Implementación de los servicios temporales de Ada 2005 en un núcleo de tiempo real

José Redondo, José A. Pulido, Santiago Urueña y Juan A. de la Puente

Departamento de Ingeniería de Sistemas Telemáticos (DIT)

Universidad Politécnica de Madrid (UPM)

Ciudad Universitaria, s/n

28040 Madrid

{jredondoh,pulido,suruena,jpuente}@dit.upm.es

Resumen

La monitorización del tiempo de ejecución consumido por las diferentes tareas es vital en los sistemas críticos, ya que permite detectar y corregir desviaciones del comportamiento ideal incluso antes de que alguna tarea comience a perder sus plazos. Este artículo muestra la implementación de los mecanismos relativos a la monitorización del tiempo de ejecución incluidas en el nuevo estándar Ada 2005 sobre un núcleo de tiempo real basado en el perfil de Ravenscar.

1. Introducción

Las principales características de los sistemas de tiempo real son la predictibilidad y la puntualidad de las respuestas. Tras elegir un método de planificación adecuado para las necesidades del sistema [1], a la hora de diseñar un sistema se lleva a cabo un análisis de tiempo de respuesta mediante el cual se comprueba si todas las tareas pueden finalizar su trabajo antes de que expire su plazo de respuesta [2]. Sin embargo, no se puede confiar la seguridad del sistema solamente a las técnicas de planificación y análisis estático, ya que éste se basa en una serie de parámetros estimados a priori que pueden no cumplirse en tiempo de ejecución, siendo dos los casos más frecuentes:

- Al realizar el análisis, el tiempo de ejecución de una tarea en el caso más desfavorable se supone conocido, sin embargo, su valor es difícil de calcular por lo que es posible que en el análisis de tiempo de respuesta se haya utilizado un valor demasiado optimista, dando lugar a resultados engañosos.

- Las tareas esporádicas pueden activarse en cualquier instante, no obstante, a la hora de realizar el análisis se parte de la base de que debe transcurrir un periodo mínimo entre dos activaciones consecutivas. Sin embargo, este periodo mínimo podría no ser respetado en tiempo de ejecución. Por ejemplo, debido a un fallo *hardware* un sensor podría provocar interrupciones continuamente.

Puesto que en un sistema de tiempo real crítico el hecho de que una tarea no cumpla su plazo de respuesta puede tener consecuencias catastróficas, se tiende a trabajar con un margen de error amplio. Es decir, a la hora de efectuar el análisis se utilizan valores muy pesimistas para asegurarse de que en la realidad el caso peor esté dentro de los márgenes del análisis. Como consecuencia, la carga real del procesador suele ser muy baja. Dicho con otras palabras, el sistema es ineficiente porque se emplean muchos recursos para desempeñar poco trabajo.

Disponer de herramientas que vigilen el comportamiento de las tareas en tiempo de ejecución, principalmente su consumo de tiem-

po de procesador, puede ayudar a mejorar la eficiencia de estos sistemas [3]. Primeramente se pueden utilizar en una implementación piloto, ayudando a ajustar los parámetros que intervienen en el análisis. Posteriormente, ya en tiempo de ejecución, permiten detectar con prontitud cualquier variación del comportamiento previsto y, por tanto, se pueden tomar las medidas adecuadas para salvaguardar la integridad del sistema antes incluso de que ninguna tarea llegue a incumplir su plazo de respuesta. Una combinación adecuada de estas herramientas permite aumentar la carga del procesador en sistemas críticos manteniendo el nivel de seguridad exigido, lo cual mejora sensiblemente su eficiencia y, como consecuencia, se reducen costes.

En este artículo se detalla la implementación de cuatro mecanismos incorporados al nuevo estándar Ada 2005 que permiten monitorizar el comportamiento de las tareas en tiempo de ejecución: relojes, temporizadores individuales y colectivos, así como eventos programados. La implementación se ha hecho sobre el sistema de compilación GNATforLEON, que contiene un núcleo de tiempo real basado en el perfil de Ravenscar (una evolución de ORK [4, 5]). Por ello, en primer lugar se destacan varios aspectos relativos a la compatibilidad de dicho perfil con estas herramientas. Posteriormente, se dedica una sección a la implementación particular de cada herramienta.

