ADA_MAST: Herramienta para el Desarrollo de Aplicaciones Ada Distribuidas de Tiempo Real [1]

Julio L. Medina, José M. Drake, J. Javier Gutiérrez, Michael González Harbour

Avda. de los Castros s/n, 39005 Santander - España
{medinajl, drakej, gutierjj, mgh}@unican.es

Resumen. Se describe una metodología para el modelado de tiempo real de aplicaciones Ada 95 construidas de acuerdo con los anexos de tiempo real (D) y distribuido (E) del estándar. El modelo se formula utilizando UML y puede ser soportado por herramientas CASE estándar orientadas a objetos. El modelo de tiempo real que resulta es apropiado para ser procesado por el conjunto abierto de herramientas que ofrece el entorno MAST. Este modelo representa independientemente la plataforma que ejecuta la aplicación, los componentes lógicos software con que se construye y las situaciones o escenarios de tiempo real con que puede operar y que son el objeto del análisis. A través de un ejemplo de sistema distribuido, se muestra como utilizar la metodología y las herramientas en fases de diseño de la aplicación (por ejemplo para la asignación óptima de prioridades) y en fases de análisis (por ejemplo, estudio de su planificabilidad, cálculo de holguras, etc.). El modelo de la invocación de componentes distribuidos, es gestionado de forma automática por la herramienta, y es establecido de forma diferente si la invocación es local o remota..

1. Introducción

Este trabajo propone una metodología para describir con UML sobre una herramienta CASE, el modelo de tiempo real de aplicaciones Ada distribuidas. Los elementos de modelado que se definen, permiten modelar aplicaciones distribuidas sobre una implementación de Ada 95 que soporte los anexos D de sistemas de tiempo real y E de sistemas distribuidos del estándar [1] y proporcionan un modelo que puede ser procesado por el conjunto de herramientas que se ofrecen en el entorno MAST (Modeling and Analysis Suite for Real-Time Applications) [8][9]. La reusabilidad de módulos software a través de la programación orientada a objetos es una de las ventajas que supone el uso de Ada en aplicaciones de tiempo real. Sin embargo, para diseñar módulos software de tiempo real reusables, se necesita tener capacidad para modelar su comportamiento de tiempo real, al igual que se describe su funcionalidad. Cuando se modelan aplicaciones que además son distribuidas, se tiene que incluir en el modelo el efecto de las comunicaciones y planificar la transferencia de los

[1] Este trabajo está financiado por la Comisión Interministerial de Ciencia y Tecnología mediante los proyectos TIC99-1043-C03-03 y 1FD 1997-1799 (TAP)

mensajes a través de la red de comunicaciones al igual que se planifican las actividades que se ejecutan en los procesadores.

Aspectos relevantes de la propuesta que se presenta son:

- Tal como se sugiere en [9], plantea modelos independientes para la plataforma (procesadores, redes de comunicación, recursos del sistema operativo, dispositivos externos, etc.), los componentes lógicos de la aplicación (requisitos de procesamiento, recursos comunes, otros componentes internos), y las situaciones de tiempo real que definen la funcionalidad esperada del sistema en sus diversos modos de operación (secuencias de eventos, transacciones de tiempo real, carga de trabajo, requisitos temporales).
- Modela el comportamiento de tiempo real de las entidades lógicas de Ada (temporización, concurrencia, sincronización, etc.), de manera que el modelo de tiempo real sigue la misma estructura del código Ada original.
- Facilita la generación del modelo de los diversos componentes lógicos de más alto nivel mediante la instanciación de modelos genéricos parametrizados. Cuando se combinan los modelos de todos los componentes que intervienen en cada situación de tiempo real, y se asigna valores concretos a los parámetros del modelo, se obtiene el modelo analizable de todo el sistema para esa situación de tiempo real.
- El modelo de comunicaciones que representa la codificación, transmisión, entrega y decodificación de argumentos en las llamadas a procedimientos remotos, se incorpora de manera automática cuando un procedimiento que se declara como remoto se invoca desde un componente que se encuentra asignado a un nodo diferente.
- Al igual que la descripción funcional y estructural de los componentes del software de la aplicación, el modelo de tiempo real se describe en UML, y se hace dentro de una nueva vista de la aplicación. Los componentes del modelo, así como su interconectividad se han descrito formalmente mediante un metamodelo.

En el siguiente apartado se apuntan algunos antecedentes de la metodología que proponemos y las herramientas sobre las que se apoya. En el apartado 3 se muestra a través de secciones del metamodelo, los tipos de componentes de modelado básicos, su semántica y posibilidades de asociación. En el apartado 4 se plantean algunos aspectos sobre la correspondencia de módulo construidos con sentencias Ada y su modelado. Un ejemplo de diseño y validación de una aplicación a través de la metodología se presenta en el apartado 5. Finalmente resumimos algunas conclusiones y líneas de trabajo futuro.

2 Antecedentes, herramientas y técnicas de soporte

Tanto desde el punto de vista semántico como conceptual, el metamodelo que soporta la descripción en UML de la metodología que se propone, es una especialización del propuesto con la herramienta UML-MAST [9] que fue desarrollado para modelar aplicaciones diseñadas con tecnologías orientadas a objetos. Y ésta a su vez es una especialización de la herramienta MAST [8], propuesta para el modelado de sistemas de tiempo real más generales, gobernados por eventos. El conjunto de herramientas disponibles es compartido por todas ellas y actualmente se dispone de herramientas

para realizar análisis de planificabilidad (tanto por el método basado en offsets[11][12] como por el método clásico), realizar asignación óptima de prioridades (por los métodos de templado simulado y Linear HOPA) o calcular holguras. Estas herramientas son además aplicables tanto a sistemas mono y multiprocesador, como a sistemas distribuidos, y permite emplear estrategias de planificación basadas en prioridades fijas, expulsoras y no expulsoras, planificación de rutinas de interrupción, servidores esporádicos y servidores de atención periódica.

