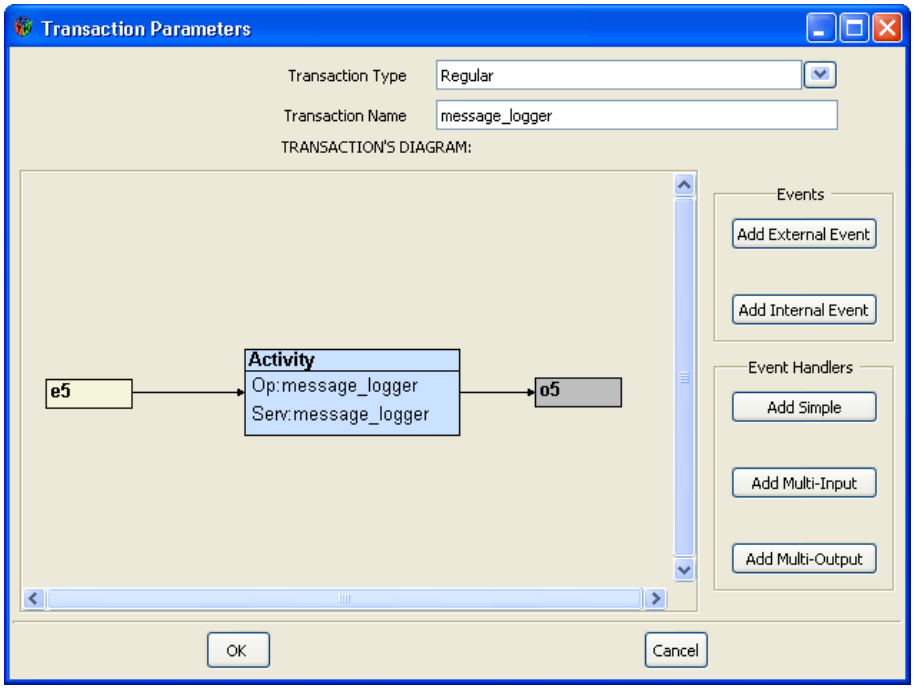

Figura 5.8: Diagrama de la Transacción *message_logger* del CASEVA

5.3. Ejemplo de Transacciones Lineales: RMT

El siguiente ejemplo es una simplificación del sistema de control de un robot teleoperado [1]. Se trata de un sistema distribuido con dos nodos especializados: un controlador de robot local y una estación de teleoperación remota, donde el operario manipula los controles y recoge la información del estado del sistema.

La figura 5.9 muestra un diagrama de la arquitectura del software. El sistema tiene tres transacciones, una de las cuales, llamada lazo de control principal (*Main_Control_Loop*), implica la ejecución en recursos de procesado diferentes y tiene un plazo global extremo a extremo (*end-to-end*).

La comunicación se realiza a través de una red Ethernet operando en modo maestro-esclavo para conseguir un comportamiento de tiempo real estricto.

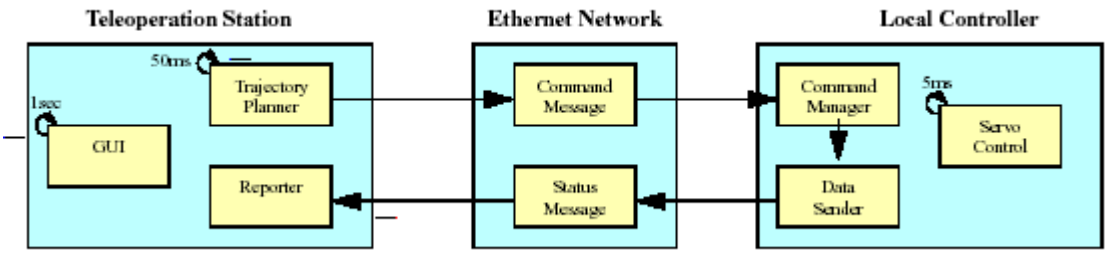

Figura 5.9: Arquitectura del controlador de robot teleoperado

Primeramente se configuran los recursos de procesado y planificadores. Por un lado, tenemos dos procesadores (*teleoperation_station* y *local_controller*), asociados a dos planificadores y con sendos temporizadores del tipo reloj despertador y, por otro lado, la red ethernet asociada a un planificador y sin ningún controlador (ver figura 5.10).

