Revista Iberoamericana de Automática e Informática Industrial
CEA IFAC comité español de automática
RIAI
ISSN: 1697-7912. Vol. 3, Núm. 2, Abril 2006, pp. 19-27
http://riai.isa.upv.es

ORK: TECNOLOGÍA DE SOFTWARE FIABLE PARA SISTEMAS DE TIEMPO REAL [1]

Juan Antonio de la Puente * **y Juan Zamorano** **

* *Departamento de Ingeniería de Sistemas Telemáticos*
** *Departamento de Arquitectura y Tecnología de Computadores*
Universidad Politécnica de Madrid, E 28040 Madrid, España.
jpuente@dit.upm.es, jzamora@datsi.fi.upm.es

Resumen. Los sistemas de control con requisitos de fiabilidad y seguridad estrictos deben tener un comportamiento temporal previsible que permita asegurar que los requisitos de tiempo real están garantizados en todos los casos. En el artículo se describe una plataforma de ejecución para este tipo de sistemas basada en el núcleo ORK *(Open Ravenscar real-time Kernel)*, orientado a la realización de sistemas de tiempo real de alta integridad utilizando un subconjunto seguro del lenguaje de programación Ada.

Palabras clave: Sistemas de tiempo real, sistemas de control por computador, seguridad, sistemas críticos, sistemas de alta integridad.

1. INTRODUCCIÓN

Muchos sistemas de control por computador realizan funciones críticas con respecto a la seguridad de las personas o los bienes, o controlan procesos de alto valor económico, en los que determinados fallos en el sistema de control pueden tener un coste inaceptable. Algunos ejemplos bien conocidos de este tipo de sistemas son los sistemas de control de vuelo de aviones, los sistemas de control de frenado en automóviles, y los sistemas de control de redes de distribucion de energía eléctrica, entre otros. En todos estos casos es necesario asegurar que el sistema de control funciona correctamente en todas las situaciones posibles, y que su comportamiento en presencia de fallos es previsible y permite mantener la integridad del sistema controlado. Se suele utilizar el término *sistemas de alta integridad* para denominar este tipo de sistemas.

Con objeto de garantizar el grado de fiabilidad requerido en los sistemas de alta integridad, generalmente se exige que estos sistemas se sometan a un proceso de *validación* y *verificación* (Sommerville 2004) que asegure que el sistema se comporta correctamente en todas las circunstancias, y que se construye de forma que se asegure su fiabilidad. En la mayoría de los casos, además, se exige que se lleve a cabo una *certificación* de que el sistema alcanza el grado de integridad necesario (Storey 1996). El proceso de certificación, que se lleva a cabo por un organismo independiente, tiene por objeto adquirir una confianza suficiente en que el sistema cumple los requisitos de funcionamiento especificados. Aunque el proceso de certificación afecta a todo el sistema, este artículo se centra, en particular, en el desarrollo de software de control de alta integridad, que se rige por normas específicas para cada campo de aplicación, como la DO-178B para sistemas de aviónica (RTC 1992), la IEC 880 (IEC 1986) para sistemas de centrales nucleares, la ECSS E40 (ECS 2002) para sistemas espaciales, y la EN 50128 (CEN 1997) para sistemas de control de ferrocarriles.

[1] Trabajo financiado parcialmente por el Plan Nacional de I+D+I (proyecto TIC2002-04123-C03-01) y por la Agencia Europea del Espacio.

La construcción de sistemas de control de alta integridad requiere, entre otras cosas, la utilización de plataformas de hardware y software adecuadas, que permitan garantizar la integridad del sistema y llevar a cabo el proceso de certificación. En lo que se refiere a la ejecución de actividades o tareas concurrentes, es fundamental utilizar métodos de planificación del procesador y otros recursos que permitan garantizar un comportamiento temporal previsible y acorde con los requisitos temporales de las tareas de control. Entre éstos los más comunes son la activación periódica o esporádica por sucesos, la limitación del *jitter* en los instantes de medida y actuación, y la necesidad de garantizar plazos de ejecución estrictos (Burns and Wellings 2001).

En el resto de este artículo se describe una tecnología para la construcción de sistemas de tiempo real fiables basada en el perfil de Ravenscar de Ada y en ORK *(Open Ravenscar real-time kernel)*, un núcleo de tiempo real especialmente concebido para este tipo de sistemas. En el apartado 2 se analizan los requisitos de planificación y análisis temporal de los sistemas de alta integridad. El modelo de concurrencia de Ada y el perfil de Ravenscar se describen en el apartado 3. En el siguiente apartado se describe el núcleo ORK, mientras que en el apartado 5 se incluyen algunos comentarios sobre el proceso de desarrollo de software basado en esta tecnología. Finalmente, el apartado 6 contiene algunas conclusiones sobre el trabajo realizado.