2. Compatibilidad con el perfil de Ravenscar

El perfil de Ravenscar en un subconjunto del lenguaje Ada del que se excluyen todas aquellas construcciones que pueden generar indeterminismo (e.g. transferencia asíncrona de control, prioridades dinámicas...) [6, D.13.1], por lo que se convierte en una opción muy válida para construir sistemas de alta integridad [7].

Tanto los relojes de tiempo de ejecución como los eventos programados están incluidos dentro del perfil de Ravenscar (estos últimos deben declararse a nivel de biblioteca). Sin embargo, tanto los temporizadores de tiempo de ejecución como las cuotas colectivas se han excluido en primera instancia. El funcionamiento de ambos temporizadores en sencillo: cada vez que ocupa el procesador una tarea asociada a una de estas herramientas se arma un temporizador con la cantidad de tiempo de ejecución restante; si esta cantidad se agota antes de que el temporizador sea desarmado, entonces se ejecuta un manejador definido por el usuario. Este mecanismo no produce indeterminismo por sí solo, sin embargo, la cuestión es que si un temporizador ha expirado, seguramente se debe a que la tarea está intentando consumir demasiado tiempo de ejecución, por lo que el manejador debería remediar la situación. Es en este punto donde aparecen problemas de compatibilidad con Ravenscar, ya que las medidas más lógicas como abortar la tarea defectuosa o disminuir su prioridad sí que están prohibidas por el perfil. Aún así hay trabajos que muestran cómo se puede efectuar un cambio de modo de manera compatible con el perfil de Ravenscar [8], lo cual deja abierta la posibilidad de tratar estas situaciones sin salirse de la filosofía del perfil.

Con el doble objetivo de investigar el comportamiento de los mecanismos mencionados, así como su impacto sobre el determinismo del sistema, la implementación se ha realizado sobre un núcleo compatible con el perfil de Ravenscar, GNATforLEON. Para seguir en la medida de lo posible las líneas generales del perfil, en algunos casos se han establecido ciertas restricciones sobre el uso de las nuevas herramientas.

3. Relojes de tiempo de ejecución

Un reloj de tiempo de ejecución o *Execution-Time Clock* es una herramienta asociada a una tarea que permite conocer en cada instante durante cuánto tiempo ha hecho uso del procesador desde su creación. Es decir, cuando se crea una tarea ésta lleva asociado un reloj inicializado a cero. Cada vez que la tarea es despachada para su ejecución el reloj comienza a contar. Cuando se produce un cambio de contexto y la tarea es desalojada, el reloj es pausado hasta el momento en el que la tarea

vuelva a hacerse con el procesador, tal y como se muestra en la figura 1.

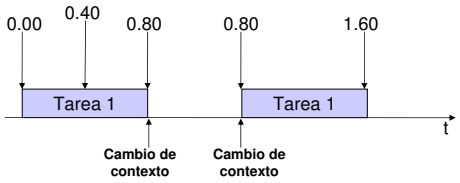

Figura 1: Funcionamiento del reloj de tiempo de ejecución

La implementación de esta herramienta es sencilla [9]. Cuando la tarea entra en el procesador se consulta el reloj de tiempo real del sistema. En el siguiente cambio de contexto, cuando la tarea es desalojada, se vuelve a consultar el reloj de tiempo real. La diferencia entre ambas lecturas es el tiempo de procesador consumido durante la última ejecución por lo que dicha cantidad se acumula en el reloj de tiempo de ejecución de la tarea.

Puesto que cada tarea cuenta con su propio reloj de tiempo de ejecución, todos los datos asociados a esta herramienta se guardan en sus respectivos bloques de control (ATCBs) mediante dos nuevos campos:

- *Time_Init_Execution*: el instante en el que la tarea ha recibido el control del procesador.
- *Execution_Time*: la cantidad de tiempo consumido por la tarea desde su creación hasta el cambio de contexto más reciente.

