Revista Iberoamericana de Automática e Informática Industrial
CEA IFAC comité español de automática
RIAI
ISSN: 1697-7912. Vol. 3, Núm. 2, Abril 2006, pp. 7-18
http://riai.isa.upv.es

UNA PANORÁMICA DE LOS SISTEMAS DE TIEMPO REAL

Alfons Crespo, Alejandro Alonso

Universidad Politécnica de Valencia, Valencia, España
Universidad Politécnica de Madrid, Madrid, España

Resumen: El enorme crecimiento de los sistemas empotrados en los últimos años ha repercutido en un renovado interés por los sistemas de tiempo real. En este artículo se pretende analizar los principales temas que tienen un fuerte impacto en el desarrollo de sistemas de tiempo real y apuntar algunos de los aspectos y propuestas más relevantes en este ámbito. Para ello, se realiza un recorrido por distintos temas tales como las políticas de planificación, los lenguajes más apropiados, los sistemas operativos y los sistemas distribuidos de tiempo real. Más que profundizar en cada uno de estos aspectos, se ha buscado el ofrecer los elementos básicos para tener una visión amplia de este tipo de sistemas.

Palabras clave: Sistemas de tiempo real, Informática industrial, Sistemas empotrados

1. INTRODUCCIÓN

Aunque el término tiempo real se utiliza con excesiva frecuencia para un gran número de aplicaciones que tienen una respuesta rápida, la correcta definición de este tipo de sistemas se puede enunciar como sistemas informáticos en los que la respuesta de la aplicación ante estímulos externos debe realizarse dentro de un plazo de tiempo establecido. La predicibilidad del tiempo de respuesta determina que el sistema será capaz de ofrecer una respuesta correcta ante la llegada de un determinado estímulo en un tiempo acotado. Esta característica permite conocer a priori cuál va a ser el comportamiento del sistema en las peores condiciones y poder analizar los tiempos de respuesta del sistema.

Por informática industrial se entiende el uso de los sistemas informáticos en el entorno industrial. Un computador en este entorno deberá ser capaz de leer variables del entorno, a través de los sensores adecuados, y determinar acciones de respuesta adecuada hacia el exterior a través de los actuadores correspondientes.

Mientras que en el mundo real se pueden producir distintos eventos al mismo tiempo, en el computador el tratamiento es secuencial. Así pues, los estímulos externos aparecen en instantes de tiempo no predecibles y, a veces, de forma paralela, que son recibidos por el computador que los tiene que resolver de forma secuencial y atendiendo a las restricciones de respuesta que tienen.

En la literatura de los sistemas de tiempo real se distinguen los sistemas de tiempo real críticos y acríticos. Los sistemas de tiempo real críticos son aquellos en los que las acciones del sistema se han de producir de forma obligatoria dentro del plazo especificado. Estos sistemas comportan que si se producen fallos de plazo, el sistema puede evolucionar de forma catastrófica. Ejemplos de este tipo de sistemas son los aeronáuticos, control de satélites, robots en operaciones críticas, control de procesos industriales altamente contaminantes, etc.

En los sistemas de tiempo real acríticos la pérdida puntual de alguno de los plazos puede producir que el sistema tenga un funcionamiento degradado durante un intervalo que debe acotarse y permitir que el sistema recupere de forma rápida el funcionamiento normal. Si no se acota este intervalo, el sistema puede inestabilizarse. Ejemplos de este tipo de sistemas son los sistemas de control industrial lentos, sistemas multimedia, transmisión de voz, etc.

En este artículo se realiza un repaso de los principales temas que tienen un fuerte impacto en el desarrollo de sistemas de tiempo real y se apuntan algunos de los aspectos y propuestas más relevantes en este ámbito. Después de una breve introducción a los sistemas de tiempo real, se analizan los temas de planificación en sistemas críticos, acríticos y sistemas distribuidos. Posteriormente, se abordan las características que ofrecen los lenguajes que permiten desarrollar este tipo de aplicaciones y el soporte de ejecución necesario. Finalmente, se elaboran las conclusiones.

Un análisis más extenso de cada uno de estos temas se puede encontrar en distintos libros de una gran calidad como son (Liu, 2000) y (Burns and Wellings, 2001). También hay que destacar los *Roadmaps* desarrollados en el marco de la red de excelencia ARTIST y que se han publicado en (Bouyssounouse and Sifakis, 2005).

2. CARACTERÍTICAS DE LOS SISTEMAS DE TIEMPO REAL

En el diseño del software del sistema de control, cada uno de los controladores en un sistema multifrecuencia se estructura en una tarea. Así pues, un sistema estará formado por varias tareas que implementan cada uno de los controladores.

El modelo de tarea periódico es un modelo muy conocido y para el cual se han desarrollado varias extensiones caracterizando distintas funcionalidades de los sistemas de tiempo real. Hay diversos métodos y herramientas para el diseño, análisis y validación de sistemas de tiempo real caracterizados por este modelo.

