UML-Mast: Una Metodología de Análisis y Diseño de Tiempo Real de Sistemas Orientados a Objetos Descritos con UML[1]

Julio Luis Medina Pasaje, José M. Drake Moyano, Michael González Harbour

Av.Los Castros s/n 39006 Santander - España
{medinajl, drakej, mgh}@unican.es

Resumen. Se describe una herramienta y una metodología que facilitan el análisis y diseño de tiempo real de sistemas orientados a objetos expresados en UML. Partiendo de la descripción lógica del sistema en UML se modela su comportamiento de tiempo real a través de una nueva vista. Se emplean para ello los componentes conceptuales de modelado que define el metamodelo UML-Mast. El metamodelo describe tanto la vista de tiempo real como la naturaleza y contenidos de todos los componentes de modelado y sus relaciones. Esta vista permite procesar el modelo facilitando la invocación, desde la propia herramienta CASE en que se está desarrollando el sistema, de un conjunto de herramientas automáticas relativas al análisis de planificabilidad, asignación óptima de prioridades y cálculo de holguras entre otros. Los resultados obtenido se incorporan automáticamente al modelo UML del sistema. Se ofrece una versión de la herramienta UML_Mast, implementada para su utilización desde Rational Rose 2000e.

1. Introducción.

UML-Mast es un entorno de modelado que aporta una nueva vista (Mast_RT_View) para la descripción del comportamiento de tiempo real del sistema que se modela. A través de ella el diseñador puede construir gradualmente el modelo de tiempo real de forma paralela al desarrollo de su modelo lógico. Este modelo puede ser analizado por un conjunto de herramientas automáticas relativas al análisis de planificabilidad, estimación de holguras, asignación óptima de prioridades, detección de bloqueos, animación etc. Con su uso el diseñador puede tener en consideración desde las primeras fases del diseño estimaciones del nivel de cumplimiento de las prestaciones de tiempo real. Tanto los componentes conceptuales de modelado como las herramientas de análisis de UML-Mast son proporcionados por el entorno Mast (Modelling and Analysis Suite for Real Time Applications) [1,2]. Mast es un entorno abierto basado en una descripción textual (Mast-File) del modelo del sistema de tiempo real.

[1] Este trabajo está financiado en el proyecto: "Diseño Integrado de Sistemas de Tiempo Real Embarcados", Plan Nacional de Investigación (TIC99-1043-C03-03)

UML-Mast proporciona mecanismos para modularizar y parametrizar secciones del modelo de tiempo real, a imagen de la vista lógica del sistema, facilitando diversos niveles de modelado, así se dispone tanto de un modelo de tiempo real del sistema completo, como del modelo de cada clase lógica o incluso del modelo de cada método de su interfaz. Con ello se consigue que los modelos de tiempo real sean reusables y que puedan constituir parte de la especificación de componentes de tiempo real. Tanto la Mast RT View como los componentes de modelado están formulados y definidos en UML mediante un metamodelo.

La utilización de las herramientas que proporciona UML-Mast se facilita mediante su incorporación a la herramienta CASE-UML Rational ROSE'2000e a través de un framework y diversos "scripts" que se han desarrollado e integrado al efecto como "personalizaciones" de los menús de Rose.

La metodología que se presenta sigue el paradigma del procesado de modelos recogido en la reciente "Response to the OMG RFP for Schedulability, Performance and Time" [3,4], cuya finalidad es crear un estándar de modelado cualitativo y cuantitativo de los requerimientos y de las prestaciones de tiempo real de sistemas diseñados con metodologías orientadas a objetos que servirá de base para el intercambio de sistemas entre diseñadores y la interoperatividad de las herramientas.

A lo largo de esta comunicación se ilustra cada sección del modelo de la vista de tiempo real Mast_RT_View mediante un ejemplo de su aplicación para el análisis de un sistema distribuido de control de un robot teleoperado [5].

2. Modelado de Tiempo Real con UML_Mast

En el entorno UML-Mast, los componentes UML que constituyen el modelo de tiempo real de un sistema se presentan mediante diagramas de clases y diagramas de actividad, que en su conjunto representan el comportamiento dinámico desde el punto de vista del análisis de tiempo real de los componentes hardware y software del sistema que se modela. Los tipos de componentes que constituyen la vista de tiempo real y las relaciones que se pueden establecer entre ellos, se definen a través del metamodelo "UML_Mast" [6].

2.1 Mast_RT_View

La Mast RT View se compone de tres secciones complementarias, cada una de ellas describe un aspecto específico del modelo de tiempo real.

- **Modelo de la plataforma:** Modela la capacidad de procesamiento y las restricciones operativas de los recursos de procesamiento hardware y software que constituyen la plataforma sobre la que se ejecuta el sistema. Estos recursos son: procesadores, threads, coprocesadores, equipos hardware específicos, redes de comunicación, etc. que tienen en común ser los agentes que ejecutan las actividades del sistema.
- **Modelo de los componentes lógicos:** Modela la cantidad de procesado que requiere la ejecución de las operaciones funcionales definidas en los componentes

que se definen en el diseño lógico del sistema. Estos son los métodos, procedimientos y funciones definidos en las clases, las primitivas de sincronización de threads, procesos de comunicación por las redes, operaciones que realizan los dispositivos hardware, etc. En esta sección del modelo se declaran los recursos que necesita cada operación para llevarse a cabo, en especial aquellos, que por ser requeridos por varias operaciones concurrentes en régimen de exclusión mutua, pueden ser causa de retrasos en la ejecución de las operaciones.

– **Escenarios de tiempo real:** Modelan las diferentes configuraciones hardware y software que puede alcanzar el sistema y en las que se establecen requerimientos de tiempo real. Cada escenario se modela como un conjunto de transacciones que describen las secuencias de eventos y actividades que deben ser analizadas para que se satisfagan los requerimientos de tiempo real establecidos en ellas. Cada transacción es una descripción no iterativa de las secuencias de actividades y eventos que se desencadenan como respuesta a un patrón de eventos autónomos (procedentes del entorno exterior al sistema, de timers, de relojes, de dispositivos hardware integrados, etc.) y de los requerimientos temporales que se imponen sobre ellas para garantizar la evolución temporal que se requiere. El conjunto de transacciones de un escenario constituye la carga del sistema en esa configuración y afecta al análisis de cada una de ellas.

2.2 Vista de Tiempo Real en la Herramienta Case UML

Para utilizar el framework que se ha incorporado en Rose 2000e, las secciones de la vista de tiempo real deben ubicarse en la estructura de paquetes predefinida que se muestra en la figura 1. Esta nomenclatura ha de seguirse estrictamente a fin de que las herramientas automáticas puedan localizar el modelo.

