Sim_MAST: Simulador de sistemas distribuidos de tiempo real[1]

Patricia López Martínez, Julio Medina y José M. Drake

Grupo de Computadores y Tiempo Real, Universidad de Cantabria.
Av. De Los Castros s/n 39005 – Santander, España
{lopezpa, medinajl, drakej}@unican.es
http://www.ctr.unican.es/

Resumen. Se describe el principio de operación, la estructura interna y las prestaciones de la herramienta Sim_MAST, que analiza mediante simulación el comportamiento de sistemas de tiempo real distribuidos complejos, proporcionando información detallada sobre su planificabilidad, sobre la holgura en sus tiempos de respuesta, y sobre los niveles de utilización de sus recursos activos (procesadores, redes de comunicación y dispositivos específicos) y pasivos (con acceso en régimen de exclusión mutua). Con el uso de esta herramienta, el diseñador de sistemas de tiempo real puede tanto verificar su correcto funcionamiento, como, en su caso, identificar los componentes internos que lo impiden, o incluso mostrar los escenarios dentro de los que éstos fallos se producen. Así mismo, esta herramienta permite abordar de forma conjunta y equilibrada tanto el cumplimiento de los requisitos de tiempo real estrictos (de peor caso) como también los requisitos de tiempo real laxos (formulados con criterios estadísticos) y los niveles de calidad de servicio. La característica más cuidada en el diseño del simulador ha sido el facilitar su futura extensión a nuevos paradigmas de diseño de tiempo real, y a nuevos elementos de modelado del comportamiento temporal. Esto se ha conseguido mediante un diseño orientado a objetos estricto, basado en un número reducido de clases raíces abstractas que por herencia se especializan en elementos concretos con la funcionalidad que requiere el paradigma que se utilice. La herramienta se ofrece como software libre, integrada dentro del entorno de diseño MAST y ofrece interfaces de usuario amigables para su invocación y para la visualización de los resultados y de las trazas que genera.

1 Simuladores en el proceso de diseño de sistemas de Tiempo Real

Los sistemas de tiempo real que se necesitan actualmente en la industria automovilística, aeroespacial, de telecomunicación, de potencia, etc., son muy complejos[1][2]. Utilizan plataformas computarizadas distribuidas constituidas por decenas de nudos procesadores, y deben ejecutar concurrentemente un gran número de aplicaciones interdependientes, de las cuales muchas tienen requisitos de tiempo

[1] Realizado dentro del proyecto "TRECOM: Sistemas de tiempo real empotrados, fiables y distribuidos basados en componentes" financiado por el Programa Nacional de Tecnologías de la Información y de las Comunicaciones (TIC 2002-04123-C03-2).

real (estrictos o laxos) y otras muchas exigen diferentes niveles preestablecidos de calidad de servicio. El diseño de los sistemas de tiempo real con estas características es un proceso muy complejo, ya que el tiempo es una magnitud que no se puede gestionar explícitamente en los lenguajes de programación ni habitualmente puede ser establecida de forma directa en el acceso a los servicios de los sistemas operativos. Por ello, en el diseño de los sistemas de tiempo real se necesita superponer a la descripción funcional estándar, una descripción o modelo adicional del comportamiento temporal, que incluya todos aquellos aspectos que son necesarios para hacer predecibles los instantes en que se producen sus respuestas. Sobre este modelo de tiempo real se negocian los requisitos de temporización en las fases de especificación del sistema, se razona sobre la arquitectura del sistema en las fases de análisis y diseño y se certifica el cumplimiento de los requisitos temporales en las fases de validación.

La complejidad de los sistemas de tiempo real distribuidos que actualmente se diseñan, así como el alcance de los objetivos del análisis (planificabilidad, cumplimiento de los requisitos temporales, niveles de utilización, holguras, cargas de trabajo, etc.) implican unos modelos que contienen multitud de datos que deben ser procesados por diferentes herramientas haciendo uso de sofisticados algoritmos de análisis. Por todo ello es imprescindible disponer de un entorno CASE que soporte la información y sus diferentes vistas, en el que se integren las diferentes herramientas de gestión y análisis y en el que se facilite el intercambio de resultados entre ellas a través de bases de datos y formatos estandarizados.

El grupo de Computadores y Tiempo Real de la Universidad de Cantabria ha dedicado un gran esfuerzo durante los últimos años en el desarrollo del entorno MAST (Modeling and Analysis Suite for Real-Time Applications) [3][4] que ha sido diseñado para ser utilizado tanto por los grupos de investigación en métodos y herramientas de tiempo real, como por los diseñadores de aplicaciones industriales, y por los proveedores de recursos y componentes de tiempo real.

Este trabajo presenta el simulador Sim_MAST, una nueva herramienta de este entorno, capaz de analizar el comportamiento temporal de un sistema de tiempo real para el que se haya construido un modelo haciendo uso de la metodología MAST. Este simulador proporciona información estadística detallada sobre los valores extremos y promedio de los tiempos de respuesta del sistema, de su planificabilidad, de la holgura con que se satisfacen los requisitos temporales y de los niveles de utilización de los recursos.

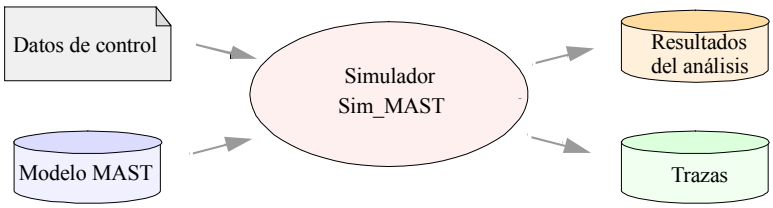


Fig. 1. Entradas y salidas del simulador

La herramienta es plenamente compatible con el entorno MAST[5]. Requiere como entrada el modelo de tiempo real de la aplicación, así como un conjunto de

datos que establecen el objetivo y el alcance del análisis, y genera como información de salida, un fichero con los resultados del análisis en un formato compatible con las restantes herramientas del entorno y un fichero de trazas que permite reproducir algunos aspectos de interés sobre la evolución del sistema durante una ejecución completa de simulación.