2. PLANIFICACIÓN DE TAREAS Y ANÁLISIS TEMPORAL

La necesidad de garantizar los requisitos de tiempo real en el proceso de certificación ha conducido con frecuencia a la utilización de métodos de planificación estáticos, en los que las tareas del sistema se ejecutan con arreglo a un plan de ejecución fijo elaborado durante el diseño del sistema y realizado mediante un *ejecutivo cíclico* (Baker and Shaw 1989, Zamorano *et al.* 1997). Sin embargo, este método tiene numerosos inconvenientes: es de muy bajo nivel, y la necesidad de reelaborar el plan de ejecución cada vez que se modifica el código de las tareas dificulta y encarece enormemente el mantenimiento de los sistemas (Locke 1992).

La aparición de métodos de análisis temporal precisos, como RMA [2] (Klein *et al.* 1993) permitió el uso de métodos de planificación de tareas más flexibles, entre los que merecen especial interés los basados en prioridades fijas con desalojo (FPPS [3]).

Desde su aparición, las técnicas de análisis temporal se han ampliado considerablemente, especialmente las basadas en el análisis del tiempo de respuesta de las tareas (RTA [4]) (Audsley *et al.* 1992). Actualmente se dispone de técnicas de análisis de tiempo de respuesta que se pueden utilizar en la mayor parte de las situaciones que aparecen en la práctica de los sistemas de tiempo real (Audsley *et al.* 1996, Sha *et al.* 2004). Estos métodos se basan en un modelo de tareas con las siguientes propiedades:

- Un sistema de tiempo real está formado por un conjunto estático de tareas. Por tanto, no se crean nuevas tareas ni termina la ejecución de ninguna de ellas durante la ejecución del sistema.
- Las tareas se activan periódicamente, con período T, o esporádicamente, cada vez que ocurre un determinado suceso, que puede ser detectado por software (en otra tarea), o por hardware (mediante una interrupción). Las activaciones sucesivas de una tarea esporádica están separadas como mínimo por un intervalo de tiempo determinado T, que se supone conocido.
- El tiempo de cómputo máximo (WCET [5]) que se necesita para ejecutar cada tarea cuando se activa se supone conocido.
- Cada tarea tiene un plazo D para completar su ejecución cada vez que se activa.
- Cada tarea tiene una prioridad fija. En cada momento se ejecuta la tarea con la prioridad más alta entre todas las que estén listas para ejecutarse, de forma que si activa una con una prioridad mayor que la que se está ejecutando en un momento dado, desaloja a ésta y pasa a ejecutarse inmediatamente.
  Si hay varias tareas listas para ejecutarse con la misma prioridad, se ejecutan en el mismo orden en que se han activado.
- Las tareas se comunican únicamente por medio de variables comunes, a las que acceden de forma mutuamente exclusiva.

La exclusión mutua en el acceso a las variables comunes tiene como efecto indeseable la denominada *inversión de prioridades* (Cornhill and Sha 1987): una tarea de prioridad baja que está ejecutando una sección crítica en la que accede a alguna variable compartida con otras tareas puede bloquear la ejecución de otras tareas de prioridad superior, aunque no utilicen esa variable. Aunque este efecto no se puede evitar, se pueden limitar sus efectos utilizando un protocolo de *herencia de prioridades* al acceder a las variables compartidas (Sha *et al.* 1990). El más sencillo de estos protocolos es el protocolo de *herencia inmediata del techo de prioridad* (ICPP [6]), que consiste en elevar la prioridad de las tareas que ejecutan secciones críticas hasta el techo de prioridad de las variables correspondientes. El techo de prioridad de una variable compartida es la prioridad más alta de las tareas que acceden a ella.

[2] *Rate-monotonic analysis*.
[3] *Fixed-priority pre-emptive scheduling*.
[4] *Response-time analysis*.
[5] *Worst-case execution time*.
[6] *Immediate ceiling priority protocol.*

En estas condiciones, y conocidos los atributos temporales de las tareas, es posible asignarles prioridades de forma óptima (Audsley *et al.* 1993). En el caso más sencillo la asignación óptima de prioridades es *monótonona en plazos*, que consiste en asignar las prioridades más altas a las tareas con plazos de respuesta más cortos (Audsley *et al.* 1992).

El modelo de concurrencia anterior, basado en planificación con prioridades fijas, junto con los métodos de análisis de tiempo de respuesta que lleva asociados, han tenido una gran aceptación para el desarrollo de sistemas de tiempo real críticos, dada su relativa sencillez y su robustez (Vardanega and van Katwijk 1999). Actualmente hay numerosos sistemas operativos de tiempo real y herramientas de desarrollo que lo soportan, lo que permite su utilización industrial en una amplia gama de aplicaciones.

3. EL MODELO DE TAREAS DE ADA

3.1 Concurrencia en Ada

El lenguaje Ada (ISO 1995) se adapta muy bien al modelo anterior. La representación de la concurrencia en el lenguaje de programación, sin necesidad de invocar explícitamente servicios de sistema operativo, permite elevar el nivel de abstracción y construir programas más fiables y robustos.