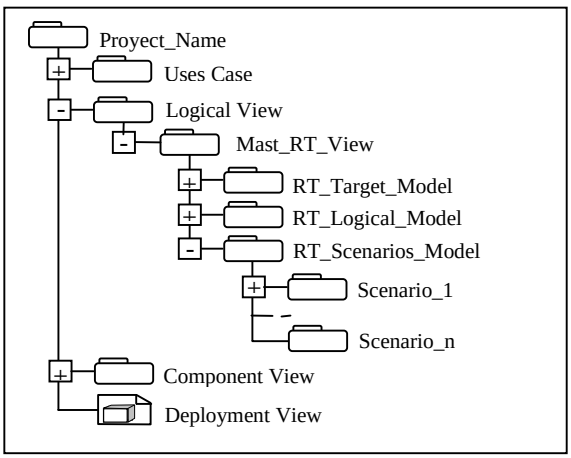

Fig 1. La Mast_RT_View en el Browser de Rational Rose

3. Modelado de Plataformas

El modelo de la plataforma describe la capacidad de procesamiento de los recursos hardware y software que ejecutan las actividades que constituyen el sistema que se modela. La capacidad de procesamiento disponible resulta de la diferencia entre la capacidad de procesamiento que proporcionan los procesadores y la capacidad de procesamiento consumida por las tareas de background que requieren los recursos software que gestionan y planifican la ejecución de las actividades de la aplicación (threads, planificadores, drivers, etc.).

3.1 Extracto del Metamodelo de Plataformas

En la figura 2 se muestran las clases abstractas básicas del metamodelo que se utilizan para describir el modelo de la plataforma:

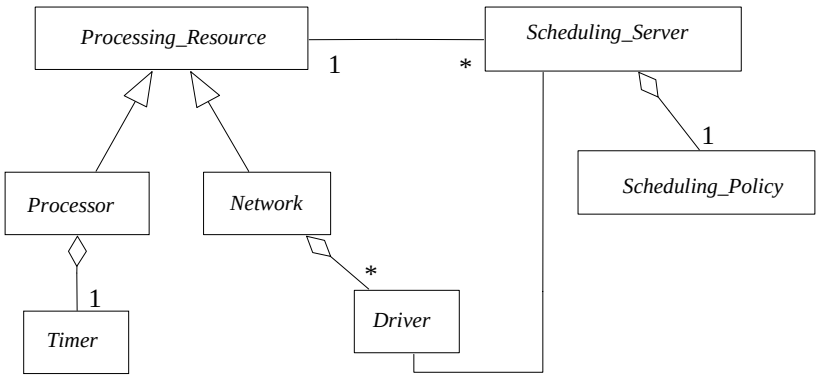

Fig. 2. Componentes de alto nivel del metamodelo de la plataforma

- **Processing_Resources:** se utiliza para modelar procesadores, equipos, redes de comunicación, etc. que ejecutan las operaciones requeridas en el sistema. Se han definido dos clases especializadas, los componentes de tipo *Processor* que representan procesadores, coprocesadores o equipos embarcados que ejecutan actividades formuladas como código de programa, y los componentes de tipo *Network* que representan redes de comunicación cuya actividad consiste en transferir información entre procesadores. Los componentes de tipo *Timer* y *Driver* modelan la sobrecarga que introducen las tareas de background que deben ejecutar los procesadores para la gestión de los timer hardware o los drivers de comunicaciones.
- **Scheduling_Server:** se utiliza para modelar procesos, threads o tareas dentro de las que se planifican las actividades del sistema que se modela. Cada Scheduling Server tiene asociada una política de planificación (Scheduling_Policy).

En la versión actual se han definido modelos concretos (no abstractos) de procesadores, redes, timers, drivers, políticas de planificación, etc. y para cada uno de ellos se han establecido los atributos que deben proporcionarse para que quede especificado el comportamiento de tiempo real de la plataforma modelada.

3.2 Ejemplo de Modelo de Plataformas

Se presenta un ejemplo de aplicación de UML-Mast a un sistema distribuido de control teleoperado para un robot. El sistema está constituido por dos procesadores comunicados a través de un bus CAN. El primero (Station) es una estación de teleoperación basada en una GUI que opera sobre windows NT, desde la que el operador monitoriza y controla los movimientos del robot. El segundo (Controller) es un microprocesador embarcado con nucleo de tiempo real mínimo que constituye el controlador de los servos y sensores del robot.

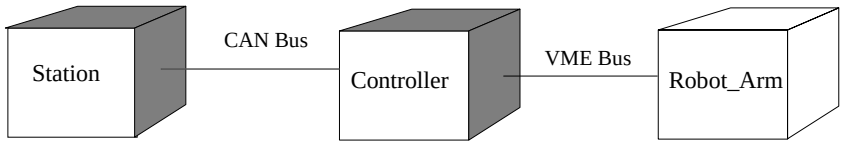

Fig.3. Diagrama de Despliegue del Ejemplo

La arquitectura software del sistema se encuentra en el diagrama de clases que se muestra en la figura 4. y se describe brevemente a continuación. El software del Controller está constituido por tres clases activas y una clase pasiva a través de la que se comunican. La clase Servo_Controller implementa el algorítmo de control de los servos y la monitorización de los sensores. Es una tarea periódica disparada por el timer. La tarea Reporter es también una tarea periódica que transfiere el status del robot a la estación de teleoperación. La tarea Command_ Manager es una tarea disparada por la llegada de un mensaje a través del bus CAN. Su función es descomponer los comando de alto nivel en secuencias de consignas para los servos. Estas tres tareas son asíncronas entre sí y se comunican a través del objeto protegido Servos_Data. El software de la estación es típico de una aplicación GUI. La tarea Command_Interpreter gestiona los eventos que el operador introduce sobre la GUI, transforma los eventos en comandos que transfiere hacia el Controller. La tarea Display_Refresher actualiza los datos de la GUI de acuerdo con los mensajes de status que recibe de Reporter. El acceso a la clase pasiva Display_Data se realiza con exclusión mutua a través de un objeto protegido.

Para ilustrar el modelado de las plataformas del sistema describimos el modelo de dos *Processing_Resource*, el procesador **Controller** y la red de comunicaciones **BusCAN**.