El paquete *Ada.Execution_Time* [6, D.14] incluye los subprogramas relativos a los relojes de tiempo de ejecución. Especialmente relevante resulta la función *Clock*, ya que permite consultar el tiempo de procesador consumido por una tarea determinada hasta ese instante. Su funcionamiento es el siguiente:

- Si la tarea está activa en dicho momento, el valor devuelto será la suma del valor guardado en el campo *Execution_Time* y el tiempo transcurrido desde *Time_Init_Execution* hasta el momento actual.
- Si es cualquier otra tarea se devolverá la cantidad almacenada en el campo *Execution_Time* de su ATCB.

4. Temporizadores de tiempo de ejecución

La funcionalidad de los temporizadores de tiempo de ejecución o *Execution-Time Timers* es la siguiente: un temporizador está ligado a una tarea, cuando dicho temporizador se arma con una cuota de tiempo de ejecución, dicha cuota comienza a reducirse a medida que la tarea va consumiendo tiempo de procesador. Si la cuota llega a agotarse antes de que el temporizador sea desarmado, se dispara una interrupción y se ejecuta un manejador definido por el usuario.

Esta herramienta ya fue incluida en el estándar POSIX antes de su inclusión en el estándar Ada. En el anexo D del manual de referencia, se especifica toda la información relativa a esta funcionalidad, así como el paquete que la implementa, *Ada.Execution_Time.Timers* [6, D.14.1]. En este paquete se especifican todas las operaciones relativas a los temporizadores de tiempo de ejecución. Éstos podrán ser armados o desarmados por el usuario; además, los valores de la cuota asociada podrán ser consultados y modificados.

Para maximizar la compatibilidad con el perfil de Ravenscar, la declaración debe hacerse a nivel de biblioteca; además, sólo se permite un único temporizador de tiempo de ejecución por tarea. De esta forma se permite asignar estáticamente la memoria necesaria para dar soporte a esta funcionalidad, lo cual es un requisito importante a la hora de eliminar el indeterminismo.

El hecho de que cada tarea sólo disponga de un temporizador permite almacenar los principales valores relativos a los temporizadores en el ATCB de la tarea y así facilitar el acceso a los mismos. Con este objetivo, se han creado varios campos nuevos:

- *Time_Remaining*: tiempo de ejecución que le queda al temporizador antes de su expiración.

- *Is_Timer_Alarm*: variable utilizada en el mecanismo de colas de alarmas.
- *Time_Diff*: diferencia de tiempo entre el principio de la ejecución y el instante de asignación del manejador.
- *TM_Integer*: identificador del temporizador asociado a la tarea.
- *Handler*: manejador indirecto que permite el acceso al manejador final definido por el usuario.

Además, para completar el soporte, la parte privada de la implementación del tipo *Timer*, contiene tres campos:

- *Thread*: la tarea asociada al temporizador.
- *TM_Id*: el entero usado como identificador.
- *Is_Set*: variable booleana que indica si el temporizador está armado o no.

Se pueden usar dos procedimientos para armar un temporizador, según se quiera cargar un intervalo de tiempo o un instante absoluto. Estos procedimientos actualizarán los campos del ATCB de la tarea y los del objeto *Timer*, así como el manejador.

Para implementar tanto éste como otros servicios es necesario tener en cuenta los recursos *hardware* disponibles. El procesador LEON2, hacia el que está orientado GNATforLEON, sólo dispone de dos temporizadores *hardware* que serán los encargados de implementar todos los servicios temporales. En la sección 7 se explica con detalle cómo se soluciona este inconveniente.