La incorporación del simulador en el entorno de diseño MAST aporta las siguientes ventajas:

- Proporciona capacidad de análisis para sistemas en los que coexistan requisitos temporales estrictos, laxos o especificados como niveles de calidad de servicio. Esto abre nuevas posibilidades al diseñador de sistemas de tiempo real ya que le permite abordar de forma conjunta y equilibrada todos los tipos de requisitos, y no como ha sido habitual hasta ahora, realizar el diseño en función sólo de los requerimientos estrictos, relegando las tareas con especificación de niveles de calidad de servicio a ser ejecutadas con la potencia residual.
- Capacita el análisis de cualquier aplicación que sea modelable con los recursos MAST. De este modo, el simulador libera al diseñador de tener que limitar la arquitectura del sistema a sólo aquellos patrones que satisfacen los conjuntos de restricciones que imponen las herramientas analíticas de análisis.
- Ayuda a la depuración de errores estructurales de los sistemas de tiempo real. Permite contrastar sus respuestas, y en caso de que presenten fallo, proporciona trazas que permiten documentar los escenarios en los que el fallo se ha producido.
- Constituye un medio de evaluación de los niveles de pesimismo que introducen las otras herramientas, y con ello establece criterios absolutos de valoración.

Por otra parte, la utilización de la simulación en el análisis de sistemas de tiempo real presenta dos desventajas relevantes, que no aconsejan considerar el simulador como la herramienta única del entorno de desarrollo:

- Es incapaz de analizar el peor caso, y en consecuencia no puede utilizarse para certificar de forma absoluta la corrección de los sistemas de tiempo real estrictos.
- Requiere una capacidad de procesamiento alta y por tanto no es útil en los casos de validación “en vivo” o test de aceptación, que se necesitan en el análisis de sistemas con cargas variables.

2 Plataforma de simulación

Al iniciar el diseño del simulador se estudió la tecnología de simulación que convenía utilizar. Había dos opciones: hacer uso de uno de los simuladores de propósito general o desarrollar el simulador como una aplicación de propósito especial. Se optó por la segunda opción, por tres razones: el entorno MAST es multiplataforma (Windows o Unix), se ofrece como software abierto y tiene que estar integrado en el entorno CASE para intercambiar resultados con las otras herramientas que éste ofrece.

El requisito que se ha priorizado en el diseño del simulador ha sido que pueda extenderse a futuros paradigmas de diseño de sistemas de tiempo real. MAST es un entorno en constante evolución y la plataforma de simulación debe prever la incorporación de nuevos componentes de simulación, e incluso que esos componentes sean introducidos por futuros usuarios.

La solución que se adoptó satisface estos requerimientos y se atiene a los siguientes criterios de diseño:

- Se ha adoptado una metodología orientada a objetos estricta. El simulador está basado en una serie de librerías que proporcionan el espacio de simulación, el cual, haciendo uso sistemático del polimorfismo, se formula en función de sólo cinco clases raíces abstractas, de las que se derivan por herencia/especialización todos los elementos concretos con los que se construyen los modelos de simulación a ejecutar. La especialización se reduce a modificar el estado interno y la funcionalidad del nuevo componente, incorporando nuevos atributos y sobrescribiendo los procedimientos que están definidos como abstractos en la clase raíz.
- El simulador se ha desarrollado a partir de un modelo UML que describe de forma precisa sus componentes, permitiendo desde él definir cada uno de ellos y generar automáticamente gran parte de sus códigos. La incorporación de un nuevo componente implica definir el modelo UML, generar automáticamente su código y sobrescribir el código de los procedimientos abstractos.
- Para la implementación del simulador se ha elegido el lenguaje Ada'95 que soporta perfectamente el estilo orientado a objeto que se ha adoptado y proporciona una programación muy segura para una aplicación como ésta, con mas de 8.000 líneas de código fuente (excluidos los comentarios).

En la figura 2 se muestra el metamodelo que describe el modelo de simulación empleado. Desde el punto de vista estático se concibe como una red, en la que los nudos son manejadores de eventos, que son instancias de tipos derivados de la clase abstracta *Event_Handler* y representan la funcionalidad de los componentes de modelado, y en la que los arcos son canales de eventos, instancias de tipos derivados de la clase abstracta *Event_Channel* que enlazan los manejadores y representan las vías de interacción entre ellos. La dinámica de la simulación la introducen los eventos, componentes derivados de la clase abstracta *Event*, que representan las interacciones que se producen entre los manejadores durante la simulación. La

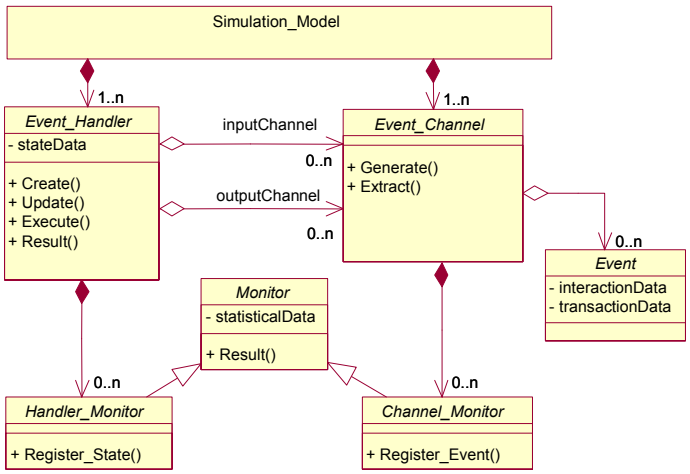


Fig. 2. Metamodelo de los modelos de simulación.

ejecución de la simulación se describe como la generación de eventos en los canales de salida de los manejadores, bien a causa de un fenómeno temporizado, o como reacción a los eventos que se han recibido a través de los canales de entrada. La estrategia de generación de eventos en los manejadores tiene una gran variabilidad y es función de su naturaleza y del estado interno en que se encuentran. Los eventos no sólo representan activación sino que también son contenedores de la información que se transfiere entre los manejadores, que bien cualifica la interacción o describe la evolución dinámica del escenario en el que se encuadran. Por último, los objetos de clases derivadas de la clase abstracta *Monitor* representan mecanismos de supervisión del proceso de simulación. Tienen una triple función: constituyen estimadores estadísticos de los tiempos y de la información representada por los eventos o por los cambios de estado de los manejadores de eventos, establecen las condiciones bajo las que la simulación finaliza, y constituyen el origen de las trazas que se generan.

