Metamodelo UML para el modelado de tiempo real de aplicaciones distribuidas basadas en componentes

Patricia López Martínez, José M. Drake and Julio L. Medina

Departamento de Electrónica y Computadores, Universidad de Cantabria,
39005-Santander, SPAIN
{medinajl,lopezpa,drakej}@unican.es

Resumen: El modelo de tiempo real es la base del proceso de especificación, análisis, diseño y validación de cualquier aplicación de tiempo real. Tradicionalmente este modelo se ha formulado siguiendo el modelo reactivo con el que la aplicación ha sido concebida. Sin embargo, en sistemas complejos, en los que las aplicaciones de tiempo real se diseñan utilizando una metodología basada en componentes, el aspecto estructural de la aplicación domina la construcción del modelo de tiempo real y de él debe deducirse posteriormente el modelo reactivo. En este trabajo, se presentan los conceptos fundamentales del metamodelo que se propone a fin de formalizar la construcción de los modelos de comportamiento temporal de aplicaciones distribuidas y de tiempo real basadas en componentes. La metodología de modelado que representa, es modular desde tres puntos de vista complementarios. En primer lugar, permite describir mediante módulos de modelado independientes el comportamiento temporal de los componentes software o hardware reusables en los que se basan el diseño de las aplicaciones, y asimismo, hace posible que se construya el modelo de una aplicación determinada mediante el ensamblado de los modelos de sus componentes. En segundo lugar, modela de forma independiente la capacidad de procesamiento de la plataforma hardware/software de ejecución, y la cantidad de capacidad de procesamiento que requiere la ejecución de las actividades que se ejecutan en la aplicación. Y en tercer lugar, el modelo se construye siguiendo la estructura de componentes de la aplicación (modelo estructural), y sobre él, se formula de forma independiente la carga de trabajo que ejecuta (modelo reactivo).

Palabras clave: Modelo de tiempo real, UML, Sistemas basados en componentes, Análisis de planificabilidad

1 Introducción[1]

Los sistemas de tiempo real que se utilizan actualmente en la industria son muy complejos. Utilizan plataformas distribuidas constituidas por decenas de nudos procesadores, que ejecutan un gran número de aplicaciones independientes o asociadas, muchas de las cuales tienen requisitos de tiempo real, y otras exigen niveles preestablecidos de calidad de servicio. Los procesos de desarrollo de estos tipos de aplicaciones se basan en la disponibilidad de un modelo que describa cualitativamente y cuantitativamente el comportamiento temporal del sistema completo. Este modelo, que se denomina

1. Este trabajo ha sido financiado por la Comisión Interministerial de Ciencia y Tecnología del gobierno español dentro del proyecto de investigación TIN 2005-08665-C03-02.

modelo de tiempo real, es el medio de que se vale el diseñador para formular los requisitos temporales durante la fase de especificación, razonar en las fases de diseño sobre la arquitectura y el modelo de concurrencia que son más adecuados, y certificar la planificabilidad en las fases de validación. El modelo de tiempo real es una abstracción del sistema que contiene de forma relacionada toda la información que se necesita para predecir y evaluar su comportamiento temporal.

Aunque hay diferentes metodologías para formular los modelos, el más utilizado es el conocido como "Modelo Transaccional" [1][2], con el que el sistema se modela mediante dos descripciones complementarias:

- Modelo de flujo de control o transaccional, es un modelo reactivo que describe la aplicación en función del conjunto de secuencias de actividades o transacciones que concurren en ella y que se desencadenan como respuesta a los eventos procedentes del entorno o de temporización generados por el reloj. Este modelo constituye el marco en el que se especifican los requisitos de tiempo real.
- Modelo de uso de recursos o de contención, que describe los recursos limitados de procesamiento y de sincronización que se disponen para llevar a cabo concurrentemente el conjunto de actividades, y que condicionan el modo en que puede hacerse uso de la capacidad de ejecución disponible en el sistema.

La importancia actual de la metodología de modelado transaccional es consecuencia de tres hechos. En primer lugar, basados en él se han desarrollado un amplio conjunto de métodos y herramientas de análisis de planificabilidad, estimación de tiempo de respuesta, asignación óptima de prioridades, etc.[2][3][4]. En segundo lugar, la mayoría de los sistemas operativos [5] y lenguajes de programación para diseño de sistemas de tiempo real ofrecen recursos de sincronización y políticas de planificación que tienen su fundamento en el hecho de que los sistemas de tiempo real basados en ellos son analizables utilizando esta metodología de modelado. En tercer lugar, el perfil "UML profile for Schedulability, Performance and Time" (SPT) [6] propuesto por OMG como estándar para garantizar la interoperatividad de herramientas CASE de diseño de tiempo real, tiene sus fundamentos en esta metodología de modelado.

Los sistemas distribuidos de tiempo real son actualmente muy relevantes debido al bajo coste y a la gran anchura de banda de las redes de comunicación de que se dispone. La arquitectura software de una aplicación que se ejecuta en una plataforma distribuida es semejante a la que se utiliza en un entorno monoprocesador, sin embargo, mientras que en ésta la comunicación entre tareas sólo requiere de los servicios del sistema operativo, en el caso distribuido, cuando los procesos son de nudos diferentes se realiza mediante intercambio de mensajes y requiere también los servicios del software de comunicaciones. El modelo de tiempo real de una aplicación distribuida es mucho más complejo que el de la misma aplicación en entorno monoprocesador, las transacciones son internudos, y para garantizar sus requisitos temporales hay que gestionar un número mayor de recursos y emplear protocolos más complejos [7][8].