Desde su inicio, MAST fue concebido como un entorno abierto que proporciona las especificaciones y los recursos necesarios para que los investigadores que trabajan en tiempo real puedan integrar y comparar las nuevas herramientas que desarrollen.

Por otra parte, el concepto de **Componente,** que es central en el modelo, corresponde al que se propone en [13] como elemento principal para el modelado de componentes de tiempo real. En la metodología que se presenta se le ha especializado para representar los tipos básicos de estructuras contenedoras de componentes Ada.

En cuanto al tipo de aplicaciones que se pretende modelar, debemos destacar que los anexos D y E del estándar de Ada 95 han sido pensados y descritos de manera independiente, por lo cual el desarrollo de aplicaciones distribuidas que a la vez deban satisfacer requisitos de tiempo real no está formalmente soportado en Ada 95 [4]. Nuestro grupo ha propuesto un esquema de priorización para las llamadas a procedimientos remotos en [6] y ha apuntado ciertas características de interés para facilitar el desarrollo de sistemas distribuidos de tiempo real en Ada en [7], que lo harían potencialmente más eficiente y fácil de utilizar que otros estándares como CORBA de tiempo real. Por tanto, para que los mecanismos de modelado y análisis que se proponen aquí sean aplicables, presuponemos que la plataforma sobre la que se implementa la aplicación a modelar satisfaga las propuestas de [6] y [7]. Actualmente los compiladores Ada que implementan el anexo D para sistemas monoprocesadores son bastante utilizados, sin embargo, el anexo E de sistemas distribuidos va siendo implementado de manera más paulatina; una de estas implementaciones es GLADE [3], la cual es actualmente parte del compilador GNAT, desarrollado por Ada Core Technologies (ACT) [5] y que es además de distribución libre.

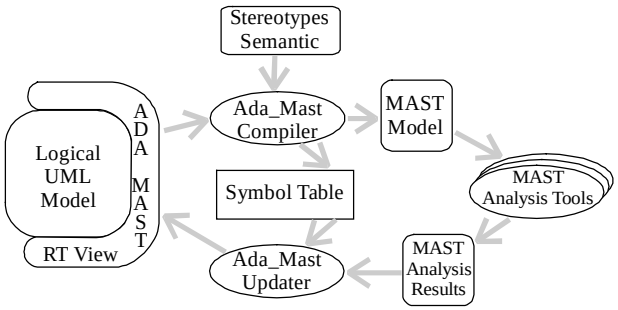

Fig. 1. Forma en que se implementa el paradigma del procesado de modelos con ADA_MAST

En ADA_MAST se sigue un paradigma de procesado de modelos similar al que se describe en [14]. La figura 1 ilustra este proceso. Para procesar el modelo, se compila el modelo UML de tiempo real a un modelo mas general basado en el entorno MAST (MAST Model). La compilación es necesaria ya que con la especialización de la

metodología se incrementa la carga de información semántica, y la generación del modelo de bajo nivel no sólo requiere una codificación de la información contenida en el modelo, sino una incorporación de los patrones introducidos a través de la semántica de los componentes que se utilizan. La compilación genera también una tabla de símbolos que será utilizada para hacer corresponder los resultados que se obtengan de la aplicación de las herramientas con los entes del modelo original a que corresponden. Estos resultados se incorporan al modelo UML en la vista de tiempo real, de manera que el modelo va evolucionando hasta representar la situación de tiempo real del sistema tal como el diseñador la requiere. Los elementos de la vista de tiempo real se designan por medio de estereotipos asignados a elementos estándar UML.

3 Metamodelo de la vista de tiempo real

Siguiendo la dinámica de representación de UML_MAST, el modelo de tiempo real de una aplicación basada en componentes se puede presentar en tres secciones, que modelan separadamente la plataforma, los componentes lógicos y las situaciones de tiempo real bajo análisis. En [13] se describe el metamodelo de cada sección con cierto nivel de detalle, en este apartado presentamos una especialización de este metamodelo que permite soportar de manera sencilla el modelado de aplicaciones Ada sobre la base de los conceptos ofrecidos por MAST.

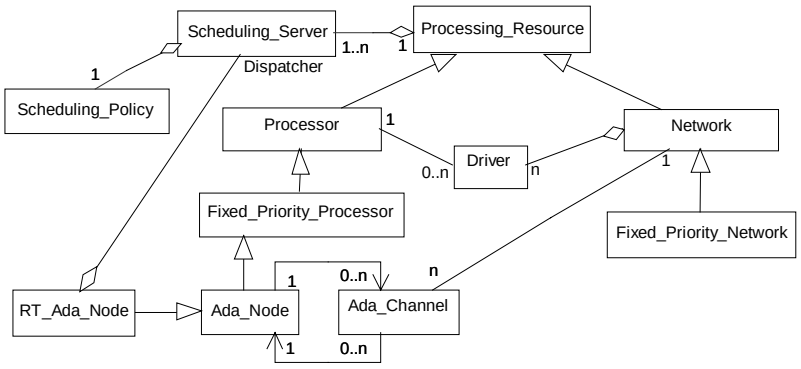

Fig. 2. Sección del metamodelo que describe el modelo de la plataforma

El 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 son recursos tales como procesadores (lo que incluye: capacidad de procesamiento, planificador, threads, temporizadores) y redes de comunicación (capacidad y modo de transmisión, planificación de mensajes, sobrecarga debida a los drivers), y la forma en que están interconectados (qué procesadores comunican a través de qué redes de interconexión).

En la figura 2 se muestra un extracto del metamodelo de la plataforma tal como se define para la metodología que se presenta.

La clase Ada_Node, representa un tipo de procesador basado en prioridades fijas, que puede disponer de canales de comunicación que le enlacen a otros procesadores a fin de invocar procedimientos de acceso remoto en ellos. La clase RT_Ada_Node es equivalente a la anterior, pero además puede exportar estos servicios al disponer de un thread destinado a invocar localmente estos procedimientos una vez demandados desde el exterior.