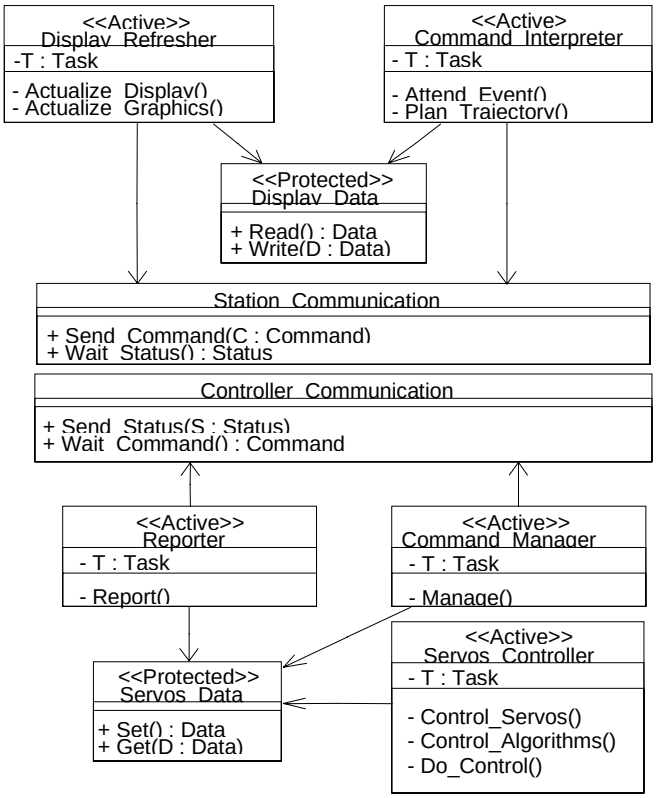

Fig. 4. Arquitectura Software del ejemplo "Robot Teleoperado"

El procesador Controller dispone de un timer de tipo Ticker para temporizar las tareas periódicas de su software. Este componente del modelo describe el overhead que representa para el procesador las tareas de gestión del timer. Es un procesador embarcado que ejecuta una aplicación desarrollada en ADA sobre el núcleo de tiempo real mínimo Marte_OS[7]. El comportamiento de tiempo real del procesador se modela con el componente Mast Controller del tipo Fixed_Priority_Processor. El Intervalo de prioridades que se utiliza para planificar los threads de la aplicación va de 1 a 30. El nivel de prioridad que se asigna a los manejadores de interrupción hardware es 31. El tiempo estimado de cambio de contexto entre threads de la aplicación es de 5 μs. y el que se da entre threads de la aplicación y rutinas de interrupción de 2.5 μs. El procesador dispone de un timer de tipo ticker para la temporización de las tareas periodicas, este tiene un periodo de activación de 1 ms y su gestión requiere al procesador un tiempo de overhead inferior a 7 μs. Las medidas de tiempos de ejecución de las operaciones de la aplicación se han realizado con el PC del procesador Station (Speed_Factor=1.0). El procesador Controller, 4 veces más lento, tiene un factor de velocidad 0.25. En la figura 5 se muestra el modelo del procesador Controller y de los threads que introducen cada uno de los objetos activos. Las actividades que se ejecutan en este procesador se planifican en cuatro threads.

Cada uno de ellos se modela mediante un componente Mast del tipo FP_Sched_Server y su respectiva política de planificación.

Servos_Controller_Task: Modela el thread que introduce la tarea Ada del objeto activo Servos_Controller. Es una tarea periodica de 5.0E-3 segundos de periodo. Su prioridad es 30.

Reporter_Task: Modela el threads que introduce la tarea Ada del objeto activo Reporter. Es una tarea periódica de 0.1 segundos de periodo. Su prioridad es 24.

Command_Manager_Task: Modela el threads que introduce la tarea Ada del objeto activo Command_Manager. Es una tarea gobernada por evento. Se ejecuta cada vez que se recibe un mensaje de command por el bus CAN. Su prioridad es 16.

Controller_Comm_Task: Modela el thread con que se ejecuta la rutina de interrupción que atiende el driver del bus CAN. Es una tarea gobernada por evento y se ejecuta cada vez que se produce una interrupción hardware en el driver del bus CAN. Al ser una rutina de interrupción hardware se le asigna una prioridad 31.

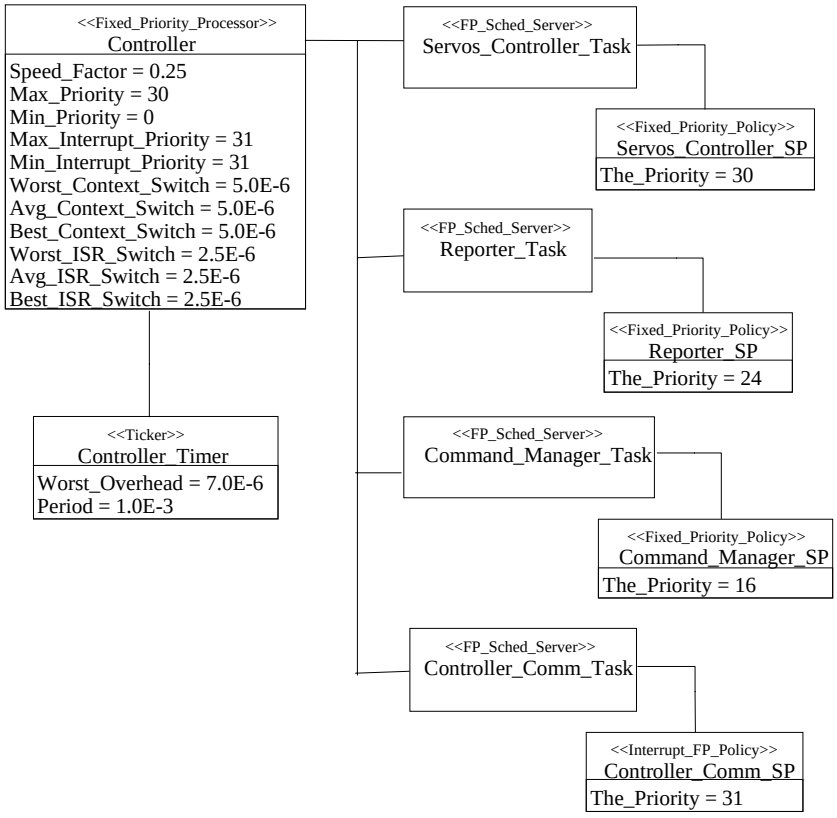

Fig. 5. Modelo del procesador Controller

En la figura 6 se muestra el modelo del bus CAN. El componente Can_Bus es el canal de comunicación que comunica el procesador Station y el procesador

Controller. Es un canal de tipo half_duplex, esto es con capacidad de transferir mensajes en ambos sentidos, pero sin que puedan coincidir en el tiempo. Es un bus orientado a paquetes y cada paquete puede estar compuesto de 1 a 8 bytes. La velocidad de transferencia del bus es inicialmente de 100 Kbytes/s.

La transferencia de los mensajes es priorizada, esto es, no se transfiere ningún paquete de un mensaje de una prioridad dada, si aún quedan pendientes de transferencia paquetes de un mensaje de mayor prioridad.