La concurrencia se expresa en Ada mediante la declaración de *tareas (tasks)*, que son unidades de programa que se ejecutan concurrentemente en un espacio de memoria común, de forma similar a las hebras *(threads)* presentes en numerosos sistemas operativos. La comunicación y la sincronización entre tareas se pueden realizar mediante un mecanismo de cita extendida o mediante *objetos protegidos*. Estos últimos son módulos que encapsulan datos y operaciones, de forma similar a los monitores, pero con un mecanismo de sincronización condicional más abstracto, las *barreras*, que permiten expresar directamente las condiciones en que se pueden ejecutar las operaciones sincronizadas *(entradas)*. Tareas y objetos protegidos son elementos de pleno derecho del lenguaje, por lo que se pueden definir con ellos tipos de datos, variables, punteros, y en general todo tipo de construcciones.

El tiempo se expresa en Ada de varias maneras. En primer lugar, hay un tipo de datos predefinido para expresar duraciones, es decir intervalos de tiempo. También hay un paquete de la biblioteca estándar *(Ada.Calendar)* que define un *tiempo de calendario*, que incluye la hora del día y la fecha, y una función para leer el reloj del sistema. Por último, otro paquete estándar *(Ada.Real_Time)* define un reloj preciso y sin saltos, adecuado para sistemas de tiempo real. Estos tipos de tiempo se pueden utilizar para suspender la ejecución de una tarea durante un intervalo de tiempo (retardo relativo) o hasta que el reloj marque un tiempo determinado (retardo absoluto), y también para programar límites de tiempo *(time-outs)* para distintos tipos de sucesos.

La planificación de tareas está basada en prioridades fijas, y la ejecución de los objetos se efectúa de acuerdo con el protocolo de herencia inmediata del techo de prioridad.

Los manejadores de interrupciones se realizan en Ada como procedimientos sin parámetros declarados en objetos protegidos.

Actualmente se está procediendo a una revisión del lenguaje (Ada 2005), que mejora entre otras cosas algunos aspectos relacionados con la concurrencia y los sistemas de tiempo real:

- Perfil de Ravenscar para sistemas críticos.
- Métodos de planificación de tareas adicionales.
- Relojes y temporizadores de tiempo de ejecución.

El perfil de Ravenscar es una restricción del modelo de tareas para su uso en sistemas críticos, que se describe con más detalle en el apartado 3.3. La inclusión de métodos de planificación adicionales, en particular EDF [7], que es el más conocido de los métodos de planificación con prioridades dinámicas (Liu 2000), añade flexibilidad al lenguaje, sobre todo de cara a las aplicaciones de tiempo real flexible. Por otra parte, los relojes y temporizadores de tiempo de ejecución permiten vigilar que las tareas no sobrepasen el tiempo de cómputo (WCET) especificado, lo que es de gran interés es sistemas de alta integridad. En conjunto, la nueva versión del lenguaje contiene elementos muy interesantes para la realización de sistemas de tiempo real.

3.2 Ada para sistemas de alta integridad

Ada es un lenguaje diseñado especialmente para construir sistemas empotrados fiables. Por este motivo se han desarrollado directrices específicas para sistemas de alta integridad (ISO 2000), que permiten seleccionar «subconjuntos seguros» del lenguaje, adecuados para realizar los distintos tipos de análisis estático que se suelen exigir en el desarrollo de sistemas críticos.

Un ejemplo especialmente interesante de subconjunto seguro de Ada es el utilizado en el lenguaje SPARK (Barnes 2003), que por otra parte extiende Ada mediante una serie de anotaciones formales que facilitan el uso de algunas técnicas de análisis estático y verificación formal mediante un conjunto de herramientas muy útiles.

En un principio los subconjuntos seguros de Ada excluían el uso de tareas, lo que es coherente con la práctica tradicional de usar ejecutivos cíclicos con planificación estática para construir sistemas críticos. Sin embargo, los avances en los métodos de análisis

[7] *Earliest deadline first*.

temporal descritos en el apartado anterior han permitido la definición de un subconjunto seguro de la parte concurrente de Ada, el *perfil de Ravenscar*, que se describe a continuación.

3.3 El perfil de Ravenscar

El perfil de Ravenscar (Burns *et al.* 1998) es un subconjunto de la parte concurrente de Ada diseñado especialmente para construir sistemas de tiempo real de alta integridad. La necesidad de este perfil surgió a la vista de la complejidad del modelo de concurrencia de Ada, que aunque muy completo y flexible, está orientado a la programación de sistemas de tiempo real en general, sin tener en cuenta las necesidades de los sistemas críticos. En particular, el modelo de concurrencia de Ada incluye numerosos elementos incompatibles con los métodos de análisis de tiempo de respuesta citados en el apartado 2, como la creación y destrucción dinámica de tareas y objetos protegidos, la comunicación mediante citas *(rendez-vous)*, la selección indeterminista, el aborto de tareas y la modificación dinámica de las prioridades de las tareas. Éste es el principal obstáculo para la utilización de tareas en sistemas de alta integridad, ya que en los procesos de certificación, como hemos visto, se suele exigir que el comportamiento temporal sea previsible y que se puedan garantizar los requisitos temporales de las tareas mediante técnicas de análisis estático. El perfil de Ravenscar prohibe el uso de todos los elementos del modelo de tareas de Ada que impiden realizar este tipo de análisis, y limita el uso de las tareas de acuerdo con el modelo de concurrencia descrito en el apartado 2. De esta manera se posibilita el uso de tareas en sistemas de alta integridad (ISO 2000), lo que permite elevar el nivel de abstracción y la flexibilidad en la programación de estos sistemas.