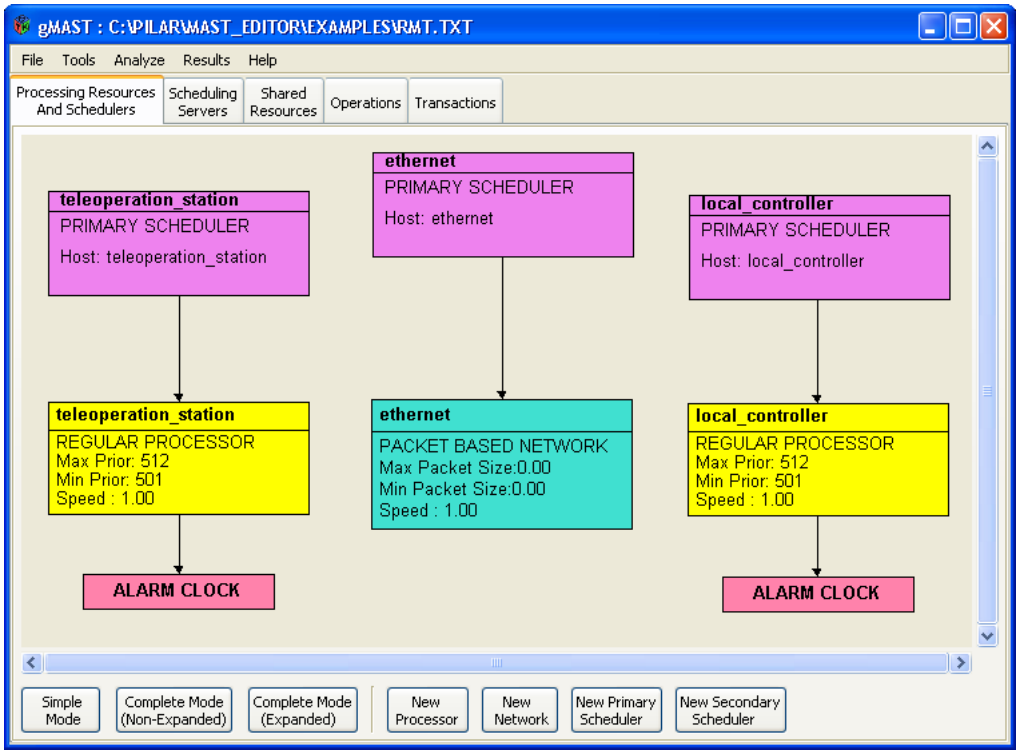

Figura 5.10: Diagrama de Recursos de Procesado y Planificadores del RMT

Dentro de la Capa de servidores de planificación se configuran los siguientes elementos: tres servidores asociados a cada uno de los planificadores de los procesadores y uno asociado al planificador de la red ethernet, como se muestra en la figura 5.11.