El modelo del canal de comunicación se describe mediante componentes Mast de los tipos Fixed_Priority_Network, Packet_Driver y FP_Sched_Server.

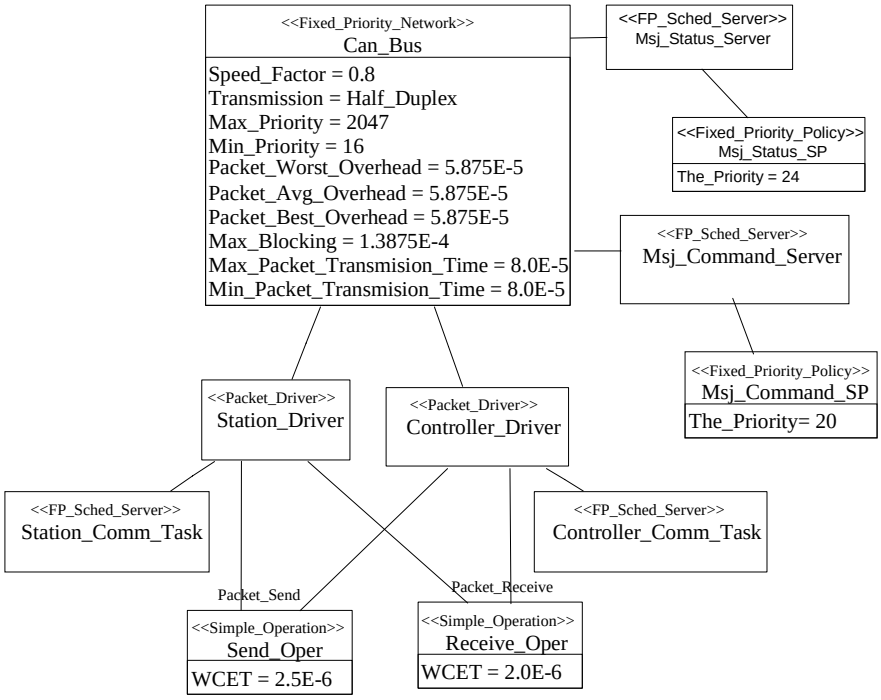

Fig. 6. Modelo del Network Can_Bus

El componente Can_Bus es un Fixed_Priority_Network que describe la capacidad del canal. El intervalo de prioridades que se utiliza para planificar los mensajes que se transfieren va de 16 a2047. La sobrecarga del bus por información de protocolo que corresponde a las cabeceras de mensajes simples del bus CAN es de 58.75 μs. La transmisión se hace en modo Half_Duplex. El tiempo máximo durante el cual el canal queda reservado es durante la transferencia de un paquete y sus cabeceras que equivale a 138.75 μs. Sólo la transmisión de un paquete (8bytes) tarda 80 μs. Los tiempos de transferencia de mensajes se han referido a un canal con capacidad de transferir 1 Mbit/segundo que es la capacidada nominal de bus, por lo que el factor de velocidad del bus empleado será de 0.8 (100Kbytes/seg).

4. Modelo Funcional

El modelo de tiempo real de los componentes lógicos modela el comportamiento temporal de los componentes funcionales (clases, métodos, procedimientos, operaciones, etc.) que están definidos en el sistema y cuyos tiempos de ejecución condicionan el cumplimiento de los requerimientos temporales definidos en los escenarios de tiempo real que van a analizarse. El modelo de cada componente lógico describe los dos aspectos que condicionan su tiempo de ejecución: el tiempo que requiere la ejecución de su código que es función de la complejidad de los algoritmos que contiene y los bloqueos que puede sufrir su ejecución como consecuencia de que necesita acceder en régimen exclusivo a recursos que también son requeridos por otros componentes lógicos que se ejecutan concurrentemente con él. El modelo de tiempo real de los componentes lógicos se establece con las siguientes características:

- Las temporizaciones de las operaciones se definen normalizadas, esto es, se formulan con parámetros cuyos valores son independientes de la plataforma en que se van a ejecutar.
- El modelo de interacción entre componentes lógicos se formula de forma parametrizada, identificando los recursos que son potenciales causas de bloqueos (Shared_Resource) y dejando hasta la descripción del escenario la declaración de los componentes lógicos concretos con los que va a interferir.
- El modelo de tiempo real de los componentes lógicos, se formula con una modularidad paralela a la modularidad que en la vista lógica ofrecen los componentes lógicos que se modelan.

El modelo de los componentes lógicos está constituido por un conjunto de diagramas de clases y de diagramas de actividad en los que se declaran:

- Los componentes que modelan el comportamiento de clases de la vista lógica del sistema que son relevantes a efecto de su respuesta de tiempo real.
- Operaciones predefinidas que realizan coprocesadores, dispositivos o periféricos no programables del sistema y que influyen en la respuesta temporal del sistema.
- Recursos compartidos que requieren ser accedidos por los componentes funcionales en régimen de exclusión mutua y que describen las posibles interacción con otros componentes que se ejecutan en concurrencia.

4.1 Extracto del Metamodelo de Componentes Funcionales

En la figura 7 se muestran las principales clases abstractas del metamodelo Mast con las que se formula el modelo de los componentes lógicos del sistema:

- **Operation:** Conceptualmente modela una actividad secuencial que puede ser ejecutada por un thread simple. Una vez iniciada no requiere ningúna sincronización con otro thread para terminar de ejecutarse. Su ejecución solo es afectada por las suspensiones que se producen mientras que no sea planificada en su Processing_Resource o por los bloqueos que resulta al requerir un Shared_ Resource que está ocupado. En el caso de las operaciones compuestas, la operación se describe mediante un diagrama de actividad.

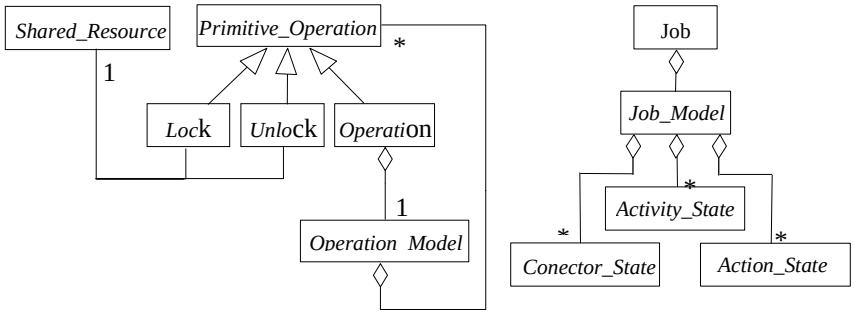

Fig. 7. Clases más representativas del metamodelo de los componentes lógicos

- **Job:** Modela la actividad concurrente que se desencadena cuando se ejecuta un componente lógico (método o procedimiento). La actividad que se desencadena requiere que intervengan múltiples threads y puede perdurar aún después de haber concluido el método que lo desencadenó. Todo job tiene agregado un diagrama de actividad que describe su naturaleza.
- **Shared_Resource:** Modela un recurso que es accedido en régimen de exclusión mutua por operaciones concurrentes y ocasiona posibles bloqueos al acceder a él.