Modelo de los componentes lógicos: describe el comportamiento de tiempo real de los componentes Ada empleados en la aplicación. Por componente software se designa aquí a cualquiera de los módulos mediante los cuales se distribuyen las operaciones lógicas de la aplicación. Se emplea para modelar paquetes, bien sean pasivos (de librería) o clases (tagged types), tareas, el programa principal (main), etc.

El modelo de un componente software describe: la cantidad de procesamiento que requiere la ejecución de las operaciones de su interfaz, los componentes lógicos de los que requiere servicios, los procesos o threads que crea para ejecutar sus operaciones, los mecanismos de sincronización que requieren sus operaciones, los parámetros de planificación definidos explícitamente en el código y los estados internos relevantes a efecto de especificar y validar requisitos de tiempo real. En la figura 3 se muestra la estructura jerárquica de componentes Ada propuesta.

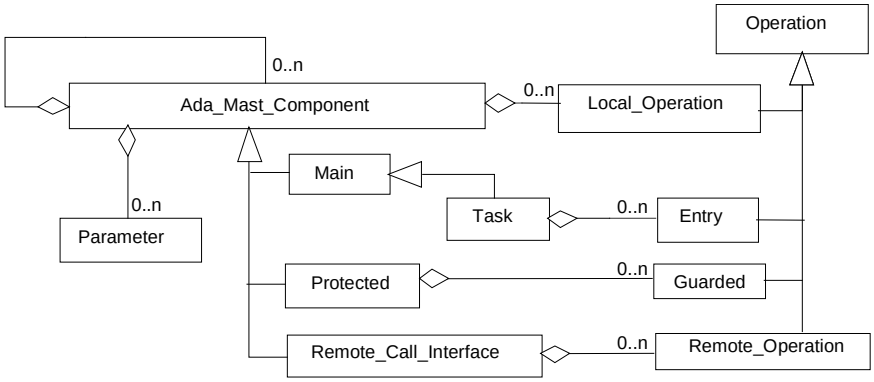

Fig. 3. Sección del metamodelo de componentes lógicos de ADA_MAST.

Ada_Mast_Component es la clase principal del modelo de componentes lógicos de ADA_MAST y agrupa los modelos de comportamiento temporal de todas las operaciones definidas en la interfaz del componente (especificación Ada); para las operaciones simples, emplea un conjunto de atributos que indican la cantidad de procesamiento que requieren del procesador en que se ejecutan, mientras que si son compuestas, su modelo representa la secuencia de operaciones que tiene incluidas. Puede actuar también como contenedor para otros componentes agregados en él, los cuales son instanciados (es decir incorporados al modelo de la situación de tiempo real en que aparecen a partir de su especificación) junto con la instanciación del componente contenedor y se declaran como atributos con el estereotipo <<obj>>.

Puede también tener parámetros, que representan entes que son parte de su descripción pero que están aún por designar, tales como otros componentes referenciados, operaciones, eventos externos, requisitos temporales, etc. Los parámetros se declaran como atributos con el estereotipo <<ref>>. Al momento de instanciar un componente se deben proporcionar los valores concretos de los parámetros que éste tenga definidos. Los parámetros y los componentes agregados de un componente son visibles tanto en su interfaz como en la descripción de sus operaciones y componentes agregados.

Las clases derivadas de Ada_Mast_Component difieren particularmente en cuanto al tipo de operaciones que pueden declarar y las prerrogativas de sus operaciones y parámetros. La clase Main modela el procedimiento principal de una partición Ada y se caracteriza por tener un Scheduling_Server implícito que modela el thread principal. La clase Task representa una tarea Ada y se caracteriza por contener operaciones del tipo Entry, que se emplean para sincronizar el thread de quien invoca la operación con el thread propio del Task. La clase Protected se emplea para modelar los objetos protegidos de Ada, que se caracterizan porque toda las operaciones que declara, bien sean Local_Operations o del tipo Guarded, serán ejecutadas en régimen de exclusión mutua.

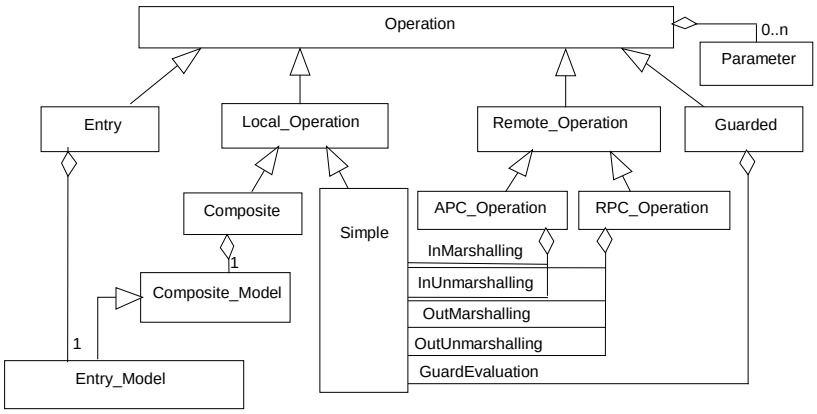

Fig. 4. Jerarquía de operaciones que modelan la interfaz de componentes

Cada operación del tipo Guarded, como son las *entrie*s de un objeto protegido, tiene agregada una operación del tipo Simple_Operation que modela el efecto temporal de la evaluación de la condición de guarda. La clase Remote_Call_Interface modela un componente cuyas operaciones pueden ser invocadas tanto desde la partición local como desde una partición remota. Cada Remote_Operation tiene los atributos necesarios para proporcionar los parámetros de la transmisión e incorpora las operaciones simples que modelan la codificación (*marshalling*) y decodificación (*unmarshalling*) de los argumentos de sus operaciones exportadas. La clase Operation y sus especializaciones se emplean para modelar el comportamiento temporal y los mecanismos de sincronización de los procedimientos y funciones declaradas en la interfaz de los componentes lógicos. La figura 4 muestra esta jerarquía de operaciones.