La concreción de este simulador genérico y abstracto a un dominio específico, como en nuestro caso ha sido a la simulación de sistemas de tiempo real, se realiza a través de la especialización por herencia de las cuatro clases raíces (*Event_Handler, Event_Channel, Event y Monitor)*, con la funcionalidad requerida por el dominio al que se aplica. Definidos éstos, un modelo de simulación resulta de la instanciación de un conjunto de ellos, interrelacionados con la topología adecuada al sistema que se simula y a la información de salida que se busca.

El entorno del simulador está compuesto por el paquete de dominio *Sim* y la clase instanciada *Space*. Todos los componentes del simulador se generan como extensiones del paquete de dominio *Sim*, el cual ofrece los tipos globales del dominio (como *Time, Priority,etc.)* y el conjunto de excepciones de ámbito general.

El paquete *Space* constituye el núcleo que gestiona y controla la simulación. Es un paquete que contiene una librería cuyo código no requiere ser modificado para las diferentes extensiones del simulador. Desempeña tres funciones:

- La clase Space constituye el elemento contenedor en el que se registran y desde el que se gestionan los componentes que constituyen el modelo que se simula. En la figura 3 se muestra la vista de la clase *Space* en la que se muestra su funcionalidad como contenedor.

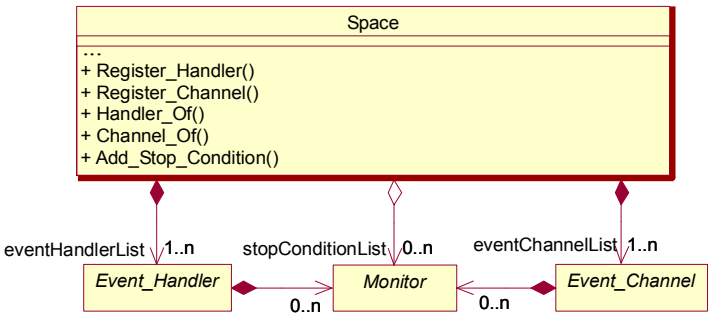


Fig. 3. Vista de *Space* como contenedor del modelo de simulación.

- La clase Space proporciona el motor de simulación que procesa el modelo, los recursos de inicialización e invocación y los recursos para finalización y elaboración de los resultados de la simulación.

El motor del simulador (*procedure Execute*) avanza la simulación hasta el siguiente tiempo en que ocurre un nuevo cambio, actualizando el estado de todos los componentes que estén registrados como temporizados y cuyo plazo coincida con el instante de ejecución, y resolviendo los efectos reactivos que se inducen en los manejadores como consecuencia de los eventos que se generan en cadena. El motor está soportado por dos listas que se gestionan desde *Space*, la lista de prioridad *independentQueue,* ordenada por tiempos y en la que se registran los manejadores que tienen pendiente una actualización por tiempos, y la lista FIFO c*hangedChannel,* en la que se registran todos los canales en los que existen eventos pendientes de ser procesados.

La inicialización (*procedure Init*) establece el estado inicial en la propia clase *Space* y recursivamente en todos los componentes registrados en ella, de acuerdo con los parámetros de simulación que se establezcan en la invocación del simulador.

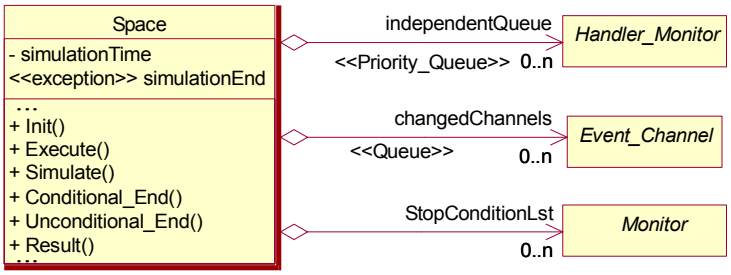


Fig. 4. Vista de la clase instanciada Space como motor de la simulación.

El lanzamiento de la simulación (*procedure Simulate*) invoca iterativamente el procedimiento *Execute* hasta que la simulación concluye porque se ha alcanzado alguna de las condiciones de finalización que ha sido establecida.

La finalización de la simulación (*procedure Conditional_End* y *procedure Unconditional_End*) es un aspecto crítico del simulador[6]. Si la simulación se extiende más allá de lo necesario, el consumo de capacidad de procesamiento puede hacerse insoportable, mientras que si se reduce por debajo de un mínimo puede hacer que la información que se genera no sea estadísticamente significativa. Por ello se han incluido diferentes condiciones de finalización que se utilizan de acuerdo con el objetivo que persiga la simulación. Todas las condiciones de finalización se pueden incluir en dos modos, bien como una condición de finalización incondicional, que cuando es alcanzada produce la conclusión inmediata de la simulación y que es útil para la detección y análisis de las trazas que han conducido a ella, o bien como una condición de finalización condicional que sólo producirá la finalización de la simulación si todas las que están registradas en Space como condicionales (*stopConditionList*) ya satisfacen el criterio de finalización. Este tipo de criterio condicional es muy útil cuando se necesita que la simulación proporcione resultados significativos sobre la estimación de muchos parámetros. Criterios de finalización que actualmente están definidos son: que en la simulación se genere un número predeterminado de algún tipo de evento o de acceso a un determinado estado, que no se haya satisfecho un requerimiento

temporal del modelo, o que la variabilidad de una variable estadística cumpla los requisitos necesarios para que se pueda estimar con un cierto nivel de confianza.