4.2 Ejemplo de Modelo Funcional

Como se aprecia en el diagrama de clases de la figura 4, el software del Controller está constituido por tres clases activas y una clase pasiva a través de la que se comunican. La clase Servo_Controller implementa el algorítmo de control de los servos y la monitorización de los sensores. Es una tarea periódica disparada por el timer. La tarea Reporter es también una tarea periódica que transfiere el status del robot a la estación de teleoperación. La tarea Command_ Manager es una tarea disparada por la llegada de un mensaje a través del bus CAN. Su función es descomponer los comando de alto nivel en secuencias de consignas para los servos. Estas tres tareas son asíncronas entre sí y se comunican a través del objeto protegido Servos_Data. El software de la estación es típica de una aplicación GUI. La tarea Command_Interpreter gestiona los eventos que el operador introduce sobre la GUI. Transforma los eventos en comandos que transfiere hacia el Controller. La tarea Display_Refresher actualiza los datos de la GUI de acuerdo con los mensajes de status que recibe de Reporter. El acceso a la clase pasiva Display_Data se realiza con exclusión mutua a través de un objeto protegido.

En los diagramas de clases y de actividad de las figuras 8 y 9 se describen los modelos de los procedimientos de los objetos Servos_Data y Servos_Controller.

El primero representa un ejemplo de clase pasiva que implementa objetos protegidos utilizados para sincronización. Se modela mediante el objeto Servos_Data_Lock que es del tipo Shared_Resource y que representa el mecanismo de sincronismo propio de un objeto protegido. Las operaciones Get o Put se modelan mediante un objeto del tipo Simple_Operation y con los valor de su atributo se define

la cantidad de cómputo que requiere su ejecución. La composite_operation Protected_Oper define para cualquier objeto protegido la secuencia de operaciones que requiere la ejecución en modo protegido de una operación. A través de su modelo de actividad se establece que al comienzo debe accederse en régimen exclusivo al recurso, luego ejecutar la operación (Op) y al finalizar liberar el recurso.

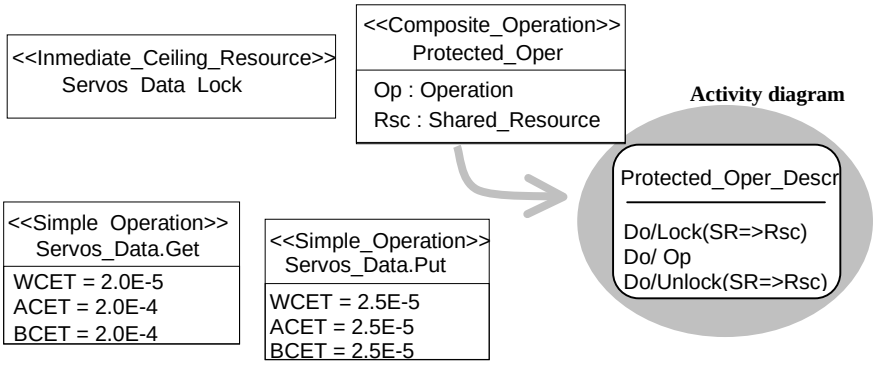

Fig. 8. Modelo de los componentes lógicos, objeto protegido Servos_Data

En la figura 9 se describen los procedimientos de la clase Servos_Controller. Aquellos (como Control_Algoritm o Do_Control) que son secuenciales y no requieren acceso o sincronización con otras clases se describen como operaciones simples, mientras que aquellos (como Control_Servos) que requieren en su ejecución la invocación de otros procedimientos se modelan como operaciones compuestas y cada una de ellas se describe mediante un diagrama de actividad en el que se enumeran los procedimientos que implica su ejecución.

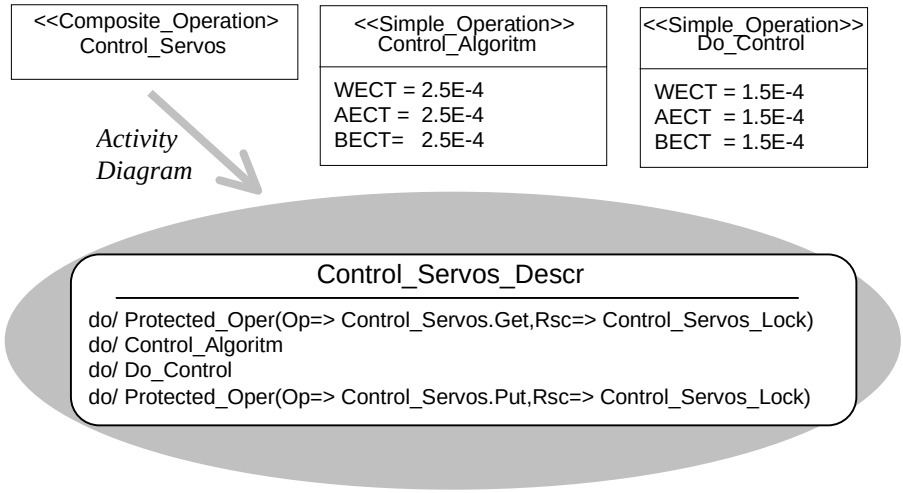

Fig. 9. Modelo de los componentes lógicos, clase Control_Servos

5. Modelado de Escenarios.

Los Escenarios de Tiempo Real son los modos o configuraciones de operación hardware/software del sistema para los que existen definidos requerimientos de tiempo real. Los escenarios representan las diferentes cargas de trabajo del sistema (workload) en las que deben satisfacerse los requerimientos de tiempo real.

Cada escenario de tiempo real se modela como un conjunto de Transacciones. El conjunto de transacciones definidas dentro de un escenario describe la máxima carga posible del sistema de tiempo real en ese escenario. Cada Transacción describe la secuencia no iterativa de actividades que se desencadenan como respuesta a un patrón de eventos externos que sirve de marco para definir los requerimientos temporales. Cada transacción incluye todas las actividades que se desencadenan como consecuencia del patrón de eventos de entrada y todas la actividades que requieren sincronización directa con ellas (intercambio de eventos, sincronización por invocación, etc.). No necesitan modelarse dentro de una transacción, otras actividades concurrentes que influyen en su evolución a través de competir con las actividades de la transacción por el uso de recursos comunes, ya sean recursos de procesamiento (processing_resource) o recursos compartidos (shared_resource) a los que se debe acceder con exclusión mútua (objetos protegidos, monitores, mutex, etc.). La presencia de estos recursos que son compartidos por las actividades de varias de las transacciones que coexisten concurrentemente dentro de un mismo escenario y que implican bloqueos y retrasos de sus actividades, es la causa de que el análisis de tiempo real deba hacerse contemplando simultáneamente todas las transacciones de un mismo escenario.