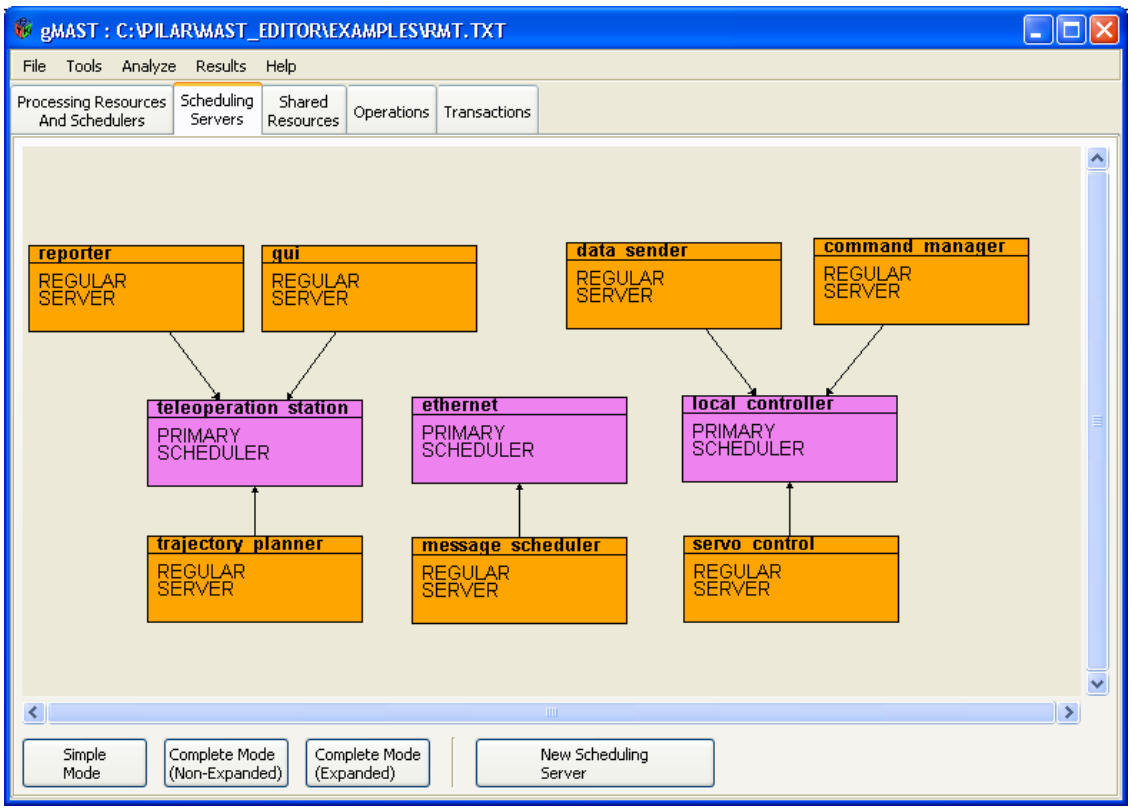

Figura 5.11: Diagrama de Servidores de Planificación del controlador RMT

A continuación se configuran los recursos compartidos que en este caso se trata de tres recursos de techo de prioridad: *status*, *commands* y *servo_data* (ver figura 5.12).

Una vez que se han definido los recursos compartidos, se añaden las operaciones utilizando las funcionalidades de la Capa de operaciones. En primer lugar se configuran las operaciones simples (*read_status, write_status, set_command, get_command, read_servos, write_servos, command_message* y *status_message*) y, posteriomente, se realiza la configuración de las operaciones englobantes (*command_manager, data_sender, servo_control, trajectory_planner, reporter* y *gui*) (ver figura 5.13).

Para finalizar el modelado se configurarían las transacciones del sistema. El aspecto del Diagrama de Transacciones se muestra en la figura 5.14. En esta ocasión existen tres transacciones lineales: *servo_control, gui* y *main_control_loop*. Las dos primeras constan de una sóla actividad temporizada con un evento de entrada y uno de salida (como se muestra en la figura 5.15), mientras que la última se compone de seis actividades que se ejecutan de manera secuencial en diferentes servidores de planificación (ver figuras 5.9 y figura 5.16).

En todas las transacciones se ha definido un plazo para la generación del evento de salida de la última actividad (por ello el último evento de cada transacción se dibuja con color naranja). Cada uno de estos plazos viene referido respecto al evento de entrada de la primera actividad de la transacción.