Una tarea está caracterizada por los siguientes parámetros funcionales:

Periodo. Cada tarea es ejecutada de forma regular cada intervalo de tiempo. Específicamente, cada periodo de una tarea T_i, denotado por P_i, es una secuencia de activaciones $(a_1, a_2, a_3, ..., a_n)$ cada una de ellas en un intervalo de tiempo. En concreto, la activación a_k deberá ejecutarse dentro del intervalo denotado por $[(k-1).P_i, k.P_i]$.

Plazo de entrega. El plazo de entrega de una tarea T_i, denotado por D_i, es el intervalo de tiempo máximo que puede transcurrir entre instante a partir del cual debe activarse y su finalización. Normalmente, el plazo de entrega de una tarea seá igual al periodo. Esto quiere decir que cada activación de la tarea debe finalizar su ejecución antes del siguiente periodo. En algunos casos, dependiendo de las restricciones de sistema, el plazo de entrega será menor o mayor que el periodo.

Desfase Inicial. El desfase inicial de una tarea T_i denota el desfase que tiene el inicio de la primera activación de la tarea respecto al tiempo de inicio y se denota como ϕ_i.

Los siguientes parámetros dependen del procesador y del planificador.

Tiempo de cómputo. El tiempo de cómputo de una tarea T_i es el tiempo de CPU necesario, C_i, para completar su ejecución en cada una de las activaciones. Este tiempo depende de la complejidad del algoritmo de control y la velocidad del procesador. El tiempo de cómputo puede variar atendiendo a distintos aspectos: código con ejecución condicional que pueden consumir distinto tiempo o el subsistema de ejecución (memoria cache y pipeline). Esto hace que la ejecución de una activación cualquiera de la tarea T_i pueda variar entre dos límites $[e_i^-, e_i^+]$, que se corresponsen con el mínimo y máximo tiempo de cómputo, respectivamente. El límite superior es el tiempo de peor caso (WCET) y es el que se usa para el análisis de la planificabilidad.

Retardo de inicio. Una actividad de una tarea T_i tiene un retardo de inicio que corresponde con el tiempo en el cual la actividad empieza a ser ejecutada. Este retraso viene determinado por el sistema de ejecución (granularidad del reloj para reconocer el inicio del periodo, la ejecución del planificador y la decisión del planificador sobre qué tarea debe ejecutarse en cada instante) y por las decisiones del planificador. Este retardo se puede modelar mediante un retardo debido a los servicios del núcleo r_{min} y un intervalo $[r_i^-, r_i^+]$ debido a la interferencia de otras tareas . Este término recibe en la literatura el nombre de *jitter* de entrada.

Tiempo de finalización. El tiempo de finalización es el tiempo en el cuál una actividad de una tarea T_i finaliza su ejecución. Este tiempo depende del instante de inicio de la ejecución de la tarea, el tiempo de cómputo necesario, los retrasos que la ejecución de la actividad ha podido sufrir al acceder a datos compartidos y la interferencia de otras tareas más prioritarias. El tiempo de finalización de una actividad se modela mediante un intervalo $[f_i^-, f_i^+]$ que delimita el retardo de finalización y se denomina *jitter* de salida.

La figura 1 muestra de forma gráfica los parámetros de ejecución.

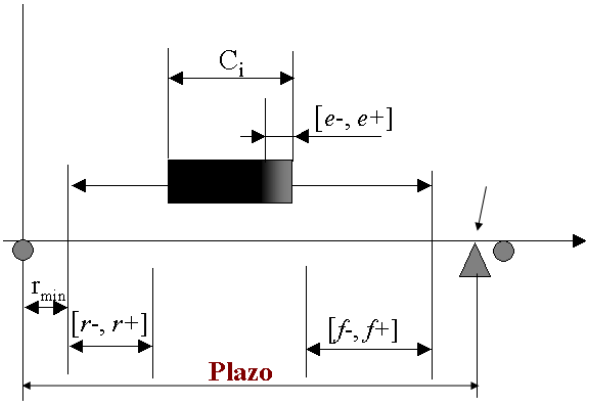

Figura 1. Parámetros de ejecución de una tarea.

En general, un sistema está formado por un conjunto de tareas: $\tau = \{T_1, T_2, T_3, ..T_n\}$ donde n es la cardinalidad de τ. Cada tarea T_i, tiene las siguientes características: $T_i = (C_i, D_i, P_i, \phi_i)$ donde C_i es el tiempo de peor caso de ejecución de la tarea, D_i es el plazo de entrega, P_i es el periodo y ϕ_i es el desfase inicial.