Tradicionalmente, las aplicaciones de tiempo real han tenido una arquitectura sencilla que implementaba directamente el modelo transaccional utilizado en su especificación, haciendo uso para ello de los recursos que a tal fin ofrecen los sistemas operativos de tiempo real. Como se muestra en la figura 1, para abordar la complejidad

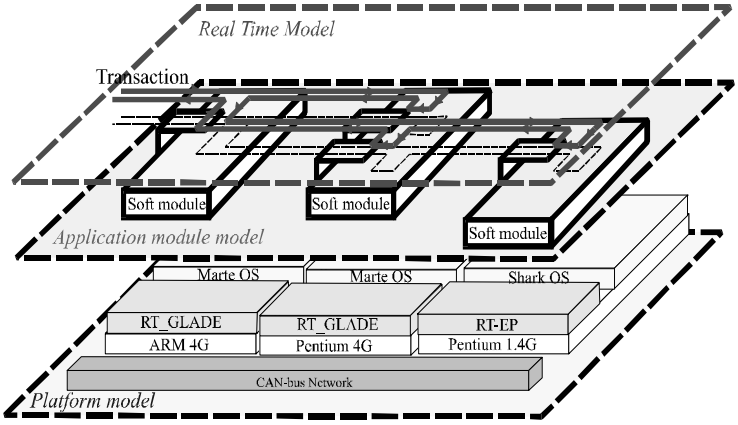

Fig. 1. Niveles de modularización y modelado de aplicaciones de tiempo real

de estos sistemas, gestionar la diversidad de versiones que se generan, y cumplir los plazos que requiere la evolución del mercado, se ha impuesto la componentización de su diseño en todos sus niveles: en los sistemas operativos, en el software de intermediación y en el diseño del código de la propia aplicación [9][10].

La componentización es un patrón estructural que es independiente del proceso de diseño de tiempo real, pero que al introducir cambios profundos en la metodología de desarrollo de las aplicaciones, interfiere con los métodos tradicionales de diseño de tiempo real. En las metodologías tradicionales, las aplicaciones son desde su inicio concebidas y descritas como conjuntos de tareas o transacciones concurrentes dentro de un paradigma de sistema reactivo, y es en una fase posterior, cuando se organiza el código asociando las operaciones utilizadas en paquetes o librerías. Por el contrario, en las metodologías basadas en componentes, el sistema se concibe y diseña como una composición de módulos reusables procedentes de catálogos, y es posteriormente en las fases de refinamiento del diseño, cuando las líneas de flujo de control se identifican y se asocian a procesos que introducen el modelo de concurrencia. En el diseño de los sistemas basados en componentes, hay que hacer compatibles los dos puntos de vista: el estructural (estático) que identifica las operaciones como servicios de las instancias de los componentes y el reactivo (dinámico) donde las actividades (invocación de las operaciones) se organizan por tareas o procesos.

Una metodología de modelado concebida para describir aplicaciones de tiempo real diseñadas utilizando componentes, debe seguir estando orientada a describir el modelo reactivo de la aplicación, pero debe basarse en elementos de modelado que puedan identificarse con la estructura de componentes de la aplicación. A tal fin, se ha de disponer de recursos de modelado para formular el modelo de tiempo real de un componente, como un elemento autocontenido de abstracciones y datos que describan las características de temporización y sincronización incluidas en su código, y que tenga las características de completitud y componibilidad necesarias para que el modelo de tiempo real de la aplicación pueda construirse por composición de los modelos de tiempo real de los componentes que forman parte de ella (de forma análoga a como se compone el código de la aplicación ensamblando los códigos de los componentes).

En trabajos previos [11] [12] se ha propuesto la metodología MAST (Modeling and Analysis Suite for Real-Time Applications) de modelado y análisis de sistemas de tiempo real distribuidos, que sigue el modelo transaccional propuesto por la organización OMG [6]. En este trabajo se describe a nivel conceptual, la estrategia que se utiliza para que los modelos tengan las características de componibilidad que requiere el modelado de sistemas de tiempo real construidos con componentes. En este documento, los modelos se presentan como una extensión de la metodología MAST que hemos denominado "Component-Based System Real-Time Model" (CBS-RTM), pero los conceptos y soluciones que se proponen transcienden de ella y son directamente aplicables a otras metodologías existentes compatibles con el perfil SPT-OMG [6].

2 Estructura del metamodelo CBS-RTM

El metamodelo CBS-RTM define la semántica y los atributos de los elementos de modelado y las formas en que se pueden asociar para constituir un modelo de tiempo real. En la figura 2, se muestra la estructura de paquetes en que se ha organizado los elementos del metamodelo**:**

- El paquete *Core* define los tipos de elementos que constituyen la base conceptual común sobre la que se definen los elementos con los que se construyen los modelos de los componentes y de las aplicaciones. Incorpora los conceptos duales descriptor e instancia de modelo, los elementos raíces para modelar la capacidad de procesamiento de los recursos y el requerimiento de procesamiento de las actividades, y los tipos básicos sobre tiempo, capacidad, prioridad, etc.
- El paquete *Component* define los elementos de modelado que describen el comportamiento temporal de los componentes software, de los recursos de las plataformas hardware/software y de las cargas de trabajo de las aplicaciones.
- El paquete *Application* define los elementos de modelado del comportamiento de tiempo real de las aplicaciones construidas por composición de componentes software de aplicación (business components), desplegados en plataformas de ejecución multiprocesadoras y distribuidas.

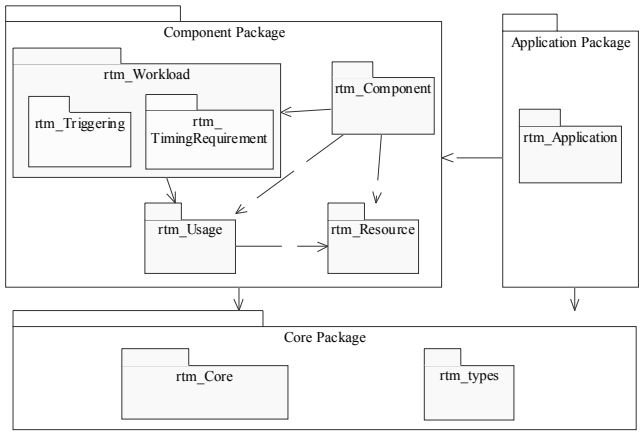

Fig. 2. Paquetes con que se define el metamodelo CBS-RTM

3 Paquete Core: Núcleo del metamodelo y clases de más alto nivel

Los dos conceptos claves de la metodología de formulación de los modelos de tiempo real que se proponen son descriptor (*rtm_Descriptor*) e instancia (*rtm_Instance*) de modelo de tiempo real, que como se muestra en la figura 3, constituyen las clases raíces del metamodelo que describe los modelos de tiempo real que se definen en la metodología CBS-RTM:

- Un *rtm_Descriptor* es una plantilla parametrizada que contiene la información completa relativa a un elemento considerado como clase, que es relevante para que describir el comportamiento temporal de cualquier instancia del mismo que se utilice en la ejecución de una aplicación de la que forme parte. El descriptor proporciona la información semántica y cuantitativa que es común a todos los entes que puedan resultar de su instanciación. Es independiente de la aplicación en la que se utiliza, o de cualquier otro elemento del que requiera hacer uso para implementar su funcionalidad. A tal fin, el descriptor tiene definidos parámetros que referencian, pero dejan indefinidas, características que serán definidas a nivel de instancia.
- Un *rtm_Instance* es una parte del modelo completo de tiempo real de una aplicación, que describe el comportamiento temporal de una instancia concreta de una clase de elemento (recurso o servicio del sistema) en el contexto de un modo de ejecución de la aplicación al que denominamos situación de tiempo real ("*rtm-Situation*"). La rtm_Instance se genera a partir del rtm_Descriptor de la clase de elemento del que es instancia, resolviendo los parámetros que presenta indefinidos, con valores y referencias a instancias de modelo concretas, que son las que existen en el contexto de la RT-Situation que se está modelando.

Las clases rtmd_Descriptor y rtmi_Instance[1] de este metamodelo se corresponden semánticamente y conceptualmente con las clases de igual nombre definidas en la sección General_Resource_Modelling del perfil SPT[6]. En la figura 3 se muestra, que al más alto nivel, estos descriptores e instancias se especializan en tres tipos de clases:

- Las clases *rtmd_Component/rtmi_Component* definen elementos contenedores

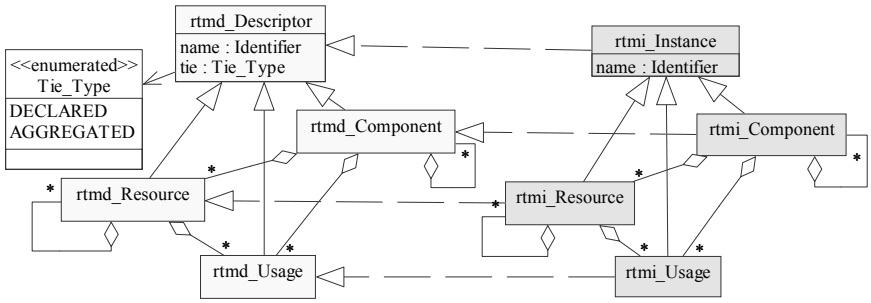

Fig. 3. Clases raíces de más alto nivel del metamodelo

1. Para que sea explícita la naturaleza de los elementos que se definen del metamodelo, se han añadido los prefijos rtmd a los elementos que son descriptores, y rtmi a los que son instancias.

que agrupan todos los elementos de modelado que constituyen el modelo de un módulo hardware o software. Los elementos de estas clases realizan dos funciones: en primer lugar, la propia función de contenedor que sirve para declarar los modelos de los recursos, los modelos de las formas de uso de los recursos hardware/software, y los parámetros y referencias a modelos externos con los que se formula el modelo del módulo. En segundo lugar, define el ámbito de visibilidad de los identificadores, parámetros y elementos de modelado.

- Las clases *rtmd_Resource/rtmi_Resource* definen los modelos de los recursos software (sistemas operativos, mutexes, drivers, etc.) y hardware (procesadores, redes de comunicación, temporizadores, etc.) que describen cualitativamente y cuantitativamente la capacidad de procesamiento de que se dispone y las características con las que puede utilizarse.
- Las clases *rtmd_Usage/rtmi_Usage* describen los modelos de tiempo real de las formas de uso de un servicio en un recurso, esto es, representan una abstracción que describe los recursos y la cantidad de procesamiento de ellos que se utiliza cuando se accede a un servicio de un componente, (por ejemplo ejecutando una operación software, transfiriendo un mensaje por la red, etc.).

4 Paquete Component

El paquete Component contiene las definiciones de las primitivas de modelado que se utilizan para describir el modelo de comportamiento temporal de módulos reusables hardware y software (recursos hardware y software de la plataforma, componentes software de aplicación, elementos de carga de trabajo, etc.). También contiene los elementos contenedores que se necesitan para organizar los modelos. Todos estos elementos de modelado sirven para describir los módulos hardware y software que son parte de una aplicación de una forma autocontenida y reusable.

El paquete *rtm_Component* define los elementos contenedores que permiten organizar los modelos con una estructura similar a la que tiene el sistema y la aplicación. La figura 4 muestra los elementos contenedores definidos en el metamodelo. El elemento básico es rtmd_Component, que sirve para agrupar en un elemento los modelos de los recursos y de las formas de uso de los recursos que constituyen el modelo de un módulo reusable hardware/software. También pueden contener recursivamente otros rtmd_Component, lo que posibilita definir modelos de módulos en función de los modelos de los elementos de que se compone. En el metamodelo, se incluye un tipo de contenedor específico de modos de uso de un recurso que se denomina *rtmd_Interface*,

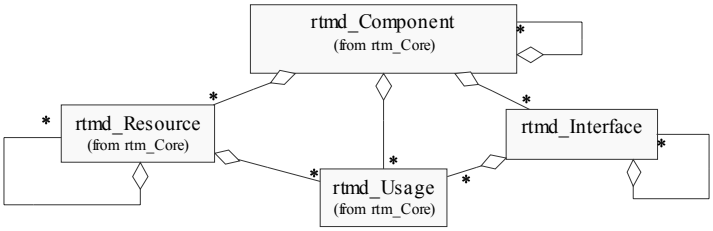

Fig. 4. Elementos contenedores definidos en el metamodelo CBS-RTM

que se utiliza para definir de forma estructurada los modelos de grupos de procedimientos que constituyen un mismo servicio ofrecido por diferentes componentes.