Modelo de las situaciones de tiempo real: una Real_Time_Situation representa un cierto modo o configuración de operación hardware/software del sistema y la carga de trabajo que soporta para la que se tienen definidos requisitos de tiempo real. Define así el modelo del sistema sobre el cual se han de emplear las herramientas de análisis que sean necesarias. El paso de una determinada situación de tiempo real a otra, es decir, el cambio de modo no está incluido en el modelo, y por tanto cada una se analiza de manera independiente.

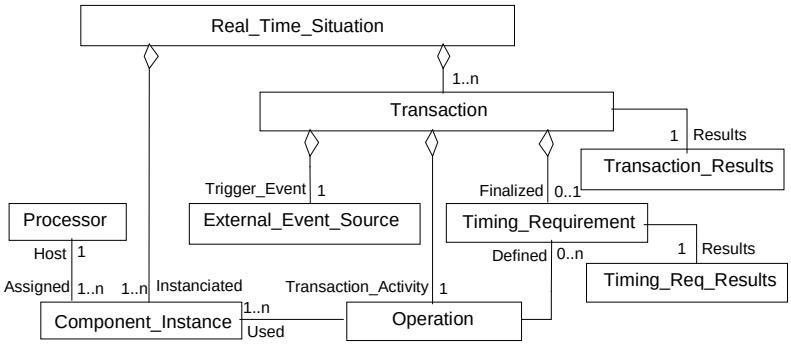

Fig. 5. Sección del metamodelo del modelo de situaciones de tiempo real de ADA_MAST

Una situación de tiempo real se describe por una parte mediante la declaración de todas las instancias de los componentes software que participan en las actividades propias de la configuración que se modela y su correspondiente despliegue sobre la plataforma; por otra, mediante un conjunto de transacciones que modelan la carga del sistema y describen las secuencias no iterativas de acciones que se ejecutan ante la llegada de los eventos externos (External_Event_Source) definidos. Estos eventos además dan la pauta para evaluar si el sistema satisface los requisitos temporales impuestos (Timing_Requirement). La actividad que se inicia ante el disparo del Trigger_Event de una transacción se especifica mediante el enlace Transaction_Activity, y la operación enlazada debe ser exportada por alguna de las instancias declaradas. Una transacción puede tener muchos requisitos temporales, pero hay al menos uno por defecto etiquetado como Finalized, que debe proporcionarse en su declaración y que corresponde a la terminación del grafo de actividad de la transacción. Cualquier otro que se considere relevante y se incluya en algún punto de su descripción, deberá ser exportado por los componentes en que aparece y finalmente ser proporcionado como argumento en la invocación de la Transaction_Activity. Se tienen diversos tipos de eventos de disparo y de requisitos temporales que dan gran versatilidad a los modelos alcanzables, y pueden ser consultados en las referencias sobre MAST [8][9].

Las instancias de clases derivadas de Real_Time_Situation, Transaction, Timing_Requirement o Scheduling_Server tienen todas una instancia asociada cuya clase es una especialización de la clase Results, que contendrá el conjunto de variables en las que recoger los resultados que las herramientas de análisis de planificabilidad proporcionan.

4 Modelo de tiempo real de los componentes Ada

Aunque la metodología de modelado y análisis que se presenta es en principio independiente del lenguaje que realmente se emplee para programar, y puede aplicarse al modelado de un amplio espectro de sistema de tiempo real, la semántica de los componentes de modelado que se han definido, la sintaxis y las convenciones seguidas están específicamente pensadas para programas diseñados y codificados con Ada. En estos casos, su aplicación es más sencilla y automatizable.

El modelo se adapta a la estructura de las aplicaciones Ada: Las instancias de Ada_Mast_Component permiten modelar el comportamiento temporal de *packages* o *tagged classes* que son los elementos estructurales básicos de una arquitectura Ada y en él se describen y se declaran respectivamente los modelos de tiempo real de los procedimientos y funciones públicos del package o clase Ada que modela, y los componentes (otros *packages*, *protected objects*, *tasks*, etc.) que están declarados en el *package* o clase Ada que éste modela, ya sea en la parte pública, en la privada o en el cuerpo, y que son relevantes para modelar su comportamiento temporal.

Un Ada_Mast_Component modela tan sólo el código incluido en la estructura lógica que describe. Así, si un *package* presenta dependencia de otros, no incluye en su modelo el de los *packages* de que depende, únicamente hace referencia a ellos.

Modela la concurrencia que introducen las tareas Ada: Los componentes con estereotipo Task modelan el thread que se introduce con cada tarea Ada. Por cada instancia de un Task se introduce de forma implícita un Scheduling_Server que se asocia al procesador en que se instancia el componente. La sincronización entre actividades de diferentes threads sólo se localiza en los modelos de las operaciones de estereotipo <<entry>>. El modelo gestiona automáticamente la sobrecarga en los procesadores debida a los cambios de contexto entre tareas de un mismo procesador.

Modela los bloqueos que introduce el acceso a objetos protegidos: Los objetos de la clase Protected se emplean para modelar los objetos protegidos Ada. El modelo incluye: la exclusión mutua con que se ejecutan las operaciones de su interfaz, la evaluación de la condición de guarda de sus Entry, el cambio de prioridad que introduce el uso del protocolo de techo de prioridad y su posible suspensión hasta que se alcance el estado en el que la condición de guarda se satisface. Aunque la metodología que se propone no es capaz de modelar todas las posibilidades de sincronización que se pueden codificar el emplear condiciones de guarda en las entry de un objeto protegido, sí que permite describir los mecanismos básicos de sincronización que son propios de las aplicaciones de tiempo real. Así, mecanismos de sincronización basados en objetos protegidos, tales como recepción de interrupciones hardware, activación periódica o asíncrona de tareas, espera a un grupo de eventos, o colas de mensajes, entre otros, pueden ser modelados cuantitativa y fielmente.

Modela la comunicación de tiempo real entre particiones Ada distribuidas: El modelo soporta de forma implícita el acceso local y remoto a los procedimientos APC y RPC de una *Remote Call Interface* tal como se describe en el Anexo E del estándar Ada. En la declaración de una RCI, el modelo incluye la información necesaria para que cuando un procedimiento sea invocado de forma remota, el *marshalling*, la transferencia de los mensajes por la red, el *unmarshalling* y su gestión por el *dispatcher* remoto, puedan ser incluidos de forma automática por la herramienta.