- La clase Space proporciona los recursos para comunicar el proceso de simulación con el operador que lo ha lanzado. Proporciona métodos para transferir mensajes sobre el estado de ejecución y sobre fallos, gestiona el almacenamiento eficiente de trazas de simulación, y exporta los resultados generados de acuerdo con el formato establecido en el entorno MAST. Para implementar esta funcionalidad, la clase Space tiene agregada la clase abstracta *Logger*, que requerirá ser especializada cuando las métricas o los formatos de transferencia de mensajes del simulador sean extendidos con nuevos criterios.

3 Modelos de tiempo real y modelos de simulación.

La actual implementación de la herramienta Sim_MAST tiene capacidad de procesar cualquier modelo de sistema de tiempo real que haya sido formulado de acuerdo con la versión 1.2.2 de MAST. Esta versión permite modelar aplicaciones de tiempo real distribuidas que se basan en políticas de programación basadas en prioridades fijas y con mecanismos de sincronismo en el acceso a los recursos pasivos que se corresponden con los ofrecidos en los sistemas operativos estándar (como los basados en POSIX de tiempo real) y en lenguajes de programación que soporten tiempo real (como Ada'95). Está basado en un modelo conducido por eventos que permite describir dependencias complejas entre las actividades. Características destacables que hacen esta metodología de modelado de tiempo real muy competitiva para el caso de sistemas de tiempo real distribuidos son:

- Utiliza primitivas de modelado que describen independientemente la capacidad de procesamiento que ofrece la plataforma en que ejecuta la aplicación, la cantidad de procesamiento que requiere la ejecución de cada actividad que se ejecuta en la misma y la forma en que se ordena la ejecución de las actividades y los accesos a recursos.
- Las transacciones pueden tener estructuras no lineales, esto es, pueden ser iniciadas mediante patrones complejos de eventos externos o temporizados y admiten relaciones de concurrencia (*f*ork), sincronismo (*j*oin), bifurcación (*b*ranch) y convergencia (*m*erge) de flujo de control entre las actividades.
- Modela con un tratamiento equivalente a los procesadores (que ejecutan código) y a las redes de comunicación (que transfieren información entre procesadores), lo que proporciona una gran capacidad para describir sistemas distribuidos.
- Admite múltiples políticas de planificación de las actividades respecto de su ejecución en un recurso de procesamiento ("*preemtible*", "*nonpreemtible*", "*interruption*", "*polling*" y "*sporadic server*") o de acceso a un recurso compartido pasivo (*Immediate_Ceiling* y *Priority_Inheritance*).
- Admite transacciones "End-to-End" distribuidas que combinan actividades que se ejecutan en diferentes procesadores y que requieren transferencia de mensajes por diferentes redes de comunicación.
- Permite declarar una amplia gama de requerimientos temporales, entre los que se distinguen globales y locales, que hacen respectivamente referencia a eventos

externos o internos, los estrictos (*hard*) y laxos (*soft*) que se caracterizan por su cumplimiento obligado o por tasas estadísticas de cumplimiento, y que pueden a su vez hacer referencia tanto a plazos de ejecución de las actividades como a *jitters*.

Una descripción completa del entorno MAST, de los recursos de modelado en que se basa, de las herramientas que ofrece, así como un conjunto de ejemplos a los que se aplica, puede encontrarse en la página web http:/mast.unican.es.

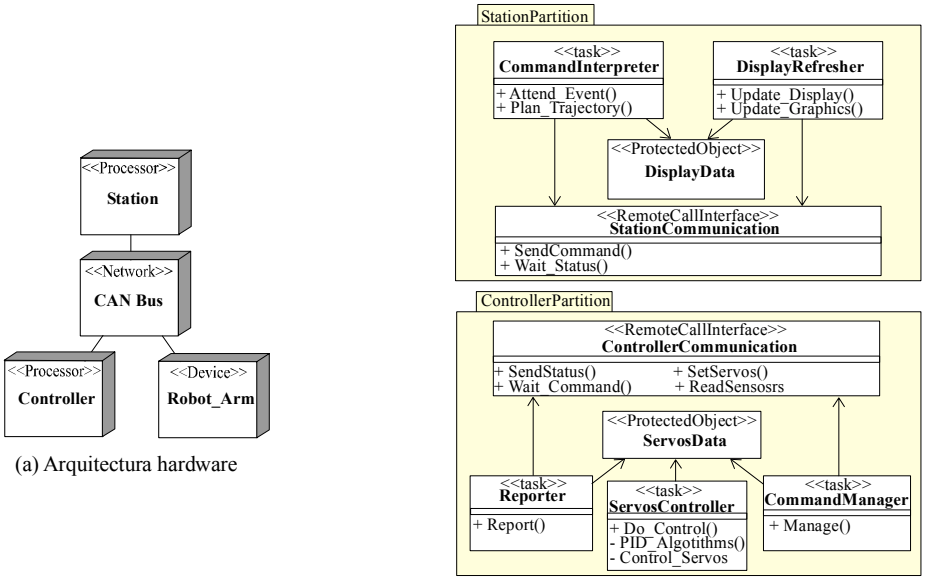
Tabla 1. Manejadores de eventos concretos definidos en Sim MAST.