La definición del perfil se efectúa mediante *restricciones*, un elemento del lenguaje Ada que permite descartar o limitar el uso de determinadas partes del lenguaje. La mayoría de estas restricciones se pueden comprobar durante la compilación,lo que simplifica el uso del perfil.

El modelo de concurrencia que define el perfil de Ravenscar permite el uso de tareas y objetos protegidos estáticos, un máximo de una entrada con una barrera simple (formada por una variable booleana local) en los objetos protegidos, el reloj de tiempo real, retardos absolutos, planificación de tareas con prioridades estáticas y desalojo, acceso a objetos protegidos mediante un protocolo de techo de prioridad, manejadores de interrupciones estáticos, y algunos otros elementos del lenguaje. Este modelo es, por tanto, equivalente al descrito en el apartado 2, y complementado con una selección adecuada de restricciones de la parte secuencial del lenguaje que facilite el uso de las técnicas de análisis estático que se requieran en el proceso de certificación, permite realizar sistemas de tiempo real completos manteniendo un alto grado de integridad (Vardanega 2003).

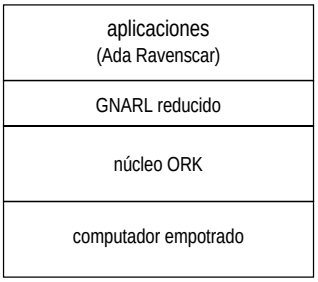

Figura 1. Arquitectura de GNAT/ORK.

Como muestra de la capacidad expresiva del perfil, los listados 1 y 2 contienen ejemplos de realización de tareas periódicas y esporádicas. Se han definido otros esquemas de programación que facilitan la realización de sistemas de tiempo real mediante este perfil (ISO 2005).

4. EL NÚCLEO ORK

4.1 Funcionalidad

ORK *(Open Ravenscar real-time Kernel)* (de la Puente *et al.* 2000, Zamorano and Ruiz 2003) es un núcleo de sistema operativo de tiempo real desarrollado en la Universidad Politécnica de Madrid para dar un soporte específico al perfil de Ravenscar de Ada. Su funcionalidad es la mínima necesaria para este perfil, por lo que carece de muchas de las características de otros sistemas operativos, en aras de la simplicidad y la robustez.

ORK se utiliza conjuntamente con GNAT [8], un compilador libre para Ada integrado en GCC. El núcleo enlaza con una version reducida de la biblioteca de ejecución de GNAT (GNARL), adaptada al perfil de Ravenscar. La figura 1 muestra la arquitectura de ejecución de las aplicaciones basadas en GNAT y ORK.

Las principales funciones que proporciona ORK son:

Concurrencia. ORK soporta hebras *(threads)*, que se crean al comienzo de la ejecución del programa y no pueden terminar. La ejecución de las hebras se planifica mediante prioridades fijas con desalojo. Las hebras de ORK realizan directamente las tareas de Ada con las restricciones del perfil de Ravenscar.

Sincronización. ORK proporciona dos tipos de mecanismos de sincronización:

- *Cerrojos (mutex)*. Un cerrojo es un semáforo binario que permite que una hebra ejecute una sección de código en exclusión mutua con respecto a otras hebras. Los cerrojos se utilizan para asegurar la exclusión mutua en las operaciones de los objetos protegidos de Ada.

[8] *GNU-NYU Ada Translator*.

Listado 1. Ejemplo de tarea periódica.

```
task Periodic is
   pragma Priority (...);
end Periodic;

task body Periodic is
   Next_Cycle : Ada.Real_Time.Time := Ada.Real_Time.Clock;
   Period     : constant Time_Span := Ada.Real_Time.Milliseconds(...);
begin
   loop
      delay until Next_Cycle;
      Read_Variables;
      Compute_Control_Action;
      Write_Control_Actions;
      Next_Cycle := Next_Cycle + Period;
   end loop;
end Periodic;
```

Listado 2. Ejemplo de tarea esporádica.

```
task Sporadic is
   pragma Priority (...);
end Sporadic;

protected Event is
   pragma Priority (...);
   entry Wait;
   procedure Signal; -- invocado por otra tarea o por una interrupción
end Event;

task body Sporadic is
begin
   loop
      Event.Wait;
      Sporadic_Action;
   end loop;
end Sporadic;
```

El perfil de Ravenscar impone el uso de un protocolo de techo de prioridad inmediato *(ceiling locking)* en la ejecución de las operaciones protegidas. Este protocolo se realiza en ORK de una forma muy eficiente, gracias a las restricciones del modelo de tareas de Ravenscar. Cada cerrojo tiene asignado un techo de prioridad, y las operaciones de bloquear y liberar un cerrojo consisten únicamente en elevar la prioridad de la hebra que se está ejecutando hasta ese techo, y restaurar después la prioridad original, respectivamente.