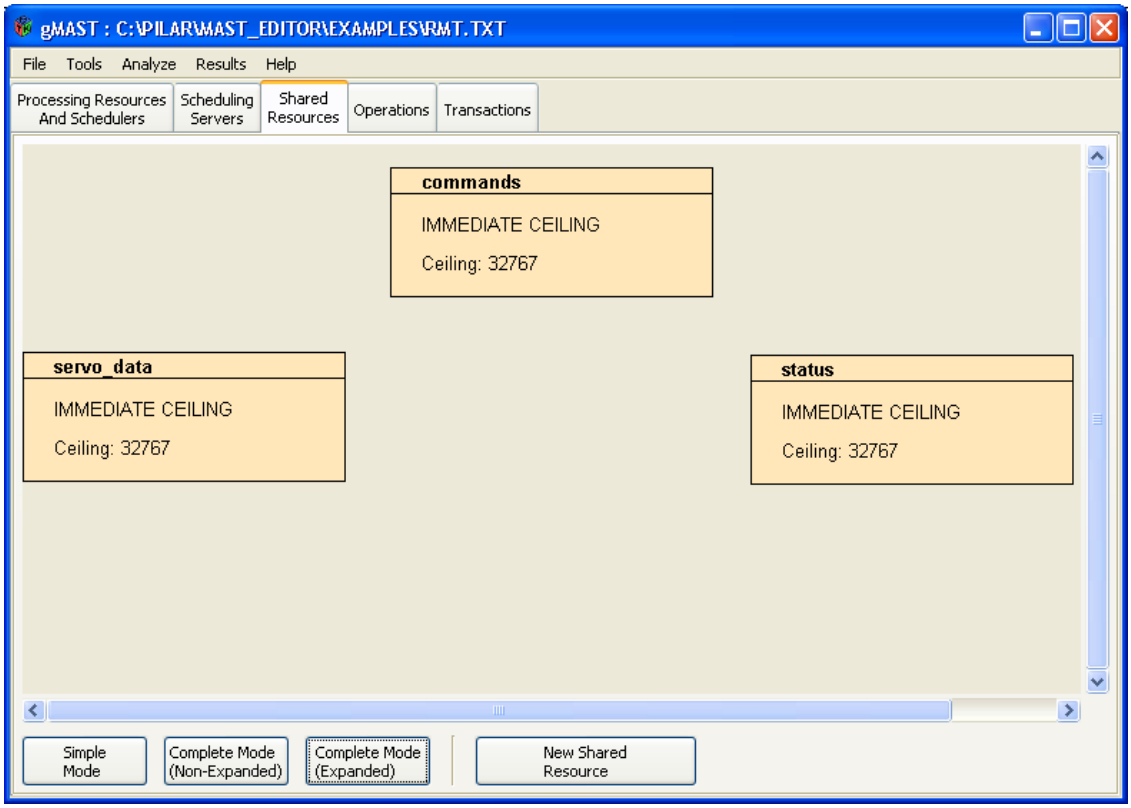

Figura 5.12: Diagrama de Recursos Compartidos del controlador RMT

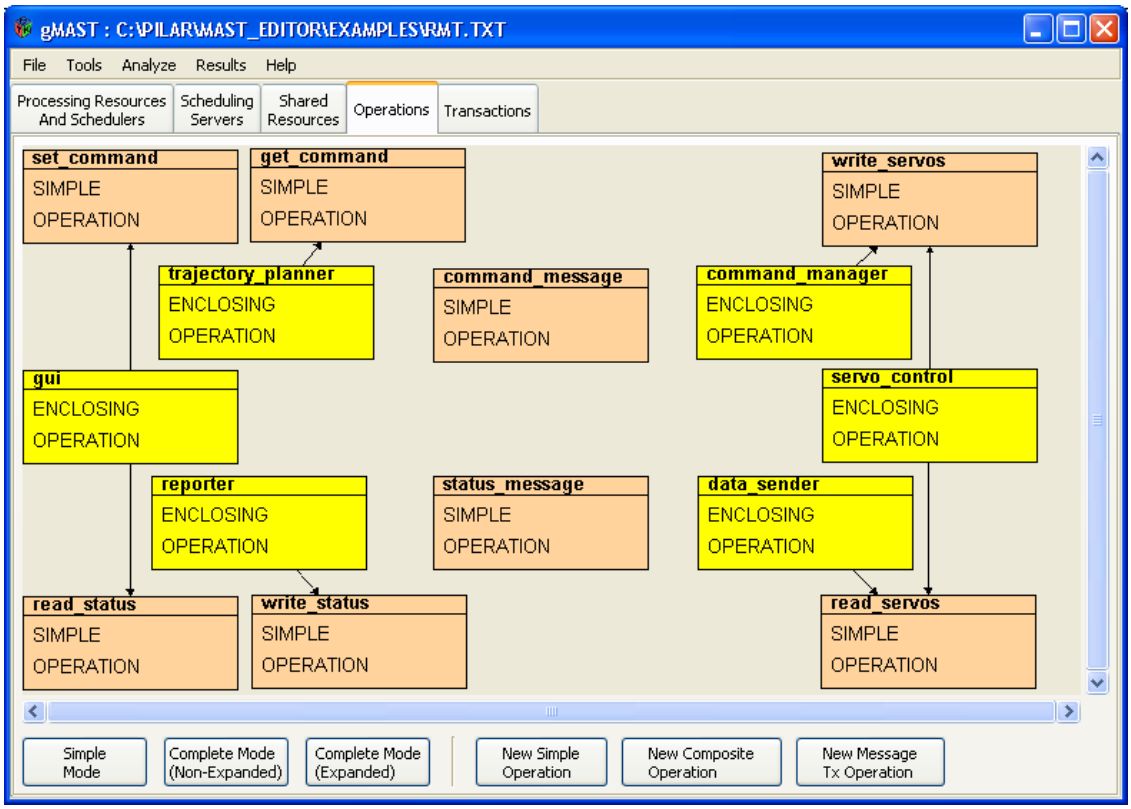

Figura 5.13: Diagrama de Operaciones del controlador RMT

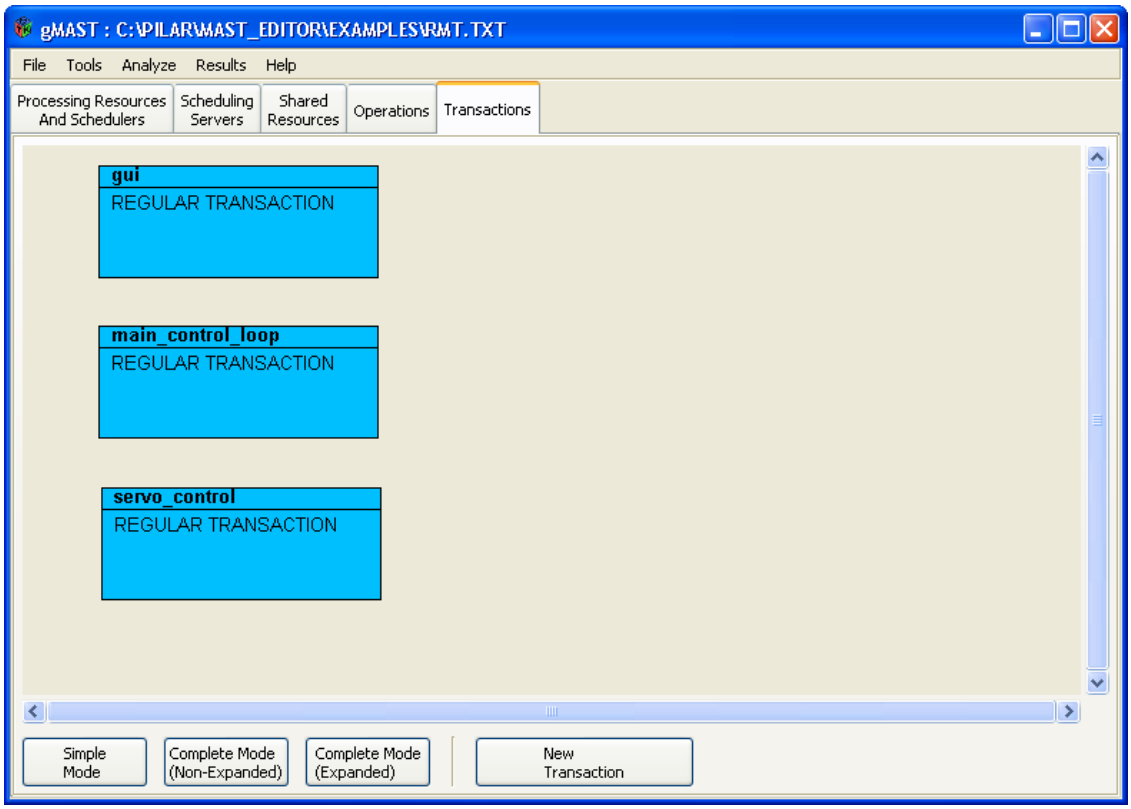

Figura 5.14: Diagrama de Transacciones del controlador RMT

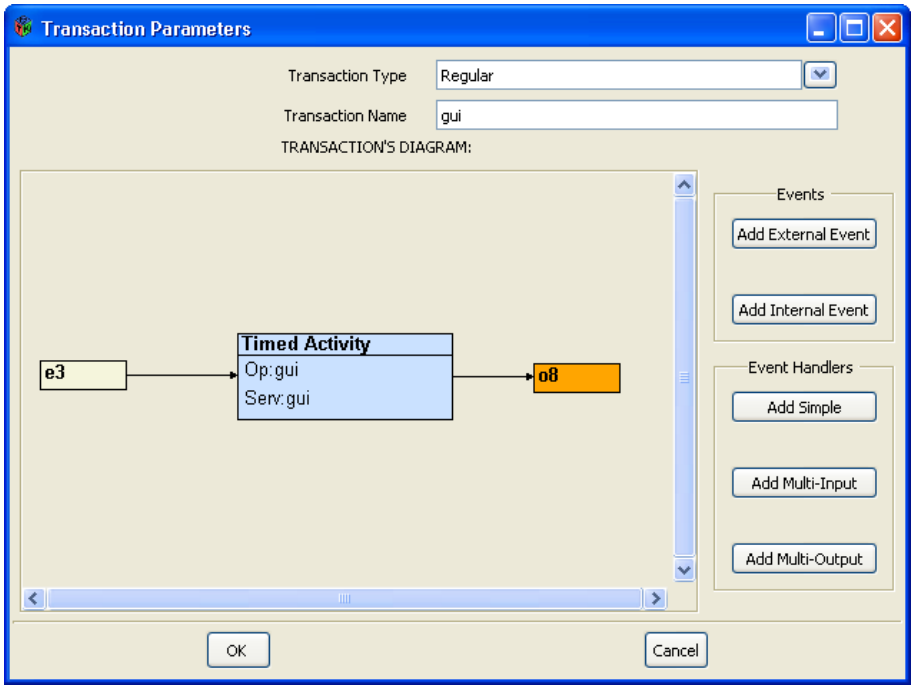

Figura 5.15: Diagrama de la Transacción *gui* del controlador RMT

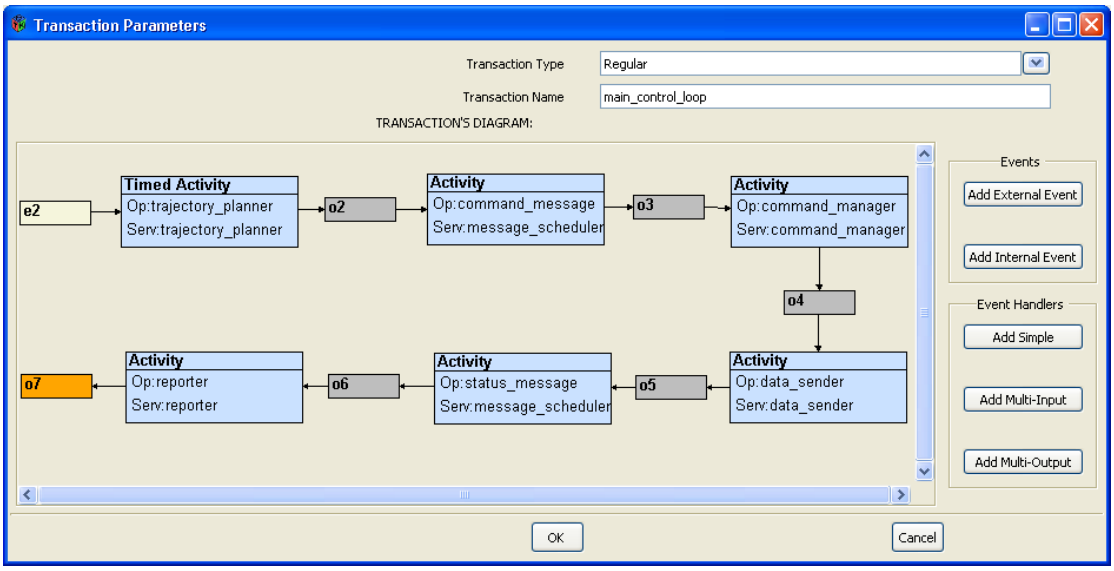

Figura 5.16: Diagrama de la Transacción *main_control_loop* del controlador RMT

6. Conclusiones y Líneas Futuras

En este capítulo se recogen una serie de conclusiones que se derivan del trabajo descrito en los capítulos anteriores. Además se expondrán algunas de las líneas futuras que se pueden seguir para ampliar y mejorar las funcionalidades de la interfaz de usuario desarrollada en este proyecto.

6.1. Conclusiones

A lo largo del trabajo hemos podido examinar una serie de decisiones de diseño e implementación de un editor gráfico integrado en el entorno MAST cuyo propósito