Mientras que las tareas periódicas tienen un comportamiento regular (periódico), las tareas aperiódicas siguen un patrón irregular de activación ya que se activan cuando ocurre un suceso. Las tareas aperiódicas críticas se las denomina esporádicas. Para poder garantizar sus plazos de entrega, es necesario acotar la frecuencia de las activaciones. Para tal fin, se define la separación mínima entre dos sucesos de activación sucesivos, que se denomina T. En el peor caso, a la máxima frecuencia, una tarea esporádica se comporta como una periódica con periodo igual a la separación mínima entre sucesos.

El tiempo más desfavorable de respuesta de las tareas aperiódicas no se puede acotar, dado que no está acotado el número máximo de activaciones en un intervalo de tiempo(frecuencia). Las técnicas de tratamiento de tareas aperiódicas tratan de minimizar su tiempo de respuesta. En la sección siguiente se detalla la integración de estas tareas en el modelo general de tareas.

3. PLANIFICACIÓN DE SISTEMAS CRÍTICOS

La ejecución de varias tareas en un mismo procesador conlleva, por un lado, la asignación del recurso (procesador) a cada una de las tareas durante un tiempo, y por otro, la necesidad a priori de determinar si el conjunto de tareas cumplirá las restricciones temporales impuestas por el usuario.

Un planificador suministra un algoritmo o política para la ordenación de la ejecución del conjunto de actividades de acuerdo a un criterio predefinido. En los sistemas operativos convencionales, las actividades son ejecutadas de acuerdo a criterios de equidad entre las distintas tareas. En cualquier caso, el sistema operativo realiza las operaciones necesarias de gestión de los distintos recursos a un nivel mayor de prioridad que la de las tareas. Esto hace que una tarea urgente tenga que esperar hasta que el sistema operativo haya finalizado el servicio demandado por otra tarea menos urgente y, por lo tanto, no se podrá garantizar el correcto funcionamiento del sistema de control desde el punto de vista temporal.

Es necesario la consideración de las restricciones temporales para poder ejecutar las actividades de tiempo real con garantía y asegurar que el sistema operativo tiene un comportamiento predecible. Los sistemas operativos de tiempo real ofrecen este comportamiento y, por lo tanto, son los apropiados para la ejecución de sistemas de control de tiempo real.

Para garantizar el cumplimiento de las restricciones temporales es necesario realizar un análisis de planificabilidad previo que permita conocer si las restricciones impuestas van a cumplirse en cualquier situación. La planificación en sistemas de tiempo real ha sido uno de los aspectos en los que se han centrado un gran número de grupos de investigación de sistemas de tiempo real en las dos últimas décadas.

La planificación de sistemas de tiempo real se puede realizar en función de cómo y cuándo se toman las decisiones de planificación. Atendiendo a estos aspectos se pueden categorizar los planificadores en dos grupos: planificadores *off-line* cuando la decisión se realiza en la fase de diseño confeccionando un plan cíclico que se seguirá estrictamente durante la ejecución, y planificadores *on-line* cuando las decisiones se toman durante la ejecución.

Planificadores cíclicos. Una primera aproximación al diseño del sistema de tiempo real consiste en analizar la ejecución de las tareas considerando una ejecución estática decidida a priori. Este tipo de planificación se denomina cíclica (*Cyclic Executives*) y consiste en almacenar en una tabla el instante en el que debe ejecutarse cada una de las activaciones de las tareas o diseñar un secuenciador de las distintas actividades a realizar.

Este esquema resulta simple cuando el sistema de control tiene periodos iguales o múltiplos entre sí. En caso de periodos muy desiguales y gestión de eventos aperiódicos el cálculo de la tabla puede resultar extremadamente largo y complejo. En este mismo sentido, cuando pueden existir retrasos en la llegada de los datos del

proceso, se produce una indeterminación que restringe el uso de este tipo de planificadores.

En el caso más complejo, la búsqueda de un plan requiere encontrar un intervalo submúltiplo del hiperperiodo (ciclo menor) que permita la ejecución de las tarea cumpliendo los plazos de éstas. Este problema es computacionalmente complejo (NP-Hard) y requiere una serie de reglas que permitan reducir su complejidad. Un buen análisis del diseño de este tipo de planificadores y sus implicaciones se puede encontrar en (Baker and Shaw, 1989).

Planificación basada en prioridades. Los algoritmos de planificación basados en prioridades seleccionan la tarea a ejecutar entre las disponibles atendiendo a un criterio básico (prioridad). Esta prioridad puede tener un carácter fijo (se fija por el diseñador atendiendo a algún criterio determinado: semántico, periodos o plazos) o dinámico (es determinada por el sistema atendiendo a algún criterio: la tarea más urgente), lo que determina el tipo de algoritmo de planificación basada en prioridades fijas (FPS) o planificación basada en prioridades dinámicas (EDF).

Recientemente, se ha publicado el artículo (Sha *et al.*, 2004) que ofrece una perspectiva histórica de la teoría de la planificación.