- *Variables de condición*. Las variables de condición proporcionan un mecanismo para que una hebra se suspenda hasta que otra hebra avise de que se ha cumplido una condición. Se utilizan para implementar la espera en las barreras de los objetos protegidos.

  El perfil de Ravenscar permite una sola entrada, como máximo, en cada objeto protegido, con una barrera consistente en una variable booleana simple. Además, prohibe que haya más de una tarea esperando en un barrera (es decir, prohibe que se formen colas). Estas restricciones permiten una implementación muy simple de las variables de condición.

Tiempo. El núcleo proporciona un reloj de tiempo real, que utiliza un mecanismo muy eficiente para obtener la máxima precisión posible del temporizador de hardware que se usa como base. También proporciona una operación de retardo absoluto, que implementa directamente la instrucción delay until de Ada.

Interrupciones. El núcleo proporciona el soporte de bajo nivel para las interrupciones (básicamente la gestión de los vectores de interrupción y una rutina de servicio de interrupciones (ISR) elemental), junto con una operación que permite enlazar una interrupción con un procedimiento protegido de forma estática. Esta operación da soporte directo a la implementación del pragma Attach_Handler de Ada.

Tabla 1. Medidas de ORK

Número de instrucciones	1361 478 82	Ada Ensamblador C
Tamaño del código binario	15	KB
Tamaño mínimo de un programa	94 133	KB (sin tareas) KB (con tareas)
Cambio de contexto	528	ciclos (26,5 μs)
Latencia de interrupciones	1708	ciclos (85,4 μs)

4.2 Diseño del núcleo

El núcleo ORK está compuesto por un conjunto de paquetes, organizados jerárquicamente a partir de un paquete raíz denominado Kernel (figura 2). Los restantes paquetes realizan funciones de concurrencia, gestión del tiempo real, gestión de interrupciones, y gestión de memoria. Los parámetros de configuración se agrupan en un paquete separado para poderlos modificar fácilmente, y las funciones dependientes del procesador y de los periféricos están encapsuladas en dos paquetes (de la Puente *et al.* 2001).

ORK está escrito en Ada, excepto en las partes en las que es imprescindible el uso del lenguaje ensamblador y, en algunos casos, C. Se ha utilizado un subconjunto seguro secuencial del lenguaje, similar al definido en SPARK (Barnes 2003). Todas las funciones del núcleo son puramente secuenciales, sin concurrencia interna, de tal forma que todas las operaciones se ejecutan por la hebra correspondiente a la tarea que las invoca. Para asegurar la exclusión mutua entre tareas en las operaciones del núcleo se inhiben las interrupciones durante la ejecución de las mismas. Los tiempos de ejecución de estas operaciones se mantienen siempre acotados, y en cualquier caso son de escasa duración, gracias a la cuidadosa codificación de todas las operaciones.

Como indica la figura **??**, el núcleo se ejecuta directamente sobre un computador empotrado, sin necesidad de ningún otro sistema operativo. La versión original de ORK se ejecuta en un computador ERC32, que es la versión protegida frente a radiaciones de la arquitectura SPARC 7, aunque hay también una versión de ORK para arquitectura PC.

4.3 Métricas

La tabla 1 muestra algunas medidas del código, que confirman la hipótesis de que las restricciones impuestas por el perfil de Ravenscar permiten la implementación de la parte concurrente de Ada mediante un núcleo de tamaño reducido y eficiente.

Vardanega *et al.* (2005) han obtenido metricas completas sobre el comportamiento temporal de ORK, que pueden utilizarse para efectuar un análisis temporal preciso de las aplicaciones desarrolladas sobre este núcleo.

5. DESARROLLO DE SISTEMAS CON ORK

5.1 Proceso de desarrollo de software

Normalmente se utiliza el conocido *modelo en V* (figura 3) (Sommerville 2004) para desarrollar sistemas de alta integridad. Este ciclo combina *etapas de desarrollo* y *etapas de ensayos* (test) de forma sistemática, de tal manera que a las actividades de desarrollo en un nivel de abstracción dado corresponden otras de ensayo, que fomran parte del proceso de validación del sistema.

Este modelo de desarrollo ha sido criticado por su elevado coste, al posponer todas las actividades de validación hasta que está escrito todo el código (después de la etapa de diseño detallado). La posibilidad de realizar actividades de validación (en particular, anáisis estático) con anterioridad, y en particular en las etapas de diseño arquitectónico y diseño detallado, abre la vía a modelos de desarrollo más eficientes y mejor adaptados a garantizar los requisitos de fiabilidad de los sistemas de alta integridad. Por ejemplo, Vardanega (1999) propone un modelo iterativo para las etapas de diseño basado en la generación automática de código y el uso de técnicas de análisis estático de propiedades temporales (figura 4). Este modelo permite detectar fallos en el comportamiento temporal del sistema desde las primeras etapas de diseño, y se puede extender con otros métodos de análisis estático.