El paquete *rtm_Resource* define los elementos de modelado de tiempo real que tienen como elemento raíz la clase rtmd_Resource y que se utilizan para describir la capacidad de procesamiento que proporcionan los diferentes recursos de la plataforma que ejecuta las actividades que constituyen la aplicación, así como las limitaciones a su uso que introducen los elementos de sincronización que se necesitan para planificar el acceso concurrente de las actividades a recursos de acceso limitado. La figura 5 muestra el conjunto de elementos de modelado definidos:

- *rtmd_ProcessingResource*: modela la capacidad de un elemento hardware para llevar a cabo actividades de la aplicación, tales como ejecución de segmentos de código o transferencia de mensajes.
- *rtmd_Processor*: modela la capacidad de un procesador para ejecutar código.
- *rtmd_Network*: modela la capacidad de una red de comunicación para transferir mensajes entre procesadores.
- *rtmd_Timer*: modela el consumo de capacidad de procesamiento del procesador que conlleva la gestión del temporizador, así como la granularidad del tiempo en la actividades temporizadas.
- *rtmd_Driver*: modela el consumo de capacidad de procesamiento del procesador que se emplea en la gestión del envío y recepción de mensajes por una red de comunicación.
- *rtmd_PrimaryScheduler/rtmd_SecondaryScheduler*: modela el consumo de capacidad de procesamiento que se emplea en la gestión del planificador. Un planificador representa el elemento del sistema operativo que gestiona la distribución de la capacidad de procesamiento entre las tareas en espera de ser ejecutadas. El metamodelo prevé una estructura jerárquica de planificadores: los planificadores primarios gestionan directamente la capacidad de un recurso de procesamiento hardware y los planificadores secundarios gestionan en segundo nivel la capacidad de procesamiento que ha sido asignada a un proceso. Estos elementos también

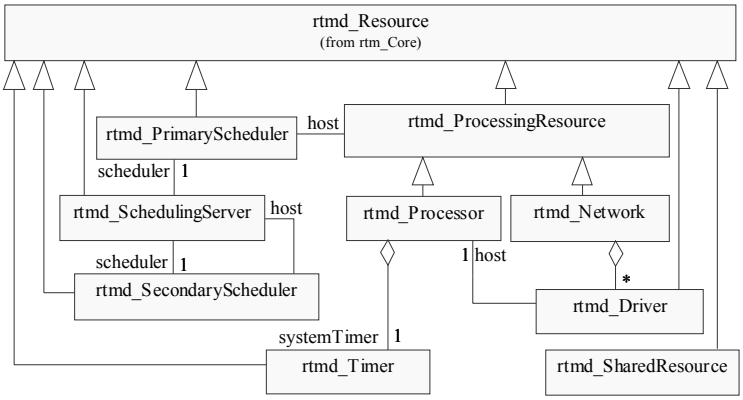

Fig. 5. Principales clases relativas al modelado de elementos de recursos de ejecución

modelan la política de planificación que cualifica la estrategia con la que se distribuye la capacidad de procesamiento. La metodología permite modelar diferentes políticas específicas de planificación basadas en prioridades fijas, EDF, o protocolos de transferencia de mensajes basados en paquetes.

- *rtmd_SchedulingServer*: modela entidades de planificación tales como procesos, tareas, hilos de un procesador o sesiones de comunicación de una red. Cada uno tiene agregado una estructura específica de datos que indica al planificador la características de su planificación. Se permiten políticas de planificación basadas en prioridades, EDF, de interrupción, servidores esporádicos, etc.
- *rtmd_SharedResource*: modela un recurso de sincronización entre procesos que deben acceder a un recurso compartido en régimen de exclusión mutua. Sólo se admiten protocolos que impiden inversión de prioridad tales como el techo inmediato de prioridad, herencia de prioridad o protocolo basado en pila (SRP).

El paquete *rtm_Usages* contiene los elementos de modelado que se utilizan para describir la capacidad de procesamiento que usan las actividades que se ejecutan dentro de una aplicación. Todos los elementos definidos en este paquete tienen como elemento raíz la clase rtmd_Usage. Corresponden, por ejemplo, al modelo de la ejecución del código de un procedimiento lógico de un componente, al modelo de las actividades que se realizan en background (p.e. la gestión del timer) o el modelo de la respuesta a una interrupción hardware o temporizada. En la figura 6 se muestran las clases raíces definidas en el metamodelo para modelar los usos de los recursos:

- *rtmd_Operation*: modela la ejecución de una sección de código o la transferencia de un mensaje. Específicamente describe la cantidad de procesamiento que requiere la ejecución de una actividad que se planifica en un mismo proceso, sin requerir sincronización con otros procesos. Declara los recursos compartidos a los que debe haber accedido para poder realizarse, y los recursos que libera tras su conclusión. En el metamodelo se definen diferentes operaciones especializadas:
  - *rtmd_SimpleOperation*: modela la ejecución de una sección simple de código. Tiene como atributos los tiempos de ejecución de peor, mejor y caso promedio, formulados como valores normalizados a la capacidad de un procesador de referencia.

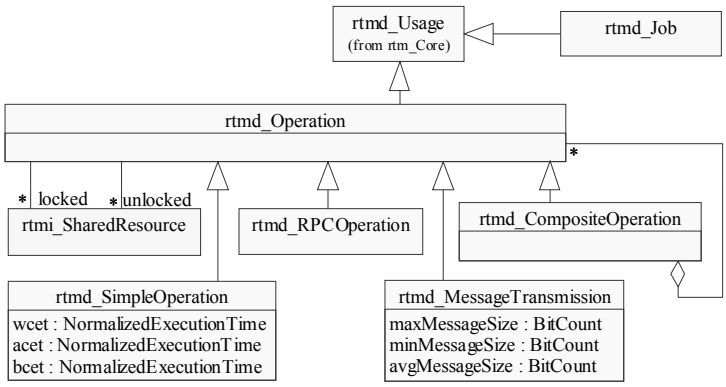

Fig. 6. Clases raíces definidas en el paquete rtmd_Usages.