MAST	**Event_Handler**	**Funcionalidad**
Processing Resource	FP_Processing_Resource	Simula un procesador dotado de un planificador de prioridades fijas
	FP_Packet_Network	Simula una red de comunicaciones con transmisión de los mensajes descompuestos en paquetes y planificados mediante prioridades fijas
Timers	Ticker	Simula un temporizador basado en interrupciones periódicas
	Alarm_Clock	Simula un reloj basado en interrupciones por evento
Scheduling Server	FP_Scheduling_Server	Entidad de planificación basada en prioridades fijas, con política reentrante, no reentrante o tipo interrupción.
	FP_Polling_Server	Entidad de planificación de una actividad con activación periódica.
	FP_Sporadic_Server	Entidad de planificación basada en la política de servidor esporádico.
Shared_Resource	FP_Shared_Resource	Recurso de acceso exclusivo. Con varias políticas de acceso basadas en P.F.
Activity	Activity	Actividad simple caracterizada por tiempo peor, mejor y promedio.
	System_Timed_Activity	Actividad simple de activación sincronizada con el reloj del sistema.
	Simplex_FP_Packet_Dispatcher	Transferencia de un mensaje por la red con descomposición en paquetes.
Flow_Controller	Generator Concentrator Barrier Multicast Delivery_Server Query_Server Rate_Divisor Delay Offset Sink	Diferentes tipos de manejadores de flujo de control que se utilizan para simular los elementos de modelado que permiten controlar la generación, transferencia y finalización de las líneas de flujo de control en el entorno MAST, excluidas las actividades.

En la tabla 1 se muestran los tipos concretos de manejadores de eventos que están actualmente implementados en el simulador. Su correspondencia con los componentes de modelado de la versión 1.2.2 de MAST es uno a uno en casi todos los casos, lo que hace que el programa que lee el modelo MAST y lo traduce en el correspondiente conjunto de manejadores de simulación registrados en Space sea sencillo desde este punto de vista.

Como ejemplo de modelado y análisis mediante el simulador de un sistema distribuido de tiempo real se muestra en la figura 5 la descripción de un sistema para

el control teleoperado de un robot. La plataforma está constituida por dos procesadores, "Station" que constituye la GUI desde la que el operador controla el robot y "Controller" que es un procesador embarcado en el robot. El programa de ambos procesadores es el objeto de la aplicación que se desarrolla. El dispositivo "Robot_Arm" está constituido por el conjunto de sensores y actuadores que controlan el brazo. Son elementos pasivos sobre los que la aplicación actúa. Los dos procesadores y el dispositivo se comunican a través de un único bus CAN. Obviamente si el software que se diseña tiene requisitos de tiempo real, la plataforma ha de ofrecer prestaciones de tiempo real, esto es, debe tener un comportamiento predecible. En este caso, ambos procesadores operan sobre el sistema operativo de tiempo real MARTE OS [7] y el software de comunicaciones está soportado por una extensión de tiempo real del compilador de Ada Distribuido GNAT-GLADE [8][9].



(a) Arquitectura hardware

(b) Arquitectura software

Fig. 5. Arquitectura hardware y software del ejemplo "Robot Teleoperado"

En la figura 5 también se muestran las clases principales que constituyen la arquitectura software de la aplicación. Es una arquitectura típica de sistema de tiempo real desarrollada con Ada distribuido. Se compone de dos particiones (una por procesador) y el software de cada una de ellas es un conjunto de tareas concurrentes que se comunican entre sí localmente a través de un objeto protegido y se comunican con los componentes remotos a través de los servicios ofertados por sendas RCI (Remote Control Interface) Ada. Las tareas se ejecutan con diferentes patrones de disparo. Algunas como "Servos_Controller" o "Reporter" son invocadas periódicamente por el reloj del sistema, otras son esporádicas como "Command_ Interpreter" y otras responden a eventos de comunicación ("Command_Manager" y "Display_ Refresher").

Desde el punto de vista de la especificación de tiempo real, esta aplicación se concibe como un conjunto de tres transacciones:

- Transacción "Control_Servos": es invocada periódicamente cada 5 ms por el reloj del procesador "Controller", y en función de las consignas establecidas que lee de ServosData y del estado actual del robot que obtiene de los sensores (a través del bus CAN), ejecuta un algoritmo de control que le permite deducir los valores que debe establecer como entradas de los servos (a través del bus CAN). Tiene como requisito temporal haber concluido antes del inicio de la siguiente invocación.
- Transacción "Report_Process": es invocada periódicamente cada 100 ms por el reloj del procesador "Controller", y requiere la elaboración de un mensaje con el estado del robot, su transferencia por el bus CAN y la actualización del display en el procesador Station de acuerdo con la información que transmite. Tiene como requisito temporal haber concluido antes del inicio de la siguiente invocación.
- Transacción "Command_Process": es invocada esporádicamente cada vez que el operador establece un comando en la GUI del procesador "Station". Conlleva la elaboración de la planificación de trayectorias en Station, la transferencia de la lista de consignas por el bus, y su gestión local en el procesador Controller. En el peor caso debe realizarse antes de que transcurra 1 s.

Para poder analizar esta aplicación se ha de disponer de un modelo de tiempo real del sistema [3]. Este se compone de tres secciones:

Modelo de la plataforma: Describe la capacidad de procesamiento de los procesadores y de transferencia de la red de comunicaciones, así como la parte de esa capacidad que es consumida por las tareas de fondo que introduce el sistema operativo. El modelo de un procesador o de una red es complejo y sus parámetros difíciles de evaluar, sin embargo tiene la ventaja de que al ser independiente de la aplicación sólo debe ser obtenido una vez. Por cada procesador se debe cuantificar su capacidad (speed_factor), los parámetros del planificador (políticas base de planificación, rangos de prioridades, tiempos de cambio de contexto y tiempo de atención de interrupciones, etc.), características del reloj (granularidad, carga de procesamiento que requiere su gestión, etc.). Igualmente de una red de comunicación se debe cuantificar su anchura de banda (throughput), los parámetros del planificador de mensajes (política base de planificación, rangos de prioridad, tiempos de transferencia de paquetes, etc.), los hilos de procesamiento y la capacidad de procesamiento que requieren los módulos de comunicación (drivers) de los procesadores que envían o reciben mensajes.
En el ejemplo "Robot Teleoperado" de la figura 5, el modelo de la plataforma requiere modelar los procesadores "Station" y "Controller" y la red de comunicación (CAN_Bus).