5 Ejemplo de aplicación de la metodología

Se muestra la utilización de ADA_MAST para el modelado y análisis de tiempo real de un sistema distribuido que controla una Máquina Herramienta Teleoperada (TMT por sus siglas en inglés). La plataforma del sistema está formada por dos procesadores comunicados mediante un bus CAN. El procesador **Station** hace de estación de teleoperación y aloja una aplicación tipo GUI a través de la cual el operador gestiona las tareas de la herramienta y en la que se muestra el estado del sistema; tiene asociado además un botón de emergencia que permite detener en seco la operación de la herramienta. **Controller** es un procesador empotrado que contiene el controlador de los servos de la máquina herramienta y la instrumentación asociada y que reporta periódicamente el estado de la herramienta a la estación de teleoperación.

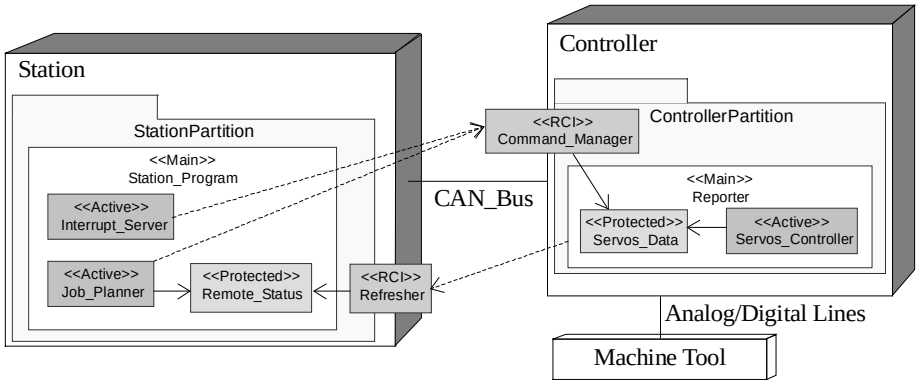

Fig. 6. Diagrama de despliegue de la Máquina Herramienta Teleoperada

5.1 Diseño lógico de la aplicación

Cada procesador tiene una partición Ada que ofrece al otro una interfaz de acceso remoto RCI. La figura 6 muestra los principales componentes y sus relaciones.

El programa principal de la partición Controller colecta el estado de la máquina herramienta cada 100ms e invoca en la otra partición el procedimiento remoto que actualiza su copia local del mismo. El objeto activo Servos_Controller que comanda la máquina herramienta tiene una tarea activada por un *timer* periódico cada 5ms. La interfaz de acceso remoto Command_Manager ofrece dos procedimientos, uno para procesar los encargos que se ordenan desde la estación y el otro para atender el eventual comando de parada de emergencia. Todos los componentes del procesador Controller comparten información mediante el objeto protegido Servos_Data.

El programa principal de la partición Station es una típica interfaz de usuario gráfica que comparte el acceso al objeto protegido Remote_Status con la interfaz de acceso remoto Refresher y dos objetos activos: la tarea Job_Planner que monitoriza y gestiona de manera periódica los encargos que se hacen a la máquina herramienta y la tarea Interrupt_Server que atiende a la interrupción hardware del botón de parada de

emergencia. La interfaz Refresher exporta un procedimiento para la actualización del estado de la máquina herramienta en la estación desde la partición remota.

Las plataformas Ada con las que se implementa el software de ambos procesadores deben contar no sólo con las librerías necesarias para la comunicación entre ambos, sino también con una implementación del *package* System.RCP que se atenga a las recomendaciones sobre el anexo E necesarias para ofrecer garantías de tiempo real.

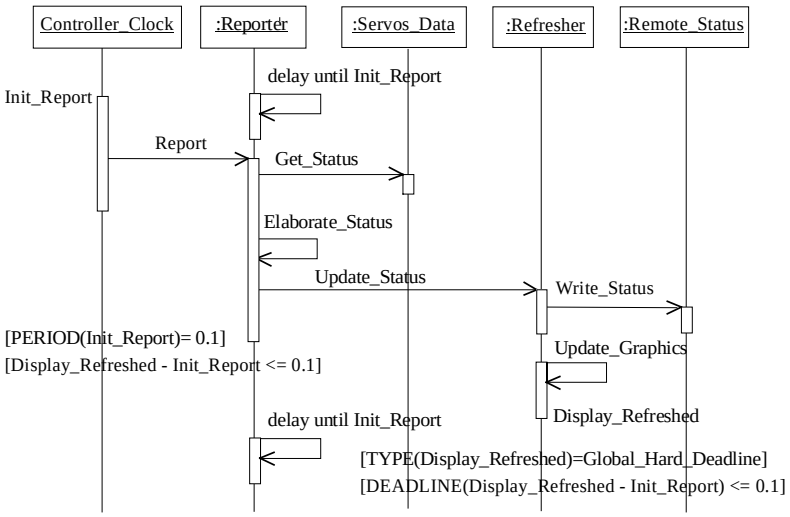

Fig. 7. Diagrama de secuencia que describe la transacción Report_Process

Este ejemplo tiene un único modo de operación y por tanto se define una sola Real_Time_Situation en la que todos los eventos, bien sean de temporizadores o externos, disparan hasta cuatro transacciones independientes con requisitos de tiempo real, y comparten tanto los recursos de procesamiento (Station, Controller, y CAN_Bus) como los objetos protegidos mencionados. Estas transacciones son:

- Control_Servos_Process: ejecuta el procedimiento Control_Servos con un periodo y plazo de finalización (*deadline*) de 5 ms.
- Report_Process: transfiere el estado de sensores y servos de Controller a Station para refrescar el *display* con un periodo y *deadline* de 100 ms.
- Drive_Job_Process: se activa por el *timer* cada segundo para revisar el estado de los encargos en curso y enviar los pendientes a la máquina herramienta.
- Do_Halt_Process: es activada de manera totalmente esporádica por el operador mediante el botón de parada de emergencia y se presume que el tiempo mínimo entre comandos sucesivos sea de 5 s. y el plazo de atención sea de 5 ms.