- *rtmd_MessageTransmission:* modela la transferencia de un mensaje simple por una red.
- *rtmd_CompositeOperation*: modela una operación compleja compuesta por una secuencia de los diferentes tipos de operaciones.
- *rtmd_RPCOperation*: modelan el uso de un recurso que se invoca remotamente, y complementa su modelo con todas aquellas características propias del uso del recurso (y no del mecanismo con el que se realice la invocación) que son necesarias para modelar la invocación remota. En la figura 7 se muestra la correspondiente sección del metamodelo, y los nuevos datos que se incorporan: las operaciones de construcción (marshalling) y decodificación de los mensajes (unmarshalling) y las características de los mensajes de invocación y de retorno de resultados.

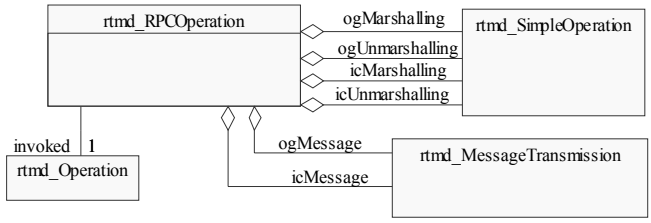

Fig. 7. Elementos del modelo de una operación remota.

- *rtmd_Job*: Modelan el uso de un conjunto de recursos que se realiza cuando se invoca un procedimiento que ejecuta un conjunto de actividades relacionadas entre sí por flujo de control, pero que se ejecutan en múltiples servidores de planificación. El caso típico de uso de recurso que se modela con este elemento es la invocación de un servicio de un componente, que para su implementación requiere la invocación de servicios de otros componentes locales o remotos, y cuyos threads han de sincronizarse. Como se muestra en la figura 8, la descripción de un Job es compleja ya que involucra conjuntos de operaciones, scheduling server, eventos y elementos de control (fork, joint, branch and merge).

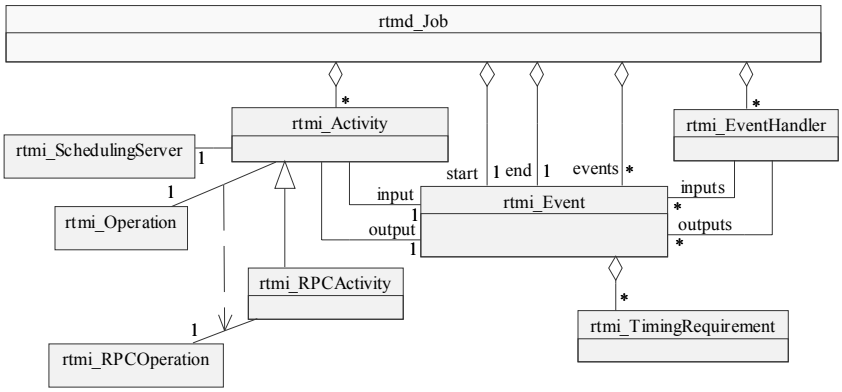

Fig. 8. Descripción del modelo de un job.

El paquete *rtm_Workload* describe los recursos que se han definido para describir los elementos del modelo reactivo que pueden ser gestionados por un componente de la aplicación. La carga de trabajo de un sistema de tiempo real se describe mediante un conjunto de transacciones, cada una de ellas modelada por un elemento del tipo rtmd_Transaction. Cada transacción describe la secuencia de actividades que se deben ejecutar como respuesta a un evento del entorno o temporizado. Asimismo, una transacción contiene la descripción del patrón de generación de los eventos que la invocan, y también los requisitos temporales que se requieren en su ejecución.

La transacción se modela mediante un grafo en el que los nudos son actividades y manejadores de eventos y los arcos son eventos entre ellos, y en conjunto representa las relaciones de flujo que existen entre las actividades que la constituyen. Como muestra la figura 9, una transacción se define por medio de tres tipos de elementos::

- *rtmi_Event*: representa una dependencia de flujo entre actividades. Puede tener asociado un elemento *rtmi_TimingRequirement*, que representa un requisito temporal que debe satisfacer la generación del evento. Se definen diferentes tipos de requisitos temporales relativos a plazos (deadline), a fluctuaciones (jitter) y al porcentaje de fallos que se admite (miss ratio). Asimismo, pueden hacer referencia a un evento externo (global) o al evento que invoca la actividad (local).
- *rtmi_ExternalEvent*: modela una secuencia de eventos que invocan la transacción, ya sean procedentes del entorno o temporizados generados por el reloj. Describe probabilísticamente los tiempos en que se generan, utilizando diferentes patrones como singulares, periódicos, esporádicos, en ráfaga, etc.
- *rtmi_EventHandlers*: representan acciones que son activadas con la llegada de un evento, y que a su vez son origen de nuevos eventos. Hay dos tipos de manejadores de eventos. Las actividades representan la ejecución de una operación por un ente de planificación en un recurso. Los otros manejadores de eventos representan diferentes opciones de distribución de eventos entre actividades, tales como retrasos (delay), invocación concurrente (fork), invocación alternativa (branch), convergencia sincronizada (joint) o convergencia alternativa (merge).

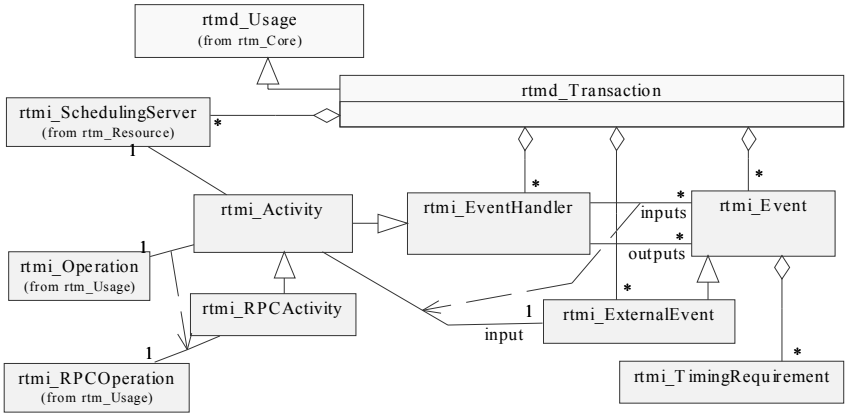

Fig. 9. Elementos de modelado de una transacción

5 Application Package.

El paquete Application ofrece los recursos necesarios para formular el modelo de una aplicación cuya configuración se ha realizado agregando componentes de tiempo real, y que se encuentra desplegada en una plataforma compuesta por recursos hardware/ software de los que se dispone de su modelo de tiempo real.