Modelo de los componentes lógicos: Describe la capacidad de procesamiento que requiere la ejecución de las operaciones definidas en los componentes lógicos que se utilizan en el diseño, tales como procedimientos y funciones definidos en las clases, primitivas de sincronización de hilos de flujo, procesos de comunicación por las redes, etc. En este modelo se declaran los recursos que cada operación necesita para poder ejecutarse, en especial aquellos que por ser requeridos por varias operaciones con régimen de exclusión mutua pueden ser origen de bloqueos en la ejecución de las operaciones. Por cada operación que sea relevante a efectos de estimación de los

tiempos de respuesta, se describe en el modelo la capacidad de procesamiento que se utiliza para su ejecución (en el mejor, peor y caso promedio), así como la lista de recursos compartidos que la operación requiere para su ejecución. El modelo de las operaciones puede formularse en función de otras operaciones más simples que han sido modeladas independientemente.
En el ejemplo "Robot Teleoperado" de la figura 5, el modelo de los componentes lógicos requiere modelar los procedimientos que son declarados en las clases, los dos recursos compartidos que modelan los mecanismos de sincronización de los objetos protegidos, y las entidades de planificación que describen los hilos de flujo que introducen las instancias de las cinco clases activas.

Modelo de la situación de tiempo real: Describe la configuración hardware/software que puede ser alcanzada por el sistema y que es objeto del análisis. La situación de tiempo real se modela como un conjunto de transacciones, cada una de las cuales es una descripción de las secuencias de operaciones y eventos que se desencadenan como respuesta a un patrón de eventos autónomos (procedentes del entorno exterior al sistema, de relojes, de dispositivos hardware integrados, etc.) y de los requerimientos temporales que se definen en ellas para especificar la evolución temporal que se requiere. En el modelo de cada transacción se describen las secuencias de operaciones de que se compone, especificando para cada una de ellas la entidad de planificación (Scheduling Server) del procesador en que se ejecuta, los patrones de generación de los eventos que la invocan y los requerimientos temporales que han de satisfacerse durante su ejecución.
En el ejemplo "Robot Teleoperado" de la figura 5, la situación de tiempo real debe contener el modelo de las tres transacciones "Control_Servos", "Report_Process" y "Command_Process".

4. Resultados de la simulación.

La información cuantitativa que proporciona el simulador tras su ejecución se proporciona asociada a los elementos del modelo MAST que se ha simulado, y se compone de:

- *Processing_Resources (Procesadores y Redes de comunicación)*: Se obtienen los siguientes datos:
  - *Utilization*: Describe el porcentaje de utilización del recurso que ha hecho la aplicación. La utilización total se desglosa en la parte utilizada por las actividades de la aplicación, en gestión del timer, en gestión de los drivers de comunicaciones y en los cambios de contexto entre procesos.
  - *Ready_Queue_Size*: Número máximo de tareas pendientes de ejecución que ha tenido durante la simulación.
- *Shared Resources*: Muy similar al anterior, se muestran los siguientes resultados:
  - *Utilization*: Representa el porcentaje de utilización total.
  - *Ready_Queue_Size*: Número máximo de tareas a la espera que ha tenido durante la simulación.
- *Transactions*: Para cada evento monitorizado de la transacción se genera la siguiente información:

- *Global_Response_Time*: Tiempos de respuesta global de peor, mejor y caso promedio con respecto a todos los eventos de generación posibles.
- *Local_Response_Time*: Tiempos de ejecución de una actividad (de peor, mejor y caso promedio).
- *Jitter*: Tiempos de jitter (de peor, mejor y caso promedio).
- *Blocking_Time*: Tiempos de bloqueo promedio y de peor caso.
- *Preemption_time*: Tiempo de expulsión de peor caso.
- *Suspension_Time*: Tiempo de suspensión.
- *Num_Of_Suspensions*: Número de suspensiones.
- *Num_Of_Queued_Activations*: Número máximo de activaciones pendientes.
- *Global_Max_Miss_Ratio*: En caso de que se haya establecido para algún evento, se describe el porcentaje de eventos que han superado el deadline.
- *Local_Max_Miss_Ratio*: Igual que el anterior pero relativo al tiempo de ejecución de una actividad.

Como ejemplo, en la figura 6 se muestra algunos de los resultados que se obtienen al analizar mediante simulación el ejemplo Robot_Teleoperado que se ha descrito en la figura 5.

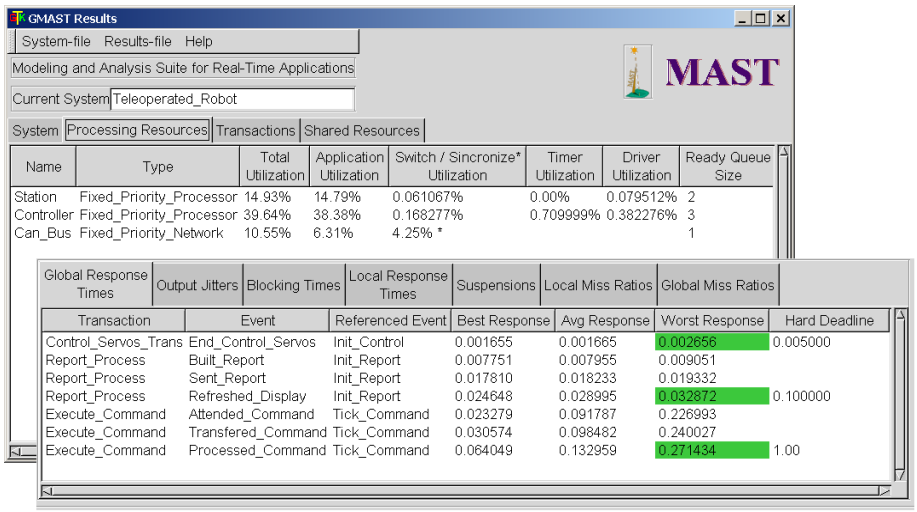


Fig. 6. Resultados del análisis del ejemplo "Robot Teleoperado"

En la primera ventana se muestra la utilización que la aplicación hace de los recursos de procesamiento. Del procesador "Station" se utiliza el 14.93% de su capacidad, del procesador "Controller" se utiliza el 39.64% y de la red CAN_Bus sólo el 10,55% de su anchura de banda. La utilización de los procesadores es muy eficiente, ya que prácticamente la totalidad de la capacidad utilizada es empleada en la ejecución de actividades de la aplicación. El uso del Bus_CAN es menos eficiente, ya que sólo el 60% de la anchura de banda es utilizado para transferir datos de la aplicación y el 40% es utilizado para transferir encabezamientos relacionados con el protocolo. Esta falta de eficiencia es consecuencia de que los mensajes que se transmiten son muy cortos.