Aunque las transacciones son secuencias abiertas (no iterativas) de actividades, las transacciones se activan según un patrón de repetición y sus sucesivas ejecuciones se solapan entre ellas. Esto es lo habitual en sistemas que operan en modo pipeline.

Cada escenario de un sistema es independiente a efectos de su análisis. Todos ellos tienen en común, que su descripción se realiza compartiendo los componentes del modelo de plataforma y del modelo lógico del sistema, y por ello, resulta conveniente su descripción conjunta dentro de una misma estructura de datos.

5.1 Extracto del Metamodelo de Escenarios

El modelo de un escenario se compone del conjunto de modelos de sus transacciones. El modelo de cada transacción se compone de su declaración y de su descripción. Una transacción se declara mediante un objeto de la clase Transaction . Agregadas a cada objeto Transaction de declaran la lista con la descripción de las fuentes de eventos externos (External_Event_Source) que constituyen su patrón de eventos externos de disparo y la lista con la descripción de los requerimientos temporales definidos en ella (Timing_Requirement). La descripción de una transacción se realiza mediante un modelo de actividad agregado a su declaración. Una Transación representa una secuencia de actividades que se disparan como respuesta a un patrón de eventos externos y que sirve de marco para formular los requerimientos temporales en esa secuencia. Una transacción incluye todas las actividades que dependen entre sí por transferencias de flujo o sincronización activa y cuyas líneas de control de flujo tienen

su origen en la ocurrencia de los eventos externos que la disparan. Las actividades y los eventos externos no son compartidos entre transacciónes, sin embargo, las actividades pertenecientes a las transacciones de un escenario compiten entre sí por recursos comunes, tanto por Processing_Resource como por Shared_Resource.

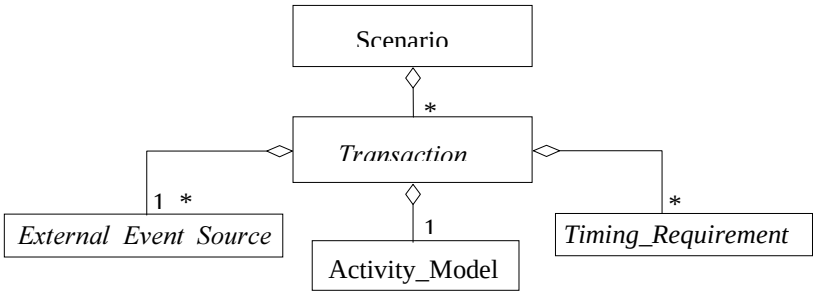

Fig. 10. Metamodelo de un escenario

Una transacción se describe mediante un Activity_Model agregado a su declaración. El modelo de actividad esta compuesto de un conjunto de diagramas de actividad que describen las secuencias de actividades, estados y transiciones que se desencadenan como consecuencia de un patrón de eventos externos de entrada.

5.2 Ejemplo de Escenario de Tiempo Real.

El escenario Control_Teleoperado que se analiza está constituido por tres transacciones asíncronas que interfieren entre sí por compartir los recursos de procesamiento (procesadores y canal de comunicación) y necesitar sincronizar sus acceso a los objetos pasivos. Las tres transacciones tienen requerimientos de tiempo real. La transacción Control_Process ejecuta el procedimiento Control_Servos con periodo y deadline de 5 ms. La transacción Report_Process transfiere a través del canal la información del status de los servos para actualizar el display con periodo y deadline de 100 ms. Por último la transacción Command_Process es de naturaleza esporádica pero su frecuencia de repetición está limitada a una por segundo. El diagrama de secuencia de la figura 11 expresa la descripción funcional de la transacción Report_ Process. En la figura 12 se muestra su modelo UML_Mast. La transacción se declara mediante un diagrama de clases con un objeto de la clase Regular_Transaction, enlazado con el objeto de la clase Periodic_Event_Source que representa el único evento externo que la dispara (Init_Report que es periódico y se genera cada 100 ms) y enlazado con el objeto de la clase Hard_Global_Deadline que representa el único requerimiento temporal que tiene (Display_Refreshed) que establece que la transacción debe finalizar antes de que se inicie la siguiente invocación. La transacción se describe mediante un diagrama de actividad, que representa las actividades que la componen y los scheduling server (threads) en que se ejecutan representados por en el diagrama por swimlanes.