En la figura 10 se muestra que el modelo de tiempo real de una aplicación se compone del modelo de cada una de sus situaciones de tiempo real. Cada *rtmi_RTSituation* modela un modo de operación de la aplicación, desplegada sobre una determinada plataforma de ejecución, en la que hay definidos requisitos de tiempo real. Cada situación de tiempo real constituye un modelo completo, analizable e independiente, que puede ser procesado por las herramientas de análisis de planificabilidad y estimación de la respuesta temporal.:

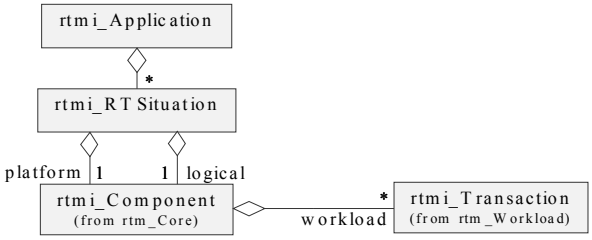

Fig. 10. Clases para el modelado de una aplicación.

La formulación del modelo de una situación de tiempo real conlleva dos tareas complementarias:

- Declaración de las instancias de los modelos de los componentes software y de los recursos de la plataforma que participan en la aplicación, lo que se corresponde con la formulación estructural del modelo de tiempo real de la aplicación.
- Declaración de las instancias de las transacciones que constituyen la carga de trabajo de la aplicación en ese modo, lo que se corresponde con la formulación del modelo reactivo de la aplicación.

6. Ejemplo de formulación del modelo de tiempo real de una aplicación.

A fin de ilustrar los conceptos que se han definido a través del metamodelo CBS_RTM, se propone un ejemplo sencillo consistente en una aplicación distribuida y de tiempo real cuya función es leer diferentes señales digitales, y cuando su estado no sea el correcto, generar ciertas acciones de alarma consistentes en establecer otras líneas digitales y generar sonidos en un altavoz. En la figura 11.a se muestra el modelo lógico de la aplicación, basado en tres clases de componentes software:

- Agent: Es un componente de tipo cliente activo que controla la ejecución concurrente de múltiples tareas de verificación de alarma utilizando un hilo de flujo independiente para cada una de ellas.
- IO_Card: En un componente pasivo que ofrece la interfaz I_Digital, a través de la

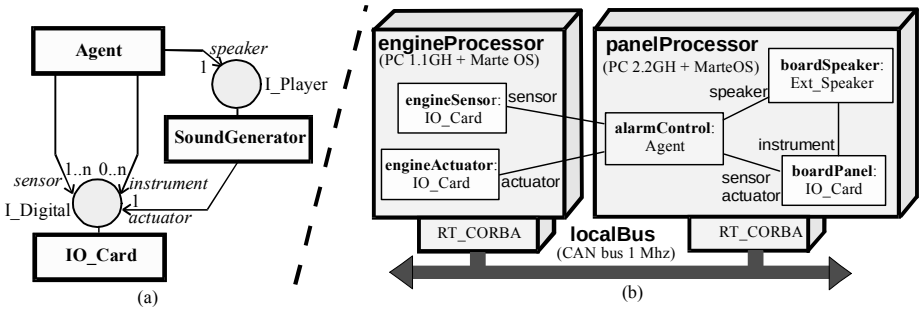

Fig. 11. Ejemplo carAlarm: (a) Arquitectura software. (b) Plataforma y despliegue

que los clientes pueden leer una línea digital de entrada (función readState) y establecer una línea digital de salida (procedure writeState).

- SoundGenerator: Es un componente que ofrece la interfaz I_Player, a través del que un cliente puede generar sonidos de alarma (procedure play). Para generar el sonido utiliza un conjunto de líneas digitales con las que controla el dispositivo que genera físicamente el sonido.

En la figura 11.b se muestra el despliegue de la aplicación carAlarm que va a ser objeto de modelado. La plataforma de ejecución está compuesta de dos nudos denominados panelProcessor y engineProcessor operando con MaRTE OS, que se comunican entre sí a través del bus CAN de comunicaciones localBus. La aplicación utiliza los recursos de distribución RT Corba. Los componentes alarmControl de tipo Agent, el componente boardSpeaker de tipo Ext_Speaker y el componente boardPanel del tipo IO_Card en el procesador paneProcessor y los componentes engineSensor y engineActuator, ambos del tipo IO_Card en el procesador engineProcesor.

El planteamiento que se realiza en este trabajo, es la elaboración del modelo de tiempo real de la aplicación en dos fases. Cuando se adquieren los componentes software (Agent, IO_Card y Ext_Speaker), se registran sus modelos de tiempo real, junto con su código y la información introspectiva asociada (metadata). Asimismo, cuando se desarrollan aplicaciones utilizando ciertas plataformas hardware y software, se formulan y validan los modelos de tiempo real que caracterizan su comportamiento. Cuando se desarrolla una nueva aplicación, se elabora el modelo de sus situaciones de tiempo real, componiendo instancias de los modelos de los componentes y de los recursos de la plataforma que han sido definidas de acuerdo con el contexto de la apli-

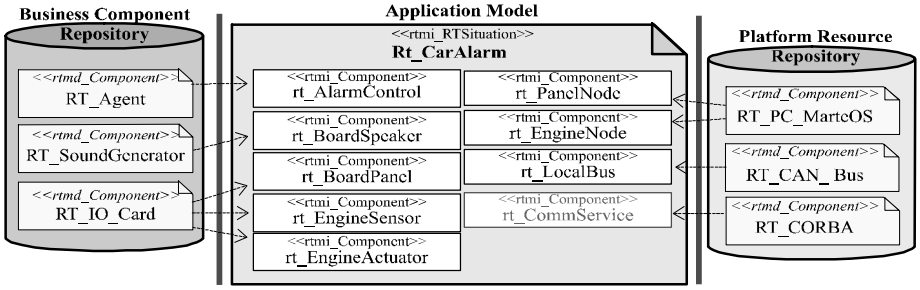

Fig. 12. Descriptores e instancias del modelo de la aplicación carAlarm

cación. En la figura 12, se muestran los descriptores y las instancias que participan en el modelado de la aplicación carAlarm cuyo despliegue muestra la figura 11.