El concepto de *modelo de cómputo* es central en este enfoque. El modelo de cómputo determina las características de la plataforma de ejecución y del software de aplicación, de tal forma que hagan posible la aplicación de las técnicas de análisis estático utilizadas. En el caso de sistemas de tiempo real críticos, el uso de un modelo de cómputo como el que subyace al perfil de Ravenscar permite el uso de técnicas de análisis temporal basadas en el cálculo del tiempo de respuesta de las tareas (Klein *et al.* 1993, Burns and Wellings 2001).

Este modelo de desarrollo está bien soportado por métodos como HRT-UML (Mazzini *et al.* 2003) y por herramientas de análisis como MAST (Medina *et al.* 2001), que pueden usarse conjuntamente con GNAT/ORK para construir sistemas de tiempo real de alta integridad.

6. CONCLUSIONES

El núcleo ORK y el compilador GNAT permiten la construcción de sistemas de tiempo real fiables de una forma accesible y con un coste reducido. El hecho de que se trate de herramientas de software libre permite disponer del código fuente y extender el proceso de validacion del software a la biblioteca de ejecución y al núcleo de tiempo real, lo que no es posible en entornos cerrados.

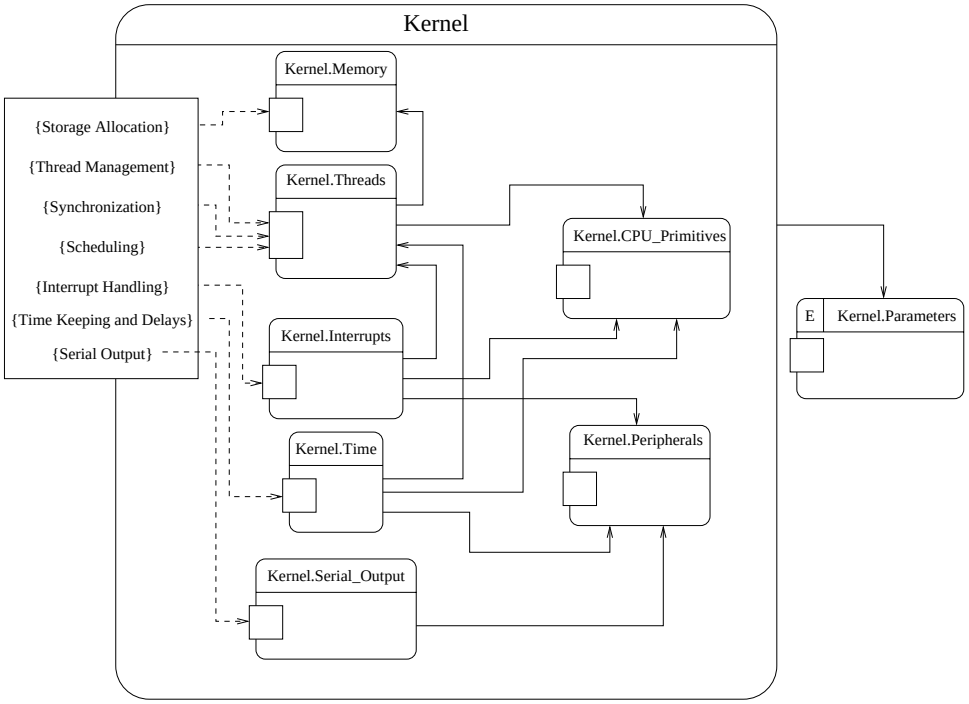

Figura 2. Estructura del núcleo ORK.

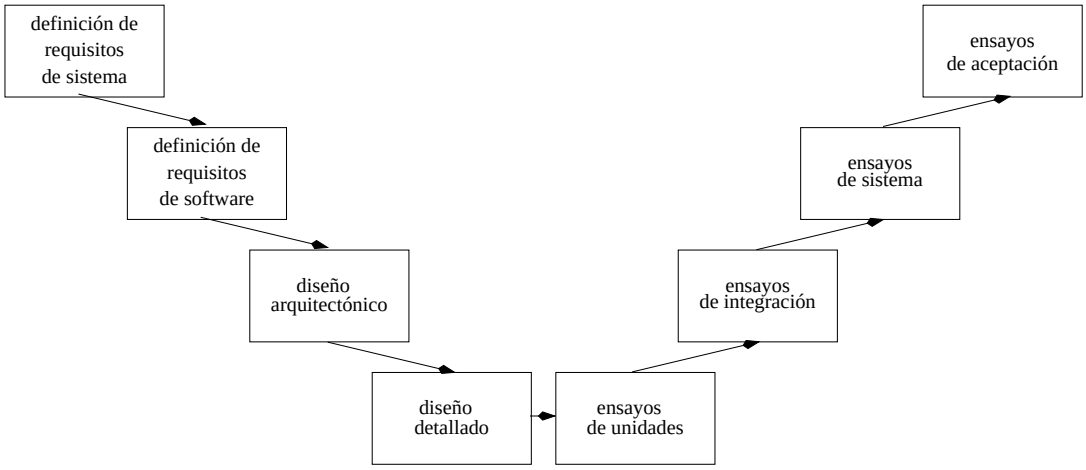

Figura 3. Modelo de desarrollo en V.