Otro problema encontrado a la hora de implementar los temporizadores de tiempo de ejecución tiene que ver con el manejador asociado, concretamente con la dificultad de hacer una llamada desde el nivel del núcleo a un procedimiento definido por el usuario a nivel de aplicación. Como se puede observar en la especificación del paquete, el manejador ha de ser un procedimiento protegido y recibir un objeto del tipo *Timer* como parámetro. Esta situación obliga al manejo de tipos definidos un nivel por encima del núcleo, por lo que se generan dependencias circulares; para evitarlas se ha diseñado una estructura indirecta. De esta manera, el manejador propiamente dicho no se encuentra en el campo *Handler* del ATCB de la tarea, sino en un *array* externo en el que se almacenan todos los manejadores. Por tanto, el campo *Handler* contendrá un puntero a un procedimiento predefinido, que usará el identificador del temporizador para acceder a la matriz y ejecutar el manejador correspondiente.

Este problema entre estructuras de bajo y alto nivel también se produce cuando es necesario acceder al ATCB de la tarea, ya que el tipo utilizado en la especificación del paquete, *Ada.Task_Identification.Task_Id* no es el que se utiliza en el núcleo; para resolverlo, se ha implementado un paquete auxiliar que realiza la conversión entre ambos tipos.

5. Cuotas colectivas de tiempo de ejecución

En algunos sistemas sería deseable poder ajustar la granularidad con la que se realiza la monitorización del tiempo de ejecución. Las cuotas colectivas de tiempo de ejecución o *Group Execution-Time Budgets* permiten al usuario monitorizar el tiempo de procesador consumido por un grupo de tareas, sin prestar atención a cómo se reparte este tiempo entre ellas.

El funcionamiento de las cuotas colectivas de tiempo de ejecución es análogo al de los temporizadores de tiempo de ejecución, sólo que, en lugar de estar asociada a una sola tarea, una cuota colectiva controla el tiempo consumido por varias de ellas. De manera semejante a lo que sucede con los temporizadores de tiempo de ejecución, las cuotas colectivas tienen una cierta cantidad de tiempo de ejecución y un manejador, de manera que si la cuota de tiempo de ejecución llega a agotarse se ejecuta el manejador. Igualmente, la cuota de tiempo de ejecución puede consultarse y modificarse, así como las tareas que forman parte del grupo y el manejador asociado. Sin em-

bargo, siguiendo con las restricciones particulares que imponemos en esta implementación basada en el perfil de Ravenscar, las cuotas colectivas deben ser estáticas, es decir, creadas al comienzo y con miembros permanentes, de manera que las diferentes tareas no pueden pasar a formar parte de un grupo o dejar de hacerlo durante el tiempo de ejecución.

Esta funcionalidad permite asegurar el buen comportamiento global de una aplicación multitarea, sin necesidad de descender hasta el nivel de tarea. Su uso puede ser de gran utilidad para implementar servidores aperiódicos, servidores esporádicos o de plazos, cuya implementación antes de contar con esta herramienta era muy compleja. La biblioteca que implementa esta funcionalidad es *Ada.Execution_Time.Groups_Budgets* [6, D.14.2].

Puesto que el funcionamiento de las cuotas colectivas es similar al de los temporizadores de tiempo de ejecución, su implementación no es muy distinta. La mayor diferencia radica en que, al estar los temporizadores individuales ligados a una sola tarea, todos los nuevos campos introducidos para dar soporte a esta función se añadían al ATCB de las tareas. Sin embargo, las cuotas colectivas son comunes a varias tareas y, por tanto, los nuevos campos necesarios también deberían serlo. Una opción sería añadir estos campos también al ATCB de las tareas, sin embargo, la sobrecarga que supondría su actualización en todos los miembros del grupo cada vez que se produjere algún cambio ha llevado a la decisión de conservar el tiempo de la cuota remanente del grupo, así como los miembros que lo forman, en variables externas al ATCB, de manera que la información más utilizada y que se ve modificada más a menudo esté en un solo lugar, accesible para todas las tareas. Los campos que se añaden al ATCB en esta ocasión son:

- *Is_GB_Alarm*: valor booleano utilizado en el mecanismo de colas de alarmas.
- *Time_Diff_GB*: homólogo al de los temporizadores.
- *GB_Id*: identificador del grupo de tareas.
- *GB_Index*: posición de la tarea en el grupo de tareas.
- *Handler_GB*: análogo a los temporizadores, puntero que permite ejecutar el manejador asociado.