La formulación del modelo de tiempo real del ejemplo es muy simple, y resulta de instanciar los modelos de los módulos hardware y software que intervienen en la aplicación (modelo estructural), y de instanciar los modelos de las transacciones que concurren en la situación de tiempo real que se modela..

Para formular el modelo el modelo de tiempo real de la aplicación, el diseñador tiene que realizar dos pasos: declarar las instancias haciendo referencia a los descriptores de los que son instancias y que se encuentran almacenados en el Registro, y asignar valores a los parámetros definidos en ellos, de acuerdo con el contexto de la situación que se modela. En la figura 13, se muestra el modelo de la aplicación del ejemplo que se ha descrito. El modelo de tiempo real de la aplicación tiene como elemento raíz *carAlarmApplication*. El modelo contiene el modelo de una única situación de tiempo real *carAlarm*. En el modelo de la plataforma *platform* se declaran los modelos de dos nudos procesadores, de una red de comunicaciones bus y de un servicio de comunica-

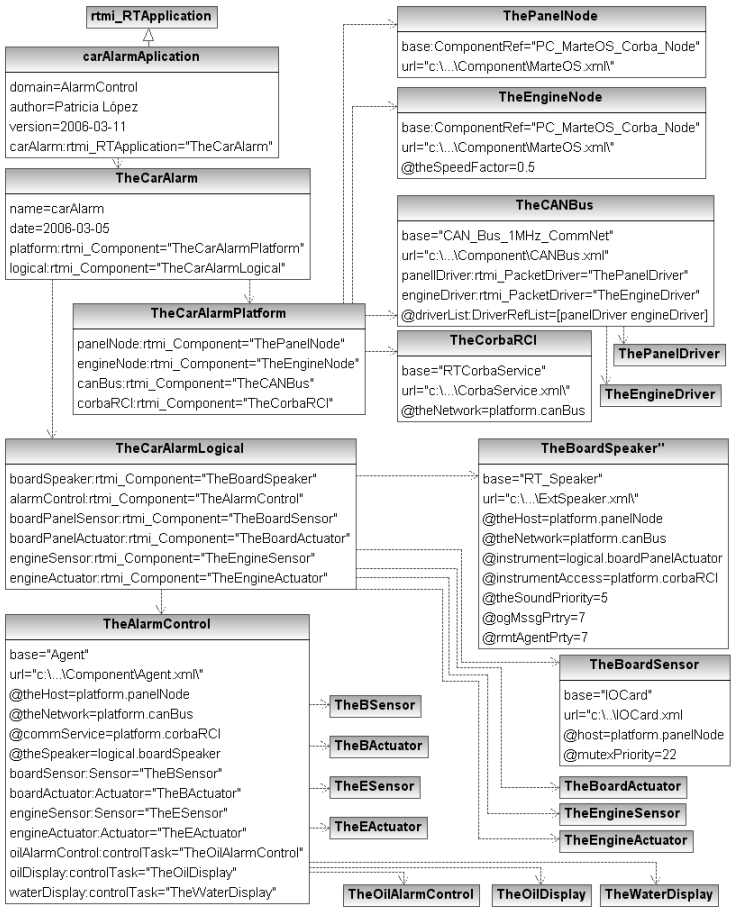

Fig. 13. Formulación UML del modelo de la aplicación carAlarm

ciones rci. Cada nudo procesador, representa el modelo del propio procesador, de su reloj, de su planificador y de los recursos de comunicación necesarios para mantener la comunicación CORBA. La red de comunicaciones representa el modelo de tiempo real del propio bus CAN, y de los drivers instalados en sendos procesadores para su gestión. El modelo de la propia aplicación se encuentra ubicado en el elemento *logical*. Se compone de los modelos de las instancias que participan en la ejecución de la situación de tiempo real: *boardSpeaker*, *alarmContro*l, e*ngineSensor*, *engineActuator*, *boardPanelSensor* y *boardPanelActuator*. Estos dos últimos son modelos del mismo componentes instanciado boardPanel en su doble papel de sensor y actuador.

La declaración de una instancia de un descriptor de un componente se realiza mediante dos atributos: url que hace referencia al fichero del Registro en el que se encuentra el modelo, y base que identifica el elemento de modelado que se referencia. Al declarar una instancia se ha de asignar valores a todos los parámetros que tenía definidos el descriptor.

En el modelo de la figura 13, la carga de trabajo se ha realizado declarando las instancias de las transacciones oilAlarmControl, oilDisplay y waterDisplay, en el modelo del componente activo alarmControl que es responsable de su gestión.

La sencillez de la elaboración del modelo de una aplicación, contrasta con la complejidad de los modelos de los componentes, en los que no sólo se ha de describir el comportamiento temporal del recurso hardware o del componente software, sino que ha de formularse de forma parametrizada. En la figura 14, se muestra el modelo del descriptor del componente pasivo IOCard, cuya función es el control de una tarjeta hardware de adquisición de datos. Para simplificar la figura, sólo se describe de forma completa el modelo de procedimiento writeState.