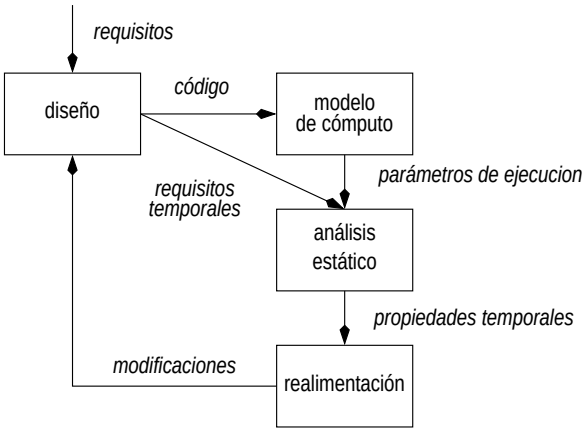

Figura 4. Modelo iterativo de diseño de software.

La experiencia inicial en algunos proyectos del sector espacial europeo (Vardanega and Caspersen 2001, Vardanega *et al.* 2001) ha aportado resultados muy prometedores (Vardanega 2003), y ha permitido mejorar el diseño y las prestaciones del núcleo. Está previsto extender ORK para soportar Ada 2005, en particular los relojes de tiempo de ejecución (de la Puente and Zamorano 2003, Zamorano *et al.* 2004).

Las versiones públicas de GNAT y ORK para PC y SPARC (ERC32) están disponibles en la Universidad Politécnica de Madrid [9] . La empresa Ada Core Technologies [10] comercializa una versión profesional para ERC32.

AGRADECIMIENTOS

Los autores desean reconocer la colaboración de Alejandro Alonso, Ángel Álvarez, José Ruiz, Rodrigo García, Ramón Fernández Marina, Andrés Arias y Juan Manuel Dodero, en el desarrollo del núcleo ORK, así como de Miguel Muñoz, José Antonio Pulido, Santiago Urureña y Santiago Palomino en su mantenimiento y adaptación posteriores. También desean agradecer la colaboración de Jesús González Barahona, Pedro de las Heras, José Centeno y Vicente Matellán en la adaptación de GDB para GNAT/ORK, de Andrés Borges, Jesús Borruel y Juan Carlos Morcuende, de CASA Espacio, en la validación del software, y de Alan Burns y Andy Wellings, de la Universidad de York, en la especificación de requisitos y la clarificación del perfil de Ravenscar. Jorge Amador, Tullio Vardanega y Morten Nielsen, del Centro de Investigación de la Agencia Europea del Espacio (ESA/ESTEC) han contribuido también al proyecto en todas sus etapas con interesantes observaciones y sugerencias.

REFERENCIAS

Ada (2005). *Ada Reference Manual. Language and Standard Libraries. Consolidated Standard ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1 (Draft 13)*. Available on `http://www.adaic.com/standards/rm-amend/html/RM-TTL.html`.

Audsley, N.C., A. Burns, M.F. Richardson and A.J. Wellings (1992). Deadline monotonic scheduling theory. In: *Real-Time Programming 1992. Proceedings of the IFAC/IFIP Workshop* (Luc Boullart and Juan A. de la Puente, Eds.). Pergamon Press.

Audsley, N.C., K. Tindell and A. Burns (1993). The end of the line for static cyclic scheduling?. In: *5th Euromicro Workshop on Real-Time Systems*. IEEE Computer Society Press.

Audsley, Neil, Ian Bate and Alan Burns (1996). Putting fixed priority scheduling theory into engineering practice for safety critical applications. In: *IEEE Real-Time Technology and Applications Symposium*. IEEE Computer Society Press.

Baker, T.P. and A. Shaw (1989). The cyclic executive model and Ada. *Real-Time Systems*.

Barnes, John (2003). *High Integrity Software: The SPARK Approach to Safety and Security*. Addison Wesley.

Burns, Alan and Andy J. Wellings (2001). *Real-Time Systems and Programming Languages*. 3 ed.. Addison-Wesley.

Burns, Alan, Brian Dobbing and George Romanski (1998). The Ravenscar tasking profile for high integrity real-time programs. In: *Reliable Software Technologies — Ada-Europe'98* (Lars Asplund, Ed.). number 1411 In: *LNCS*. Springer-Verlag. pp. 263–275.

CEN (1997). *EN-50128:1997. Railway Applications: Software for Railway Control and Protection Systems*.