En la segunda ventana se muestra la respuesta temporal de los eventos de flujo definidos en las tres transacciones. Los tres eventos que tienen asociados requerimientos temporales, satisfacen con holgura los plazos requeridos.

5 Entorno de la herramienta.

La herramienta Sim_MAST está constituida por los cuatro módulos que se muestran en la figura 7. Estos programas son:

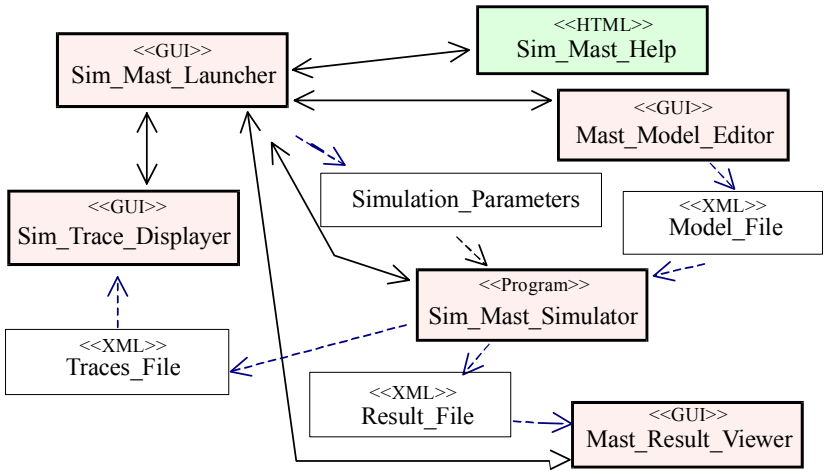


Fig. 7. Entorno de herramientas Sim_MAST

Sim_Mast_Launcher: Es una sencilla interfaz gráfica que permite invocar cualquiera de las herramientas que ofrece el entorno MAST, y entre las que se encuentra Sim_MAST. A través de ella el operador puede establecer los parámetros de ejecución que considere adecuados en cada caso.

Sim_Mast_Simulator: Programa que ejecuta la simulación del modelo, recibiendo como entradas el fichero con el modelo y los parámetros de configuración de la simulación que correspondan.

Mast_Resuls_Viewer: Interfaz gráfica que permite al operador visualizar los resultados que se han generado durante la simulación.

Mast_Traces_Displayer: Interfaz gráfica que permite al operador visualizar y explorar las trazas generadas por el simulador (en desarrollo).

En la figura 8 se muestra la interfaz Sim_Mast_Launcher que permite invocar el simulador Sim_MAST. Esta interfaz ha sido desarrollada empleando la librería gráfica de Ada, GtkAda, version 1.3.12. Proporciona un navegador que permite localizar o establecer el fichero que contiene el modelo que se simula. Así mismo, a través de un marco proporciona los diferentes controles que se necesitan para establecer los parámetros de configuración de la simulación. Y por último, a través de diversos botones habilita la invocación de las diferentes herramientas que permiten

gestionar el modelo, ejecutar la simulación y visualizar los resultados y las trazas. Así mismo, los visores de resultados (figura 7) y de trazas son sendas interfaces realizadas con la misma tecnología y que permiten visualizar amigablemente los resultados que se han generado en la simulación, siendo presentados en el contexto del modelo.

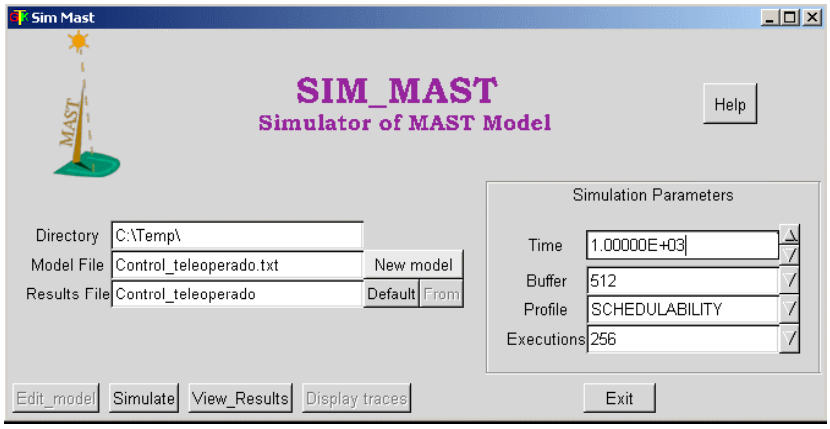


Fig. 8. Interfaz de lanzamiento del simulador**.**

Un aspecto complejo de la simulación es como el operador establece los parámetros de control de la simulación. Hay que decidir fundamentalmente tres tipos de parámetros: Que información se quiere extraer de la simulación, que criterio de finalización se utiliza y que trazas deben ser generadas. En Sim_MAST todo ello se traduce en decidir que conjuntos de monitores deben ser añadidos al modelo y a que elementos del mismo deben ser asociados. Como se ve en la tabla 2, existen diferentes tipos de monitores lo que haría esta tarea muy tediosa al operador. La solución propuesta es sencilla y consiste en la definición de un conjunto de *perfiles de simulación*. Cada perfil define genéricamente un objetivo para el que la simulación es invocada, en función de él se establecen los parámetros de control de la simulación.