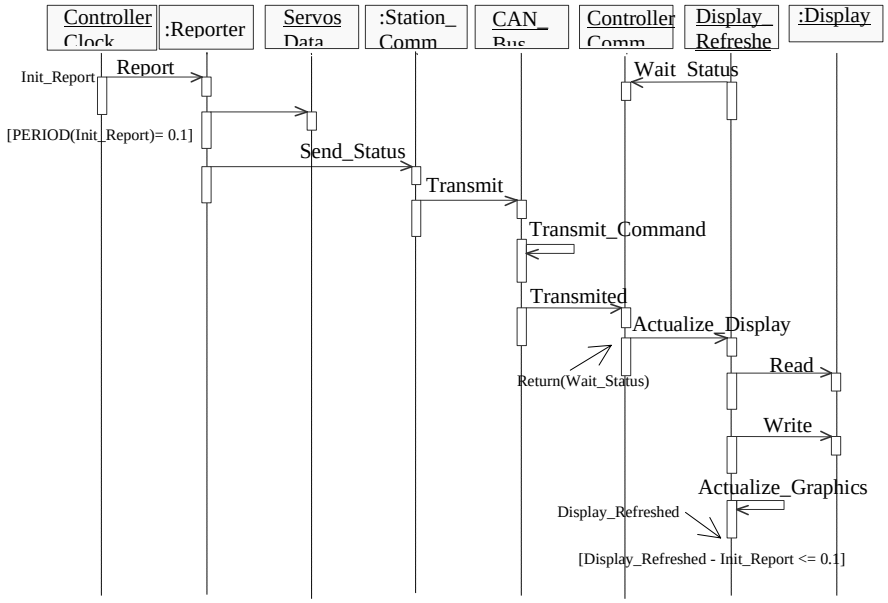

Fig. 11. Diagrama de Secuencia de la Transacción Report_Process

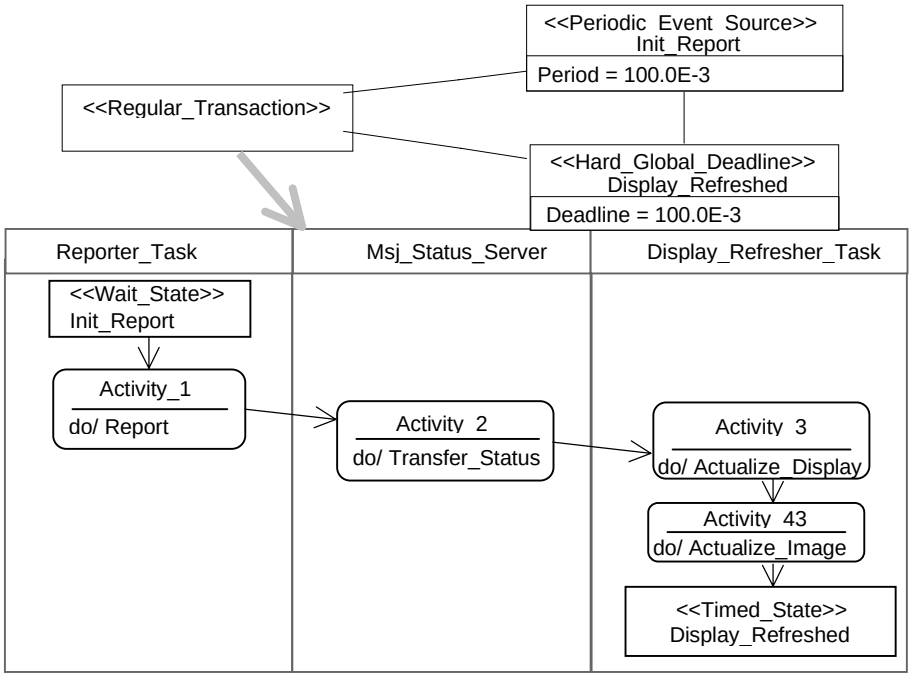

Fig. 12. Modelo de Escenarios: transacción Report_Process

6. Procesado del Modelo de Tiempo Real

El término **Procesado** de un modelo se emplea aquí para indicar la realización tanto de análisis como de síntesis [3]. Algunas de las herramientas del entorno Mast lo convierten por tanto en un procesador de modelos, puesto que realiza el cálculo de valores óptimos para las prioridades de planificación. Por ello UML-Mast se atiene al paradigma del procesado de modelos tal como se enuncia en [3].

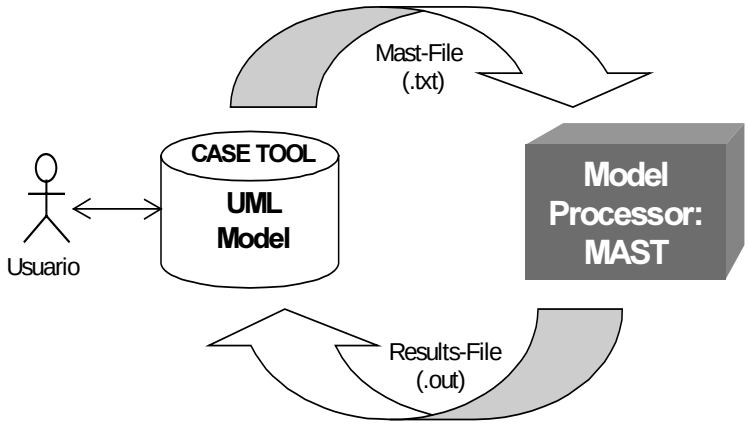

Fig. 13. El Paradigma de procesado de modelos.

Como se muestra en la figura 13, el procesado de la Vista de Tiempo Real se realiza a través del entorno Mast. El compilador genera un moldelo Mast_File del sistema, y lo entrega al entorno Mast en el que se pueden invocar las diferentes herramientas de análisis. Los resultados de las diferentes herramientas de análisis, pueden ser importados por UML-Mast nuevamente en ROSE y visualizados como valores de atributos de los objetos del modelo con los que están relacionados.

6.1 Utilización de la UML-Mast

UML-Mast incorpora a la herramienta Rational ROSE 2000e, un framework y un conjunto de herramientas que facilitan la introducción y el procesado de la Mast RT View:

- Genera en el Browser la estructura de directorios.
- Incluye en la herramienta la lista de estereotipos que se utilizan.
- Permite la elaboración asistida de los componentes del modelo.
- Proporciona un compilador que permite validar el modelo y en caso de que sea correcto generar el Mast File que requiere el entorno de análisis Mast.
- Finalmente permite reincorporar los resultados del análisis al modelo UML.

El análisis se realiza desde la propia herramienta ROSE. En la figura 14 se muestran las opciones que son añadidas por UML-Mast al menú Tools de Rose una vez instalado UML-Mast.

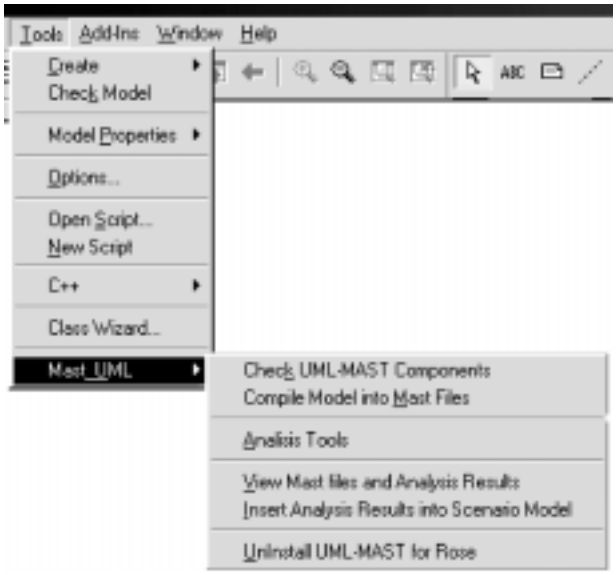

Fig. 14. Menú de acceso a UML-Mast desde Rational Rose.

- **Check UML-MAST Componets:** Permite verificar la definición de los componentes de la Mast RT View y la consistencia de sus asociaciones que constituyen el modelo de tiempo real. Los resultados del chequeo se proporcionan en una ventana específica “Mast Real-Time View components check”.
- **Compile Model into Mast File:** Compila la Mast RT View en un modelo Mast formulado mediante su correspondiente fichero textual. Por cada uno de los escenario del modelo se genera un modelo Mast independiente. En nuestro ejemplo, la compilación del modelo Teleoperated Robot genera en el direcctorio en el que está almacenado el modelo “Teoperated_Robot.mdl”, un nuevo directorio con el nombre “Teleoperated_Robot_Mast_Files”. En el nuevo directorio se almacena el fichero “Control_Teleoperado.txt” resultante de la compilación, que tiene como nombre el del escenario analizado y que en este caso es el único.
- **Analysis Tools:** Da acceso a la ventana de invocación de las herramientas del entorno Mast, que se muestra en la figura 15. Desde ella se elige el tipo de herramienta, los aspectos que se desean analizar y el directorio y el fichero en que se encuentra el modelo Mast que se analiza. El resultado del análisis se almacena en el fichero Control_Teleoperated.out.
- **View Mast Files and Analysis Results:** Permite observar los ficheros de texto resultantes de la compilación de cada escenario así como los resultados que la herramienta de análisis retorna para cada uno de ellos.