Cornhill, D. and L. Sha (1987). Priority inversion in Ada or what should be the priority of an Ada server task?. *Ada Letters*.

de la Puente, Juan A., José F. Ruiz and Juan Zamorano (2000). An open Ravenscar real-time kernel for GNAT. In: *Reliable Software Technologies — Ada-Europe 2000* (Hubert B. Keller and Erhard Ploedereder, Eds.). number 1845 In: *LNCS*. Springer-Verlag. pp. 5–15.

de la Puente, Juan A., Juan Zamorano, José F. Ruiz, Ramón Fernández and Rodrigo García (2001). The design and implementation of the Open Ravenscar Kernel. *Ada Letters*.

de la Puente, Juan A. and Juan Zamorano (2003). Execution-time clocks and Ravenscar kernels. *Ada Letters* **XXIII**(4), 82–86.

ECS (2002). *ECSS-E-40B Draft 1. Space Engineering — Software*. Available from ESA.

IEC (1986). *IEC 880:1986. Software for computers in the safety systems of nuclear power stations*.

ISO (1995). *Ada 95 Reference Manual: Language and Standard Libraries. International Standard ANSI/ISO/IEC-8652:1995*. Available from Springer-Verlag, LNCS no. 1246.

ISO (2000). *TR 15942:2000 — Guide for the use of the Ada programming language in high integrity systems*.

ISO (2005). *TR 24718:2005 — Guide for the use of the Ada Ravenscar Profile in high integrity systems*. Based on the University of York Technical Report YCS-2003-348 (2003).

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

[9] `http://www.dit.upm.es/ork`.
[10] `http://www.act-europe.com`.

Liu, Jane W. S. (2000). *Real-Time Systems*. Prentice-Hall.

Locke, C. Douglass (1992). Software architectures for hard real-time applications: Cyclic executives vs. fixed priority executives. *Real-Time Systems* **4**(1), 37–53.

Mazzini, Silvia, Massimo D'Alessandro, Marco Di Natale, Andrea Domenici, Giuseppe Lipari and Tullio Vardanega (2003). HRT-UML: Taking HRT-HOOD onto UML. In: *Reliable Software Technologies, Ada-Europe 2003* (Jean-Pierre Rosen and Alfred Strohmeier, Eds.). number 2655 In: *LNCS*. Springer-Verlag. pp. 405–416.

Medina, Julio L., Michael González Harbour and José María Drake (2001). MAST real-time view: A graphic UML tool for modeling object-oriented real-time systems. In: *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS 2001)*. IEEE Computer Society Press. London, UK. pp. 245–256.

RTC (1992). *RTCA SC167/DO18B — Software Considerations in Airborne Systems and Equipment Certification*. Also available as EUROCAE document ED-12B.

Sha, Lui, Ragunathan Rajkumar and John P. Lehoczky (1990). Priority inheritance protocols: An approach to real-time synchronization. *IEEE Tr. on Computers*.

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

Sommerville, Ian (2004). *Software Engineering*. 7 ed.. Pearson Education.

Storey, Neil (1996). *Safety-Critical Computer Systems*. Addison-Wesley Longman.

Vardanega, Tullio (1999). Development of on-board embedded real-time systems: An engineering approach. Technical Report ESA STR-260. European Space Agency. ISBN 90-9092-334-2.

Vardanega, Tullio (2003). Reflections on the use of the Ravenscar profile. In: *Proceedings of the 12th International Ada Real-Time Workshop (IRTAW12)*.

Vardanega, Tullio and Gert Caspersen (2001). Using the Ravenscar Profile for space applications: The OBOSS case. In: *Proceedings of the 10th International Workshop on Real-Time Ada Issues* (Michael González-Harbour, Ed.). Vol. XXI. Ada Letters. pp. 96–104.

Vardanega, Tullio and Jan van Katwijk (1999). Productive engineering of predictable embedded real-time systems: The road to maturity. *Information and Software Technology* **40**, 745–764.

Vardanega, Tullio, Juan Zamorano and Juan Antonio de la Puente (2005). On the dynamic semantics and the timing behaviour of Ravenscar kernels. *Real-Time Systems* **29**(1), 1–31.

Vardanega, Tullio, Roberto García and Juan Antonio de la Puente (2001). An application case for Ravenscar technology: Porting OBOSS to GNAT/ORK. In: *Reliable Software Technologies — Ada-Europe 2001* (Alfred Strohmeier and Dirk Craeynest, Eds.). number 2043 In: *LNCS*. Springer-Verlag. pp. 392–404.

Zamorano, Juan, Alejandro Alonso and Juan Antonio de la Puente (1997). Building safety critical real-time systems with reusable cyclic executives. *Control Engineering Practice*.

Zamorano, Juan, Alejandro Alonso, José Antonio Pulido and Juan Antonio de la Puente (2004). Implementing execution-time clocks for the Ada Ravenscar profile. In: *Reliable Software Technologies - Ada-Europe 2004* (Albert Llamosí and Alfred Strohmeier, Eds.). Vol. 3063 of *LNCS*. Springer-Verlag. ISBN 3-540-22011-9.

Zamorano, Juan and José F. Ruiz (2003). GNAT/ORK: An open cross-development environment for embedded Ravenscar-Ada software. In: *Proceedings of the 15th IFAC World Congress* (Eduardo F. Camacho, Luis Basañez and Juan Antonio de la Puente, Eds.). Elsevier Press. ISBN 0-08-044184-X.