Tabla 2. Manejadores de eventos concretos definidos en Sim MAST**.**

Tipo	*Monitor*
Condición de finalización	Stop_On_Num_Event Stop_On_Global_Deadline Stop_On_Local_Deadline Stop_On_Deadline_ _Confidence_Variation
Estadística de parámetros asociados a eventos	Global_Time Local_Time Jitter Local_Timing Global_Max_Miss_Ratio Local_Max_Miss_Ratio
Estadística de parámetros asociados a Event_Handler	Percent_On_State State_Time_Monitor Stop_On_State Log_State_Access

Actualmente se han definido tres perfiles:

- Perfil "*Schedulability":* Con él se configura el modelo de simulación para que genere la información básica que se utiliza en el proceso de diseño de un sistema de tiempo real. Ofrece una información similar a la que genera la herramienta de análisis de planificabilidad, aunque enriquecida con la información de valores promedio y con los niveles de uso de los recursos.
- Perfil *Verification*: Además de generar la misma información que en el perfil anterior, su objetivo es detectar y justificar la causa del incumplimiento de los requisitos temporales estrictos.
- Perfil *Exhaustive*: Tiene como objetivo generar una información exhaustiva de la ejecución de la simulación y es habitualmente utilizado sólo en fases de depuración del propio simulador y de los nuevos elementos que se vayan introduciendo.

6 Conclusiones y líneas de trabajo.

La versión actual del simulador Sim_MAST, se encuentra disponible en la página web **http://mast.unican.es/simmast/** para las plataformas Windows y Unix.

La herramienta desarrollada constituye una nueva ayuda para el análisis y diseño de sistemas de tiempo real distribuidos complejos:

- Proporciona una información exhaustiva del sistema de tiempo real tanto sobre los tiempos de respuesta, como sobre los niveles de utilización de los recursos.
- Es la única herramienta del entorno MAST que tiene capacidad de analizar sistemas en los que coexisten requisitos de tiempo real estricto y laxo, y que permite diseñar el sistema de forma que el cumplimiento de ambos tipos de requisitos sean considerados de una forma equilibrada.
- Es capaz de analizar cualquier sistema de tiempo real distribuido, con el único requerimiento de que sea modelable utilizando la metodología MAST. Esto contrasta con las limitaciones de las herramientas analíticas que introducen conjuntos de restricciones muy fuertes para poder ser aplicadas.
- Posee la capacidad de generar trazas de la ejecución de la simulación con lo cual podemos verificar el comportamiento del sistema e identificar posibles situaciones de fallo del sistema de manera sencilla.

La capacidad de extensión con que se ha dotado al simulador nos va a permitir añadir nuevos elementos en el futuro para abordar diferentes tipos de sistemas. Así en los próximos meses está previsto concluir su adaptación a la versión 1.3 de MAST, en la que se podrán simular sistemas con planificadores jerárquicos y que admitan también políticas de planificación basadas en EDF.

Una línea de particular interés futuro, es adaptar el entorno MAST y con él al simulador, para el análisis de sistemas de tiempo real que operan en sistemas abiertos[10]. En este tipo de sistemas la plataforma no está exclusivamente dedicada a la aplicación que se analiza sino que ejecuta concurrentemente con ella otras aplicaciones desconocidas, y que, incluso, pueden cambiar de manera dinámica, con lo cual no se puede elaborar un modelo exacto de la plataforma. Para este tipo de sistemas se están desarrollando nuevos servicios del sistema operativo que permitan a

las aplicaciones, en el momento de su instanciación, "negociar" con el sistema operativo por medio de un "contrato de servicio" en el que expongan los requerimientos que necesiten para cumplir sus requisitos. Si el sistema operativo acepta el contrato, la aplicación tendrá la seguridad de que va a obtener todo lo requerido y por tanto, va a cumplir sus plazos. Basándose en este modelo, se podrá modelar la plataforma en función de los contratos que tenga establecidos, pudiéndose realizar de este modo el análisis del sistema completo. Para abordar esta nueva estrategia se han realizado modificaciones en algunos elementos de MAST que será necesario trasladar al simulador.

Referencias.

1. C.Norström y otros: "Experiences from Introducing State-of-the-art real-time Techniques in the Automotive Industry". Proc. Of 8th IEEE Int. Conf. On Engineering of Computer Based Systems (ECBS01) Washington, April, 2001.
2. Sha L. Rajkumar R. and Gagliardi M.: "Dependable System Upgrade" IEEE Proc. 19th Real-Time Systems Symposium, pp. 440-448, Madrid 1998.
3. M. González Harbour, J.J. Gutiérrez, J.C.Palencia and J.M. Drake: "MAST: Modeling and Analysis Suite for Real-Time Applications". Proceedings of 13th Euromicro Conference on Real-Time Systems, Delft, IEEE Computer Society Press, pp. 125-134, June 2001.
4. J.L. Medina, M.Gonzalez Harbour y J.M. Drake: "MAST Real-Time View: A Graphic UML Tool for Modeling Object-Oriented Real-Time Systems". 22nd IEEE Real-Time Systems Symposium (RTSS 2001). London, December 2001.
5. J.M.Drake, M. Gonzalez Harbour, J.J.Gutierrez y J.C. Palencia: "Description of the MAST Model". http://mast.unican.es/.
6. R. Jain:"The art of Computer Systems Performance Analysis" Wiley, 1991.
7. González Harbour and M. Aldea: "MaRTE OS: An Ada kernel for Real-Time Embedded Applications". International Conference on Reliable Software Technologies, Ada-Europe'01. Leuven, May, 2001
8. L. Pautet and S. Tardieu: "GLADE: a Framework for Building Large Object-Oriented Real-Time Distributed Systems. In Proceedings of the 3rd IEEE". International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'00), Newport Beach, California, USA, March 2000.
9. J. L. Campos, J. J. Gutiérrez and M. González Harbour:" The Chance for Ada to Support Distribution and Real-Time in Embedded Systems" The 9th International Conference on Reliable Software Technologies, Ada-Europe, Palma de Mallorca (Spain), June, 2004.
10. K. Gopalan: "Real-Time Support in General Purpose Operating Systems".