Excepto el valor de la cuota restante, que será almacenado en una variable interna del núcleo, los datos restantes de los grupos son incluidos en el tipo que representará a las cuotas colectivas, *Group_Budget*. La implementación de su parte privada tiene los siguientes campos:

- *GB_Id*: entero que identifica al grupo de tareas.
- *Num_Members*: número de tareas que forman el grupo.
- *Is_Set_Handler*: indica si el manejador asociado al grupo está armado o no.
- *Array_Threads*: matriz con los ATCB de las tareas del grupo.
- *Array_Task_Id*: matriz con los datos de alto nivel de los miembros.

La información contenida en las dos matrices podría parecer redundante. Sin embargo, es la manera más eficiente de tratar con el problema ya mencionado de trabajar con tipos definidos a alto nivel desde el propio núcleo sin crear dependencias circulares.

6. Eventos programados

Los eventos programados o *Timing_Events* permiten ejecutar un procedimiento definido por el usuario en un instante dado, sin necesidad de recurrir a una tarea o una instrucción *delay* para ello. Se trata de un mecanismo de bajo nivel que permite asociar un procedimiento protegido a un instante temporal (mediante una interrupción), por lo que pueden sustituir a la tarea de alta prioridad necesaria hasta ahora, con el consiguiente ahorro de recursos.

Esta herramienta ha sido incluida en el perfil de Ravenscar y, además de otras muchas

aplicaciones, gracias a ella se pueden detectar violaciones del plazo de respuesta [10] justo en el momento en que se producen y no a posteriori, por lo que se puede actuar en consecuencia a tiempo de minimizar las consecuencias negativas sobre el sistema. Por esta razón, se ha realizado su implementación para el núcleo GNATforLEON. El paquete Ada.Real_Time.Timing_Events implementa este servicio. Su especificación puede ser consultada en el anexo D del Manual de Referencia de Ada [6, D.15]. Los subprogramas de manejo y consulta de los eventos temporizados, así como los de sus manejadores son muy similares a los implementados para el resto de servicios temporales.

A diferencia de los mecanismos vistos hasta el momento, los eventos programados no están asociados a ninguna tarea, por lo que los datos relativos a ellos se almacenan a nivel de biblioteca en el tipo que los representa: *Timing_Event*. La implementación de este tipo consta de los siguientes campos:

- *TE_Id*: entero que identifica al evento programado.
- *Time_Of*: instante en el que está fijado el evento.
- *Is_Set*: indica si el evento está o no armado.
- *Queue_Id*: variable del tipo *TE_Alarm_Queue_Id*, utilizada en la implementación de la nueva cola de alarmas (ver sección 7).

El tipo *TE_Alarm_Queue_Id* es interno al núcleo y es un puntero a un registro con los siguientes campos:

- *TE_Id*: entero que identifica al evento programado asignado.
- *Alarm_Time*: tiempo del evento asociado al objeto.
- *Previous_Alarm*: indica su antecesor en la cola de alarmas.
- *Next_Alarm*: apunta al siguiente elemento de la cola.

7. Funcionamiento e interacción de los mecanismos de colas

GNATforLEON está diseñado para trabajar directamente sobre el procesador LEON2. Como ya se ha detallado, esta plataforma solamente dispone de dos temporizadores *hardware*. Uno de ellos es fijado como temporizador periódico, ya que se utiliza para dar soporte el reloj de tiempo real del sistema. Por lo tanto sólo queda un temporizador *hardware* disponible para soportar el resto de servicios temporales del sistema: instrucciones *delay*, temporizadores de tiempo de ejecución, cuotas colectivas y eventos programados.

Dado que cada servicio tiene unas características distintas no se puede utilizar una sola cola de alarmas en la que integrar elementos procedentes de servicios distintos, por lo tanto, para solventar el problema de la falta de dispositivos *hardware* se ha diseñado el esquema representado en la figura 2, que cuenta con diferentes colas de alarmas. Así pues, cada uno de los servicios listados utiliza su cola particular.