es facilitar al usuario la confección del modelo MAST de un Sistema de Tiempo Real. A continuación expondremos cuáles son los principales logros del trabajo:

- Se han adquirido los conocimientos suficientes sobre las propiedades de los Sistemas de Tiempo Real y los parámetros que los caracterizan para poder comprender y seguir una de las metodologías utilizadas para la simulación y análisis de este tipo de sistemas como es la implementada en el entorno MAST.
- Se ha desarrollado una herramienta software capaz de representar gráficamente el modelo MAST por medio de una serie de diagramas en los que se pueden visualizar y consultar los elementos que lo componen y las relaciones existentes entre ellos. De esta manera el usuario de la aplicación obtiene una visión global de la estructura del modelo y se hace más fácil su creación o modificación.
- Con la nueva herramienta se consigue generar automáticamente la descripción ASCII del modelo MAST utilizando los valores de los parámetros introducidos por el usuario en las diferentes ventanas de la interfaz. De este modo ya no es necesario crear el archivo de texto manualmente y se evita el laborioso proceso de tener que escribir línea a línea cada uno de los atributos del modelo siguiendo el formato definido en la especificación MAST. Esta funcionalidad reduce el tiempo empleado en la configuración del modelo y se eliminan los problemas que pueden derivarse al confeccionar la descripción ASCII utilizando un formato incorrecto.
- Hemos implementado un editor gráfico que contiene una serie de ventanas y menús mediante los cuales se pueden configurar de forma rápida y sencilla todos los atributos de los diferentes elementos que conforman el modelo MAST de un Sistema de Tiempo Real. Se facilita en gran medida la creación del modelo ya que el usuario puede definir los atributos de cada elemento casi intuitivamente por medio de las ventanas de configuración, sin tener que conocer en detalle cúales son los atributos definidos para cada tipo definido en la especificación MAST y los valores que éstos toman por defecto.