El modelo del componente software se formula como el rtmd_Component IOCard. En el modelo de la figura se modelan dos procedimientos readState y writeState que corresponden a las operaciones que permiten leer el estado o establecer el estado de una línea digital controlada por la tarjeta de IO. Estos procedimientos pueden ser invocados remotamente a través de CORBA (o de otro software de comunicaciones), por lo que se describen mediante rtmd_RPCOperation y rtmd_APCOperation respectivamente. En el caso de la operación writeState que es APC, el modelo describe tanto la

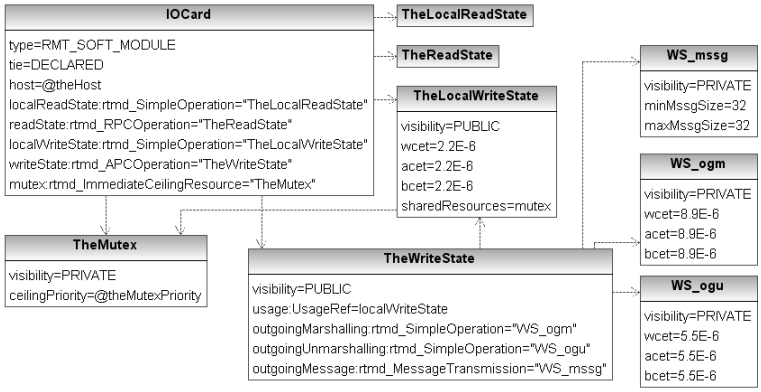

Fig. 14. Modelo del descriptor del componente software IO_Card

temporización de la ejecución del código que la lleva a cabo localmente (localReadState), como también la temporización de la operación de serialización de los argumentos de entrada (outgoingMarshalling), de recuperación de los mismos (outgoing Unmarshalling) y las características del mensaje dependientes de la operación invocada, que se requiere para codificar sus datos de entrada. En el caso de una operación RPC, que implica un mensaje de retorno, también se incluyen la temporización de las operaciones de serialización y recuperación de los datos de retorno (incomingMarshalling y incomingUnmarshalling) y del mensaje de retorno (incomingMessage).

Este modelo sólo declara dos parámetros: @theHost que hace referencia al nudo en el que se instala cada instancia que se modele, a través de la que se accede a las características de capacidad de procesamiento y de planificación, y @theMutexPriority que corresponde al techo de prioridad que se asigna al mutex que permite invocar de forma concurrente sus operaciones. A ambos se deben asignar valores cuando se modele una instancia del componente (véase por ejemplo el modelado de la instancia boardSensor en la figura 13).

7. Conclusiones

En este trabajo se describe una estrategia para formular el modelo de tiempo real de un sistema mediante la composición de los modelos de tiempo real que describen el comportamiento temporal de los módulos hardware y software de que se compone el sistema. Con esta estrategia se facilita el diseño de aplicaciones de tiempo real basadas en componentes ya que permite asociar modelos de comportamiento temporal a los componentes software que son independientes de la aplicación en que se utilizan y por tanto son reusables. Asimismo, se simplifica el diseño de sistemas de tiempo real distribuidos, en los que se debe evaluar el comportamiento de una aplicación en diferentes plataformas de ejecución y con diferentes configuraciones de distribución.

La clave de la estrategia de modelado es el uso de los conceptos de descriptor e instancia de modelo. El primero es una plantilla parametrizable que permite formular las características de comportamiento de un módulo hardware o software de forma autocontenida, supliendo la dependencia de otros módulos por referencias que quedan indeterminadas. El segundo es el modelo concreto y final que describe el comportamiento temporal de una instancia software o de un módulo hardware en el contexto de un modo de operación del sistema que se analiza. La instancia del modelo se deriva del descriptor de modelo, asignando a los parámetros de éste, bien valores concretos o referencias a otras instancias concretas definidas en el propio modelo.

Esta metodología se ha utilizado para simplificar el modelado de tiempo real de sistemas distribuidos complejos, reutilizando los modelos de plataformas de ejecución (sistemas operativos, protocolos de comunicación, servicios de intermediación etc.) y de componentes software (librerías de funciones, servidores remotos, drivers, etc.). Utilizándola como base, se está actualmente desarrollando herramientas que simplifiquen el diseño y la validación de sistemas de tiempo real basados en componentes.

References

[1] J. Liu, Real-Time Systems. ISBN 0-13-099651-3. Prentice Hall Inc., 2000.

[2] H. Kopetz, R. Zainlinger, G. Fohler, H. Kantz, P. Puschner, and W. Schutz, ”The design of real-time systems: from specification to implementation and verification” Software Engineering Journal, Vol.6 , no. 3, pp. 72-82. May, 1991.

[3] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. González Harbour, A Practitioner's Handbook for Real-Time Systems Analysis, Kluwer Academic Pub., 1993.

[4] A. Cheng, “Real-time Systems Scheduling, Analysis and Verification”. ISBN 0-471-18406-3. John Wiley & Sons Inc., New Jersey, 2002

[5] IEEE Std 1003.13™-2003: ”IEEE Standard for Information Technology-Standarized Application Environment Profile (AEP)-POSIX® Realtime and Embedded Application Support”, 2003.

[6] Object Management Group: "UML Profile for Schedulability, Performance and Time Specification", Version 1.0. OMG document formal/03-09-01, September, 2003.

[7] J.J. Gutiérrez, M. González Harbour. “Optimized Priority Assignment for Tasks and Messages in Distributed Real-Time Systems”, Proc. of the 3rd Workshop on Parallel and Distributed Real-Time Systems, California, 1995

[8] K. Tindell and J. Clark, Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. Microprocessing & Microprogramming, Vol 50, N0s. 2-3, 1994.

[9] J. Stankovic. "VEST: a toolset for constructing and analyzing component based operating systems for embedded and real-time systems", Proc. of the Embedded Software, First International Workshop (EMSOFT 2001), Tahoe City, CA, USA, October 2001.

[10]A. Tesanovic, D. Nyström, J. Hansson, and C. Norström: "Aspects and Components in Real-Time System Development: Towards Reconfigurable and Reusable Software", J. of Embedded Computing, February, 2004.

[11]M. González Harbour, J.J. Gutiérrez, J.C.Palencia and J.M.Drake: "MAST: Modeling and Analysis Suite for Real-Time Applications" Proc. of the ECRTS June 2001.

[12].L. Medina, M.González Harbour, J.M. Drake: "MAST Real-time View: A Graphic UML Tool for Modeling Object_Oriented Real_Time Systems", RTSS, December, 2001.

[13]MAST: Modeling and Analysis Suite for Real-Time Applications. “http://mast.unican.es” and “http://mast.unican.es/umlmast”.