En primer lugar, las instrucciones *delay* tienen como característica principal el que no pueden ser eliminadas, por lo que se organizan mediante una cola simple, empezando por la más próxima en el tiempo y terminando por la más lejana. Las alarmas debidas a eventos programados también se ordenan mediante una lista ordenada, sin embargo, estos eventos sí que pueden ser cancelados, por lo que para simplificar su mantenimiento se ha optado por una lista doblemente enlazada que utiliza el tipo *TE_Alarm_Queue_Id* como elemento principal. En cuanto a los temporizadores de tiempo de ejecución, teniendo en cuenta que sólo se permite uno por tarea y que sólo puede haber una tarea activa, no es necesario implementar una cola; con un mecanismo de silla (cola con un solo elemento) en el que conste el instante de la alarma es suficiente para soportar el servicio. Análogamente, para las cuotas colectivas sólo es necesario conocer el valor de una de ellas, la ligada a la tarea activa, por tanto una silla vuelve a ser suficiente para soportar el servicio.

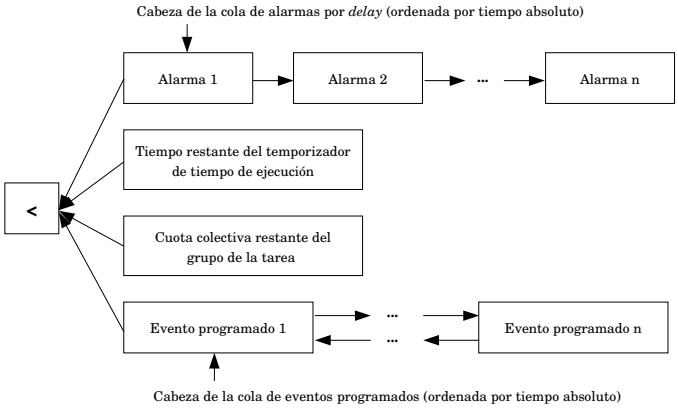

Figura 2: Esquema de funcionamiento de las colas del sistema.

Una vez que las alarmas debidas a los diferentes servicios están clasificadas el problema queda solucionado, ya que el primer valor de cada una de estas listas será consultado y el valor más próximo en el tiempo será el que se cargue en el temporizador *hardware*. Además, se modifican unas variables para distinguir a qué cola pertenece la alarma cargada. Por supuesto, el procedimiento de comprobar cuál es la alarma más cercana y cargarla en el temporizador *hardware* debe ser repetido cada vez que se produce algún cambio en los elementos del esquema descrito, lo cual incluye: cambios de contexto, modificación de un temporizador individual o colectivo así como la inclusión, cancelación o modificación de un evento programado, etcétera. Cuando se produce la expiración del temporizador no periódico, su rutina de interrupciones consulta las variables que permiten determinar a qué cola pertenece la alarma expirada y realiza la acción pertinente: ejecución del manejador correspondiente o activación de una tarea suspendida. Una vez llevada a cabo la gestión apropiada, se volverá a comparar los valores iniciales de las colas, iterando de nuevo el ciclo.

La tabla 1 muestra a modo de ejemplo varios valores estadísticos que indican la sobrecarga que introducen estos servicios. Lo más significativo es el aumento de 200 instrucciones en el cambio de contexto. Solamente la lectura del reloj de tiempo real ya necesita 71 instrucciones. El resto de las instrucciones son necesarias para llevar a cabo la gestión de las diferentes colas de alarmas. Este número es más elevado a causa de la escasez de recursos *hardware* disponibles en LEON2, ya que en caso de contar con un número elevado de temporizadores *hardware* la gestión sería mucho más sencilla.