- La aplicación es capaz de realizar la creación, lectura y escritura de archivos destinados al almacenamiento de los datos referentes al modelo MAST y a su visualización en la pantalla. Cada modelo lleva asociados dos archivos que son generados automáticamente por la aplicación: el primero contiene la descripción MAST del modelo y es utilizado por las herramientas de análisis para determinar la planificabilidad del sistema modelado, el segundo guarda los parámetros referentes a la representación gráfica y sólo es utilizado por el editor para elaborar los distintos diagramas incluídos en la interfaz.
- La utilización del editor gráfico constituye una opción alternativa para aquellos usuarios que pretenden hacer uso de las herramientas de análisis de MAST pero no están interesados en recurrir a la descripción UML.
- El software desarrollado para implementar la interfaz está compuesto por una serie de paquetes que han sido programados utilizando el lenguaje Ada 95. Según la función que desempeñan los paquetes dentro de la aplicación y el modo en que han sido programados se han definido tres tipos:
    - **Paquetes Glade:** Son paquetes creados automáticamente utilizando el constructor de interfaces de usuario para gtk+ Glade. La función de estos paquetes es determinar la estructura y funcionamiento de las ventanas y diálogos que componen la interfaz y las relaciones que existen entre ellas. El código original generado por defecto con la herramienta Glade se ha modificado posteriormente para hacer que el diseño y comportamiento de las ventanas sea el deseado.
    - **Paquetes Mast_Editor:** Estos paquetes han sido creados de forma manual. Contienen el código utilizado para mostrar o recoger los valores de los atributos del modelo de las ventanas y para dibujar los elementos en los diagramas del editor. El diseño y programación de estos paquetes se ha hecho siguiendo una estructura jerárquica y utlizando técnicas de programación orientada a objetos.
    - **Paquetes de Manejo de Archivos y de Control:** Este tipo de paquetes se elaboraron manualmente y su principal función es la de realizar la creación, lectura y escritura de los archivos manejados por el editor, control de cambios y control de los asistentes de la aplicación.
- Como el entorno MAST todavía está en desarrollo es muy probable que en el futuro surjan nuevos requerimientos y que sea preciso añadir nuevas funcionalidades al editor, por eso éste ha sido diseñado de forma que sea relativamente fácil implementar esos cambios en él.
- Se ha logrado desarrollar una aplicación software portable tanto para plataformas Windows como Linux de modo que cualquier usuario interesado en utilizar el editor va a poder ejecutar el programa en ambos sistemas operativos.
- Se ha verificado el correcto funcionamiento del editor gráfico realizando el modelado de varios Sistemas de Tiempo Real. Para ello se comprobó que los archivos de descripción MAST generados con el editor tienen el formato adecuado para utilizarlos posteriormente como entrada de las herramientas de

análisis y así poder realizar una simulación apropiada de la planificabilidad y del comportamiento temporal de los sistemas.

- Se han proporcionado varios ejemplos ilustrativos que pueden servir de ayuda o como referencia a la hora de crear y configurar el modelo de cualquier Sistema de Tiempo Real. Los modelos de los ejemplos corresponden a sistemas reales que se han simplificado para no hacer el proceso de modelado demasiado laborioso, si bien la aplicación puede realizar la simulación de sistemas más complejos.

6.2. Líneas Futuras

Este apartado incluye una serie de posibles opciones que pueden seguirse en el futuro para extender o mejorar la aplicación desarrollada en este trabajo.

El trabajo futuro más importante será integrar el editor con la interfaz de configuración de las herramientas de análisis y con el visor de resultados. Seguidamente se muestran más detalladamente algunas de las líneas a seguir en el futuro:

- Dentro del menú de la ventana principal, falta implementar la herramienta de ayuda del editor invocada desde el menú *Help* que sirva al usuario como asistente en el manejo de la aplicación.
- Se podrían añadir nuevas opciones al menú emergente para que se mostrasen en pantalla otros diagramas auxiliares, por ejemplo:
    - Diagrama de Transacciones asociadas a una operación (invocado desde el menú emergente de la Capa de Operaciones).
- Una posible ampliación de este trabajo es añadir a la interfaz algunos diagramas integrados en nuevas "*capas*" con el propósito de ofrecer al usuario una vista más completa de las relaciones existentes entre los distintos elementos del modelo MAST. Un ejemplo de capa auxiliar podría ser el siguiente:
    - Capa de Mapeado de Actividades: en esta capa se visualizarían las actividades realizadas en cada transacción usando un diagrama de flujo que mostrase la secuencia de ejecución de las tareas ordenadas según el procesador que las realiza.
- Si en un futuro se realizasen extensiones de la especificación MAST, se podría modificar el código existente o programar rutinas nuevas que doten al editor de las funcionalidades adicionales motivadas por dichos cambios en la especificación.
- En caso de querer desarrollar aplicaciones gráficas de similares características, puede ser interesante seguir el proceso realizado en el diseño e implementación de esta aplicación e incluso puede reutilizarse parte del código elaborado en el presente trabajo. Si bien esto último no parece muy probable para aplicaciones fuera del entorno MAST ya que la mayoría de funciones y procedimientos programados hacen referencia a procedimientos de esa librería.