La figura 7 muestra la descripción funcional de la transacción Report_Process mediante un diagrama de secuencia.

5.2 Vista de tiempo real de la Máquina Herramienta Teleoperada

Se describen aquí las tres secciones de la vista UML de tiempo real de este ejemplo:

5.2.1 Modelo de la plataforma

Describe la capacidad de procesamiento y de conexión entre los tres recursos de procesamiento del sistema: los procesadores Station y Controller y la red CAN_Bus.

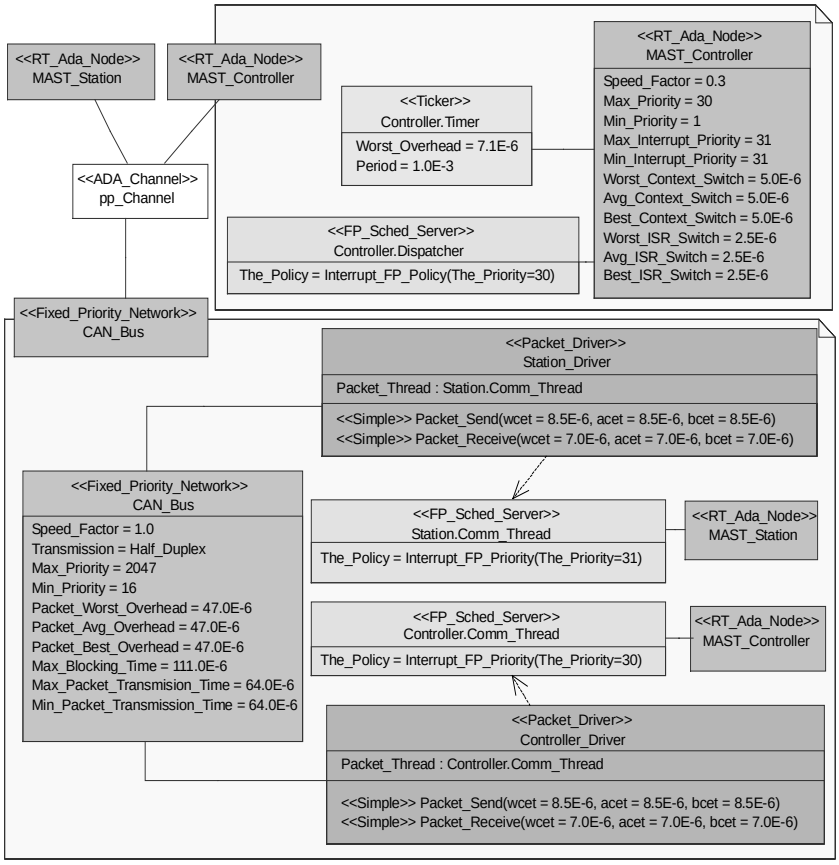

Fig. 8. Visión parcial del modelo de la plataforma de la Máquina Herramienta Teleoperada.

En la parte superior izquierda de la figura 8 se muestra el modelo de alto nivel y la conectividad que hay entre los elementos de la plataforma y a su derecha y debajo se detallan los modelos de la partición Controller y de la red CAN_Bus. La capacidad de procesamiento y los atributos del procesador están modelados por MAST_Controller de la clase RT_Ada_Node. Se trata de un procesador empotrado que ejecuta la partición Controller sobre un núcleo de tiempo real mínimo MaRTE OS [10], el cual emplea un *timer* del tipo *ticker* para temporizar las tareas que tiene asignadas. En esta parte del modelo se hacen explícitos sólo los threads de comunicaciones (*drivers*) y los propios del sistema operativo. La política de planificación y la prioridad asignadas al thread dispatcher de la partición, se especifican mediante el atributo The_policy, (del tipo Interrupt_FP_Policy en este caso) y su argumento The_Priority.

Los valores dados a los atributos propios del procesador Controller son:

Max_Priority= 30 Min_Priority= 1	Rango de prioridades admitido por el compilador Ada y MaRTE OS.
Max_Interrupt_Priority= 31 Min_Interrupt_Priority=31	Rango de prioridades admitido para manejadores de interrupciones hardware.
Worst_Context_Switch=5.0E-6 Avg_Context_Switch=5.0E-6 Best_Context_Switch=5.0E-6	Tiempo estimado de cambio de contexto entre threads de aplicación
Worst_ISR_Switch= 2.5E-6 Avg_ISR_Switch= 2.5E-6 Best_ISR_Switch= 2.5 E-6	Tiempo estimado de cambio de contexto entre un thread de aplicación y una rutina de atención a interrupciones o viceversa.
Speed_Factor= 0.3	El procesador Controller tiene un 30% de la capacidad de procesamiento del procesador de referencia del sistema.

La red CAN_Bus es un canal de tipo *half_duplex* orientado a paquetes, cada uno tiene una cabecera de 47 bits y un campo de datos de 8 bytes. La velocidad de transferencia del bus es de 1 Mbit/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 un componente del tipo Fixed_Priority_Network y los Packet_Driver que corren en los procesadores en que están desplegadas las particiones que acceden a la red. La instancia CAN_Bus de la clase Fixed_Priority_Network describe la capacidad de transferencia por la red y los atributos que tiene y que modelan su comportamiento de tiempo real son:

Max_Priority= 2047 Min_Priority= 16	Rango de prioridades admitida para los mensajes sobre un bus CAN.
Packet_Worst_Overhead=47.0E-6 Packet_Avg_Overhead=47.0E-6 Packet_Best_Overhead=47.0E-6	Sobrecarga debida a los bits del formato de trama de mensaje del bus CAN que no son de datos (47 bits por paquete).
Transmission= Half_Duplex	El modo de transmisión del bus CAN es *half-duplex*.
Max_Blocking= 111.0E-6	Tiempo de bloqueo de peor caso debido a un paquete.
Max_Packet_Transmission_Time= 6.4E-5 Min_Packet_Transmission_Time= 6.4E-5	Tiempo para transmitir el campo de datos de un paquete en este caso de una sola trama (8 bytes/packet).
Speed_Factor= 1.0	Sea esta la red de referencia del sistema (1 Mbit/s).