Cuadro 1: Sobrecarga introducida

Operaciones	**Instrucciones**
Cambio de contexto	+200
Armar Temporizador	250
Latencia Manejador	400

8. Conclusiones y trabajo futuro

Los nuevos servicios temporales introducidos en el estándar Ada 2005 pueden ayudar extraordinariamente a aumentar la seguridad de los sistemas basados en tareas, fomentando su uso en sistemas críticos, campo en el que aún existe cierta reticencia a abandonar modelos tradicionales basados en esquemas de planifi-

cación estática similares al ejecutivo cíclico.

La implementación satisfactoria de estos servicios sobre un núcleo basado en el perfil de Ravenscar ha permitido en primer lugar demostrar que es posible soportarlos en un núcleo mínimo diseñado para sistemas de alta integridad que puede someterse a un proceso de certificación. Su impacto en términos de sobrecarga no es despreciable, como ha podido comprobarse en la tabla anterior, pero sí asumible, especialmente si se tiene en cuenta las ventajas potenciales que se le supone al uso de estos servicios temporales. En realidad, este trabajo no es sino el primer paso en la investigación que debe hacerse sobre estos mecanismos, con objeto de cuantificar su impacto en un sistema real y comprobar si se verifican las hipótesis formuladas relacionadas con el aumento de eficiencia y seguridad.

Referencias

[1] Lui Sha, Tarek Abdelzaher, Karl-Erik Årzén, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Systems*, 28:101–155, November 2004.

[2] Mark H. Klein, Thomas Ralya, Bill Pollack, Ray Obenza, and Michael González Harbour. *A Practitioner's Handbook for Real-Time Analysis. Guide to Rate Monotonic Analysis for Real-Time Systems.* Kluwer Academic Publishers, Boston, 1993.

[3] Santiago Urueña, José Antonio Pulido, Juan Zamorano, and Juan Antonio de la Puente. Adding new features to the Open Ravenscar Kernel. In *1st International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2005)*, Palma de Mallorca, Spain, July 2005.

[4] Juan A. de la Puente, José F. Ruiz, and Juan Zamorano. An open Ravenscar real-time kernel for GNAT. In Hubert B. Keller and Erhard Plödereder, editors, *Reliable Software Technologies — Ada-Europe 2000*, number 1845 in LNCS, pages 5–15. Springer-Verlag, 2000.

[5] Juan Zamorano and José F. Ruiz. GNAT/ORK: An open cross-development environment for embedded Ravenscar-Ada software. In Eduardo F. Camacho, Luis Basañez, and Juan Antonio de la Puente, editors, *Proceedings of the 15th IFAC World Congress*. Elsevier Press, 2003.

[6] S. T. Taft, R. A. Duff, R. L. Brukardt, E. Plödereder, and P. Leroy, editors. *Ada 2005 Reference Manual. Language and Standard Libraries. International Standard ISO/IEC 8652/1995(E) with Technical Corrigendum 1 and Amendment 1.* Number 4348 in Lecture Notes in Computer Science. Springer-Verlag, 2006.

[7] Alan Burns, Brian Dobbing, and Tullio Vardanega. Guide for the use of the Ada Ravenscar profile in high integrity systems. Technical Report YCS-2003-348, University of York, 2003.

[8] Alejandro Alonso and Juan Antonio de la Puente. Implementation of mode changes with the Ravenscar profile. *Ada Letters*, XXI(1), March 2001. Proceedings of the 11th International Real-Time Ada Workshop.

[9] Juan Zamorano, Alejandro Alonso, José Antonio Pulido, and Juan Antonio de la Puente. Implementing execution-time clocks for the Ada Ravenscar profile. In Albert Llamosí and Alfred Strohmeier, editors, *Reliable Software Technologies — Ada-Europe 2004*, volume 3063 of *LNCS*. Springer-Verlag, 2004.

[10] José A. Pulido, Santiago Urueña, Juan Zamorano, and Juan A. de la Puente. Handling temporal faults in Ada 2005. In Nabil Abdennadher and Fabrice Kordon, editors, *Reliable Software Technologies — Ada-Europe 2007*, number 4498 in LNCS, pages 15–28. Springer-Verlag, 2007.