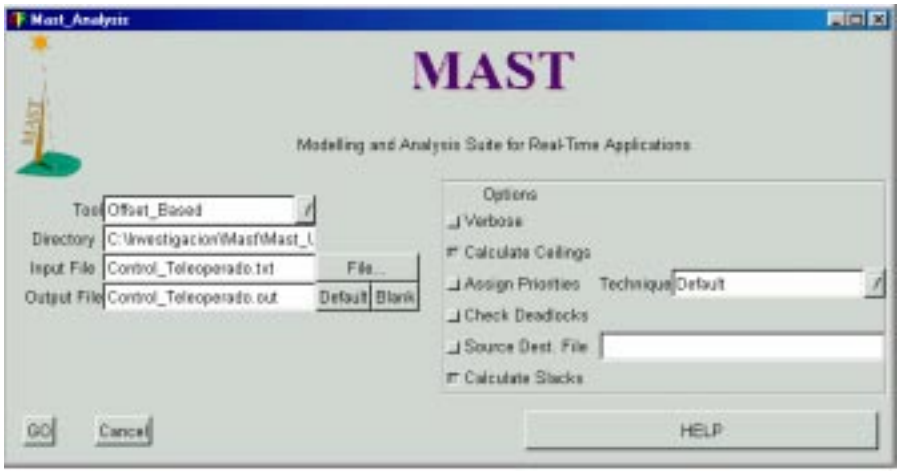

Fig. 15. Ventana de invocación de las herramientas Mast.

- **Insert Analysis Results into Scenario Model:** Si el sistema resulta planificable, permite recoger en el modelo de escenarios los resultados de tiempos de respuesta y holguras para cada transacción del escenario elegido.

Las herramientas de compilación y de incorporación de resultados se han realizado en BasicScript, utilizando las librerías proporcionadas por ROSE.

6.2 Aplicación al Ejemplo y Resultados Obtenidos

El sistema se ha analizado utilizando las herramientas que calculan tiempos de bloqueo, techos de prioridad de los recursos compartidos y la holgura de las transiciones y del sistema del entorno Mast. Se ha utilizando el método de análisis basado en offsets, que es el menos pesimista de los disponibles. Los resultados se obtienen de vuelta en el modelo UML de las transacciones y se muestran en la tabla 1.

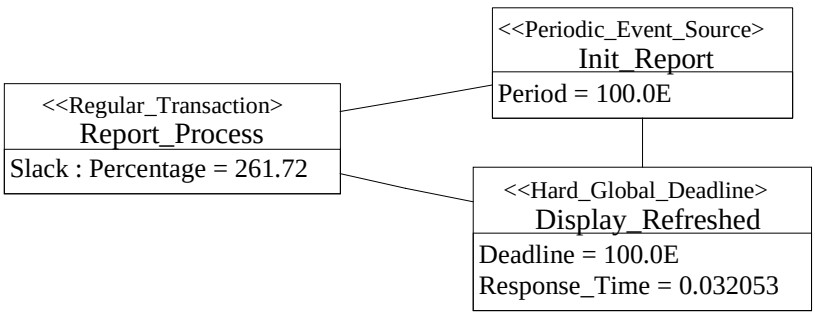

Fig. 16. Resultados incorporados en una de las transacciones del escenario

Tabla 1. Resultados obtenidos para las transacciones del Ejemplo Robot Teleoperado

Transacción	Slack(%)	Evento(s)	Respuesta(ms)	Deadline(ms)
Control_Servos_Trans	88.28%	End_Control_Servos	3.2	5
Report_Process	261.72%	Display_Refreshed	32.1	100
Execute_Command	180.47%	Command_Processed	370.1	1000

7. Conclusiones

Se presenta UML_Mast, una metodología para el modelado y análisis de sistemas de tiempo real descritos usando técnicas orientadas a objetos. Esta herramienta es independiente de la metodología de diseño que se use, pues consiste en complementar la descripción del sistema con una nueva vista que describe en UML mediante componentes conceptuales específicos aquellos aspectos que condicionan su comportamiento temporal, así como las capacidades de los recursos para satisfacerlas. Se han desarrollado herramientas para analizar de forma automática la "Mast RT View" del sistema y con ellas el diseñador puede aplicar, en las diferentes fases de desarrollo, diversas técnicas de análisis sin adentrarse en los algoritmos concretos que estas utilizan, tan sólo modelar el sistema e interpretar los resultados. UML-Mast está aún en fase experimental, tanto por depuración, como porque parte de las herramietas de análisis que se tiene por integrar están en fase de desarrollo. Sin embargo está operativa y permite analizar los aspectos mas relevantes de un sistema de tiempo real. Tanto la herramienta como toda la documentación explicativa y de ejemplo está disponible en http://ctrpc17.ctr.unican.es/umlmast. Cualquier pregunta o sugerencia sobre la instalación o uso de UML_Mast puede dirigirla a los autores de esta comunicación.

Referencias

1. González Harbour M., Gutiérrez J.J., Palencia J.C. and Drake Moyano J.M.: “MAST: Modeling and Analysis Suit for real Time Applications” Euromicro,2001.
2. Drake J.M., Gonzalez Harbour M., Gutierrez J.J. y Palencia J.C.: “Description of the Mast Model”. http://ctrpc17.ctr.unican.es/mast.
3. Selic B., Moore A., Bjorkander M., Gerhartdt M. y Watson B.: "Response to the OMG RFP for Schedulability, Performance and Time” OMG document ad/2000-08-04. Agosto, 2000.
4. Selic B.: “A generic framework for modeling resources with UML”. Computer. Vol. 33 nº 6, pp. 64 – 69. June, 2000.
5. Drake J.M. y Medina J.L.: “Robot Teleoperado: Ejemplo de uso de UML-Mast”, Internal Report. http://ctrpc17.ctr.unican.es/umlmast
6. Drake J.M. y Medina J.L.: “UML-Mast Metamodel”. http://ctrpc17.ctr.unican.es/umlmast
7. González M. y Aldea M.: “MaRTE OS: An Ada Kernel for Real-Time Embedded Applications”. International Conference on Reliable Software Technologies,Ada-Europe'01.