5.2.2 Modelo de tiempo real de los componentes lógicos

Describe el comportamiento temporal de todos los módulos del diseño lógico que podrían afectar la respuesta de tiempo real del sistema. En la figura 9 se muestra el modelo del programa principal de la partición Controller, MAST_Reporter. Se trata de una instancia de la clase Main del metamodelo, que se emplea como contenedora y tarea periódica a la vez. Declara como suyos los objetos, The_Data y The_Controller, mediante atributos de sus correspondientes tipos. La política de planificación y la prioridad del thread interno se declaran también como atributos. La descripción de la operación compuesta Report se hace en un diagrama de actividad agregado y sigue la sintaxis propuesta por UML_MAST [9]. Los argumentos de la operación simple Elaborate_Report, dan valor a los tiempos de peor, medio y mejor caso esperados para el tiempo de ejecución de la operación. En la descripción del tipo de objeto protegido MAST_Servos_Data y del tipo de tarea MAST_Servos_Controller (asignados después a The_Data y a The_Controller), los argumentos de sus operaciones han sido

omitidos tan sólo para dar simplicidad y claridad a la figura. Obsérvese como el argumento The_rc de la operación compuesta Report se emplea internamente en su descripción para invocar el procedimiento Update_Status de la interfaz Refresher (que es del tipo APC), la cual a su vez ha sido declarada con un atributo <<ref>> y es accesible por tanto dentro de MAST_Reporter bajo el nombre Remote_Refresher.

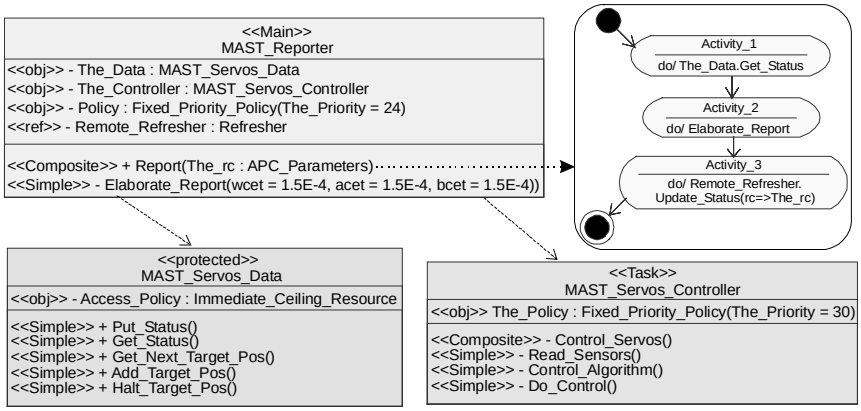

Fig. 9. Extracto de la descripción del componente MAST_Reporter

5.2.3 Modelo de las situaciones de tiempo real

La situación de tiempo real que se ha de analizar comprende la formulación de las cuatro transacciones descritas en el apartado 5.1. En la figura 10 se muestra el modelo de la transacción Report_Process, que fue descrita funcionalmente en la figura 7.

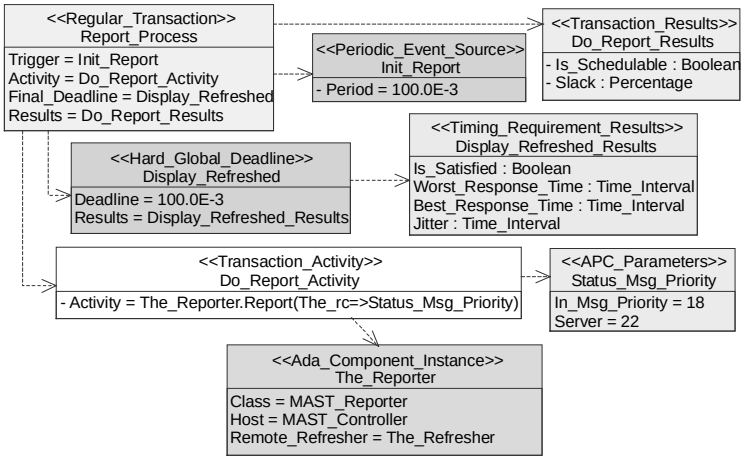

Fig. 10. Transacción Report_Process

Para analizar el sistema se requiere tener declarados en la situación de tiempo real los cuatro componentes de primer nivel que modelan la aplicación y que corresponden a los dos programas principales: The_Reporter y The_Station_Program y a las dos interfaces exportadas: The_Command_Manager y The_Refresher.

La transacción Report_Process se declara como una instancia de la clase Regular_Transaction, su atributo Trigger referencia el Periodic_Event_Source Init_Report y Final_Deadline al Hard_Global_Deadline Display_Refreshed la actividad a ser iniciada por la transacción es la operación Report del objeto The_reporter. El atributo Period de Init_Report indica que se recibe un evento cada 100 ms. El atributo Deadline de Display_Refreshed indica que el plazo para alcanzar el estado Display_Refreshed que aparece en la descripción de Do_Report_Activity es de 100 ms. contados desde la generación del evento de disparo (Init_Report). La descripción detallada de la secuencia de actividades que conlleva la transacción se obtiene sustituyendo de manera recursiva todos los parámetros y modelos de actividad de las operaciones tal como se les declara en el modelo lógico, con los valores instanciados a partir de los objetos que se declaran en la Situación de tiempo real. Las *do_actions* de las actividades invocan operaciones y los *swim lanes* representan los Scheduling_Servers (threads) en los que se ejecutan.