Bibliografía y referencias

[1] Drake, J.M. González Harbour, M. Gutierrez, J.J. and Palencia, J.C. "Description of the MAST Model". http://mast.unican.es/

[2] Drake, J.M. González Harbour, M. Gutierrez, J.J. and Palencia, J.C. "UML-MAST Metamodel, specification, example of application and source code". http://mast.unican.es/umlmast

[3] English, J. "Ada 95: The Craft of Object-Oriented Programming". Prentice Hall, 1997

[4] Feldman, M. Koffman, E. "Ada 95: Problem Solving and Program Design". Addison-Wesley, 2nd Edition, 1996

[5] González Harbour, M. Gutiérrez, J.J. Palencia, J.C. and Drake, J.M. "MAST: Modeling and Analysis Suite for Real-Time Applications". Proceedings of the Euromicro Conference on Real-Time Systems, Delft, The Netherlands, June 2001

[6] González Harbour, M. Klein, M.H. and Lehoczky, J.P. "Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority". Proceedings of the IEEE Real-Time Systems Symposium, December 1991, pp. 116-128

[7] González Harbour, M. Medina, J.L. Gutiérrez, J.J. Palencia, J.C. and Drake, J.M. "MAST: An Open Environment for Modeling, Analysis and Design of Real-Time Systems". http://mast.unican.es/

[8] Gutiérrez García, J.J. Palencia Gutiérrez, J.C. and González Harbour, M. "Schedulability Analysis of Distributed Hard Real-Time Systems with Multiple-Event Synchronization". Euromicro Conference on Real-Time Systems, Stockholm, Sweden, 2000.

[9] Gutiérrez García, J.J and González Harbour, M. "Optimized Priority Assignment for Tasks and Messages in Distributed Real-Time Systems". Proceedings of 3rd Workshop on Parallel and Distributed Real-Time Systems, Santa Barbara, California, pp. 124-132, 1995.

[10] http://www.ics.uci.edu/~arcadia/Aflex-Ayacc/aflex-ayacc.html

[11] Joseph, M. and Pandya, P. "Finding Response Times in a Real-Time System," BCS Computer Journal, Vol. 29, no.5, October 1986, pp 390-395.

[12] Lehoczky, J.P. "Fixed Priority Scheduling of Periodic Tasks Sets with Arbitrary Deadlines". IEEE Real-Time Systems Symposium, 1990.

[13] Medina, J.L. González Harbour, M. and Drake, J.M. "MAST Real-Time View: A Graphic UML Tool for Modeling Object-Oriented Real-Time Systems" RTSS'01, London, December, 2001.

[14] Palencia, J.C. and González Harbour, M. “Schedulability Analysis for Tasks with Static and Dynamic Offsets”. Proc. of the 19th IEEE Real-Time Systems Symposium, 1998.

[15] Palencia, J.C. and González Harbour, M. “Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems”. Proceedings of the 20th IEEE Real-Time Systems Symposium, 1999.

[16] Pennington Havoc, “Gtk+/Gnome Application Development”. http:// www.opencontent.org

[17] Skansholm, J. “Ada 95: From the Beginning”. Adisson-Wesley, 3rd Edition, 1997

[18] Silberschatz, A. Galvin, P. “Operating System Concepts” Addison-Wesley, 1998

[19] “The GtkAda Reference Manual” . http://libre.act-europe.fr/GtkAda

[20] “The GtkAda User Guide” . http://libre.act-europe.fr/GtkAda

[21] Tindell, K and Clark, J. “Holistic Schedulability Analysis for Distributed Hard Real-Time Systems”. Microprocessing & Microprogramming, Vol. 50, Nos.2-3, pp. 117-134, 1994.

[22] Del Río Trueba, María del Pilar y González Harbour, Michael “Diseño de un editor gráfico para el modelado y análisis de Sistemas de Tiempo Real”. Proyecto Fin de Carrera, Ingeniero de Telecomunicación. Universidad de Cantabria, 2003.

Indice de figuras