5.3 Análisis de tiempo real y diseño de la planificabilidad del sistema

Utilizando las herramientas del entorno MAST se ha analizado el ejemplo que se presenta a fin de obtener los tiempos de bloqueo y techos de prioridad de los recursos, la asignación óptima de prioridades y la holgura disponible tanto para el sistema como para cada transacción independientemente. Se ha empleado el método de análisis de planificabilidad basado en offsets por ser el menos pesimista [11][12]. En la tabla 1 se muestra un resumen de los resultados de planificabilidad obtenidos para cada transacción, pudiendo compararse el tiempo de respuesta de peor caso de cada una con su correspondiente plazo. Las prioridades asignadas a las tareas han sido calculadas con el algoritmo HOPA integrado en MAST.

Table 1. Resultados del análisis para las cuatro transacciones de la situación de tiempo real

Transaction/Event	Slack	Worst response	Deadline
Control_Servos_Process	19.53%		
End_Control_Servos		3.833ms	5 ms
Report_Process	254.69%		
Display_Refreshed		34.156ms	100 ms
Drive_Job_Process	28.13%		
Command_Programmed		177.528ms	1000 ms
Do_Halt_Process	25.00%		
Halted		4.553ms	5 ms

En la tabla 1 se muestran también las holguras de cada transacción. La holgura (*Slack*) es el porcentaje en el que todas las operaciones involucradas pueden ser incrementadas sin hacer el sistema no planificable o decrementadas para que lo sea si es negativo. Como se puede apreciar el porcentaje en el que se puede aumentar los tiempos de ejecución de las operaciones de la transacción Report_Process

manteniendo la planificabilidad del sistema es de 254.69%. Mientras que desde el punto de vista global el tiempo de ejecución de todas las operaciones del sistema tan sólo podría crecer un 2.34%, ya que es éste el valor de la holgura calculada para el sistema.

6 Conclusión

En este trabajo se ha presentado una metodología para modelar el comportamiento de tiempo real de una aplicación codificada en Ada, con el suficiente nivel de detalle como para que herramientas tales como el análisis de planificabilidad, la asignación optima de prioridades, el cálculo de holguras, etc. sean aplicables.

Su principal característica es que permite modelar componentes Ada complejos (*package*, *tagged class*, *protected objects*, *tasks*, etc), de forma independiente al uso que de ella se haga en una aplicación concreta, lo que sirve como base de una metodología de diseño de sistemas de tiempo real basada en componentes Ada reusables.

La metodología libera al diseñador de la aplicación de tener que modelar los mecanismos internos del sistema operativo o del ejecutivo de tiempo real que soporta la aplicación Ada. Así, el modelado de los cambios de contexto entre tareas, de las tareas de background de gestión de las comunicaciones o de los *timers*, los *mutexes* de acceso a los objetos protegidos, o los mecanismo de acceso a interfaces remotas (RCI), son soportados a partir de un modelo de la plataforma independiente de la aplicación o del código Ada de los componentes.

El poder de modelado de la metodología que se presenta es capaz de cubrir la mayor parte de los patrones o estructuras software que son de uso extendido en el desarrollo de la mayoría de aplicaciones Ada que son analizables. Desde luego el modelado de componentes que implican mecanismos complejos de sincronización basados en su estado interno (que en Ada se implementan mediante condiciones de guarda de *entries* en tareas o en objetos protegidos) y que conduce a situaciones no analizables, no está de momento resuelto con carácter general, y en tales casos no es aplicable de manera directa la metodología propuesta. Sin embargo justamente por su no analizabilidad estas estructuras no son frecuentes en sistemas de tiempo real

La metodología que se ha presentado está actualmente siendo implementada como parte de las herramientas del entorno UML-MAST y al igual que las de simulación y la de modelado basado en componentes de tiempo real, constituye una de nuestras líneas principales de trabajo futuro. La descripción detallada de estas herramientas y metodologías y las implementaciones que de ellas estén disponibles se pueden encontrar en: http://mast.unican.es

Bibliografía

[1] S. Tucker Taft, and R.A. Duff (Eds.) “Ada 95 Reference Manual. Language and Standard Libraries”. International Standard ISO/IEC 8652:1995(E), in Lecture Notes on Computer Science, Vol. 1246, Springer, 1997.

[2]L. Pautet and S. Tardieu, “Inside the Distributed Systems Annex”, Intl. Conf. on Reliable Software Technologies, Ada-Europe’98, Uppsala, Sweden, in LNCS 1411, Springer, pp. 65-77, June 1998.

[3] 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.

[4]Luís Miguel Pinho, "Distributed and Real-Time: Session summary", 10th.Intl. Real-Time Ada Workshop, IRTAW'01, Ávila,Spain, in Ada Letters, Vol. XXI, Number 1, March 2001.

[5] Ada-Core Technologies, Ada 95 GNAT Pro Development Environment. http:// www.gnat.com/

[6] J.J. Gutiérrez García, and M. González Harbour: “Prioritizing Remote Procedure Calls in Ada Distributed Systems”. 9th International Real-Time Ada Workshop, ACM Ada Letters, XIX, 2, pp. 67-72, June 1999.

[7] J.J. Gutiérrez García, and M. González Harbour: “Towards a Real-Time Distributed Systems Annex in Ada”. 10th International Real-Time Ada Workshop, ACM Ada Letters, XXI, 1, pp. 62-66, March 2001.

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

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

[10] M. González Harbour, and M. Aldea: “MaRTE OS: An Ada kernel for Real-Time Embedded Applications". Int. Conf. on Reliable Software Technologies, Ada_Europe'01. Leuven, May, 2001

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

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

[13] J.M. Drake, J.L. Medina y M. González Harbour: ”Entorno para el Diseño de Sistemas Basados en Componentes de Tiempo Real”, X Jornadas de Concurrencia, Jaca, Junio, 2002

[14] Selic B., Moore A., Woodside M., Watson B., Bjorkander M., Gerhartdt M., y Petriu D.: "Response to the OMG RFP for Schedulability, Performance and Time” OMG document ad/2001-06-14. Junio, 2001.