3.1 Planificación basada en prioridades fijas

En el trabajo (Liu and Layland, 1973), se propusieron dos algoritmos óptimos basados en prioridades para la ejecución de tareas con plazos iguales a periodos. El primer algoritmo propuesto consideraba una asignación de prioridades fijas creciente con la frecuencia de las tareas (*rate monotonic*, RM), y otro con asignación de prioridades dinámicas atendiendo al plazo más corto (*earliest deadline first*, EDF). En este artículo se demostraba la posibilidad de poder determinar si un conjunto de tareas era planificable o no (todas las tareas cumplirán los plazos). La condición de planificabilidad se basaba en la utilización y era una condición suficiente pero no necesaria. Este trabajo puso las bases de la teoría de planificación tal y como se la conoce actualmente.

Aunque las restricciones impuestas en el análisis fueron muy estrictas (tareas periódicas con periodos iguales a los plazos, tareas independientes, cambio de contexto despreciables frente a la carga de las tareas y la no auto-suspensión de las tareas), en posteriores trabajos estas restricciones se han ido eliminando e incluyendo en el modelo.

A continuación se resumen algunos de los trabajos más significativos para planicadores de prioridad fija.

Cálculo exacto del tiempo de respuesta. En los trabajos anteriores se analizaba el sistema en base a la utilización, en (Lehoczky *et al.*, 1989) y (Audsley *et al.*, 1993*b*), se propone un método de cálculo exacto de la respuesta de una tarea en la situación de peor caso (instante crítico). Este algoritmo tiene un coste pseudo-polinomial y permite determinar si el sistema es planificable o no bajo la condición que el tiempo de respuesta de peor caso de cada una de las tareas sea menor o igual que su plazo de entrega.

Periodos distintos de plazos. La restricción de periodos iguales a plazos es analizada posteriormente y en (Audsley, 1990) se propone una asignación monotónica con el plazo (*Deadline Monotonic*) y el cálculo exacto de los tiempos de respuesta en el peor caso.

Tareas dependientes. La inclusión recursos compartidos en el análisis de planificación fue objeto de una serie de protocolos basados en herencia de prioridad. En (Sha *et al.*, 1990) se propone y analiza el protocolo básico de herencia de prioridad. También en este mismo artículo se propone una mejora basada en protocolos de techo de prioridades que asignan al recurso una prioridad techo (máxima prioridad de las tareas que usan el recurso) que es heredada por la tarea. Otro protocolo de gestión de recursos compartidos es debido a (Baker, 1990) y se basa en un protocolo de pila.

Servidores aperiódicos. La integración de tareas aperiódicas se realiza mediante este tipo de servidores. Éstos son tareas periódicas que con un tiempo de cómputo limitado por periodo, dan servicio a tareas aperiódicas mejorando sus tiempos de respuesta e integrándolas en el modelo de tareas periódicas (Sprunt *et al.*, 1989).

El libro (González-Harbour *et al.*, 1993) ofrece un amplio detalle de los métodos para el análisis y diseño de sistemas basado en RMA (*Rate Monotonic Analysis*).

3.2 Planificación basada en prioridades dinámicas

En la planificación basada en prioridades dinámicas las propuestas han surgido de forma similar a las fijas. El principal algoritmo de este grupo (EDF) es óptimo entre cualquier algoritmo de los basados en prioridades. En (Dertouzos and Mok, 1989) se propuso otro algoritmo óptimo que asigna prioridades mayores a las tareas con menor laxitud (*Least Laxity First LLF).* De forma similar a la planificación con prioridades fijas, a continuación se realiza un breve recorrido por las principales aportaciones.

Cálculo exacto del tiempo de respuesta. Se han propuesto varios métodos exactos y aproximados para el cálculo del tiempo de respuesta.

Entre las propuestas destacan las de (Baruah *et al.*, 1990) y (Ripoll *et al.*, 1996). En este último se realiza un análisis basado en el instante crítico de forma similar a lo realizado en planificación con prioridades fijas.

Recursos compartidos. En EDF se han propuesto varios protocolos basados en herencia de prioridad (Chen and Lin, 1990) y basados en pila (Baker, 1991). Este último es uno de los más usados en EDF por sus propiedades y eficiencia.

Servidores aperiódicos. Al igual que en el método anterior, se han propuesto distintas políticas para la inclusión de tareas aperiódicas en el modelo de tareas periódico (Ghazalie and Baker, 1995). Una de las propuestas más interesantes por su eficiencia es de la servidores de ancho de banda (Spuri and Buttazzo, 1994) y (Abeni and Buttazzo, 1998).

3.3 Planificación y control

En los sistemas de control en los que la adquisición de datos de los sensores se realiza al inicio del tiempo de cómputo y la actuación en los instantes finales, se considera que los retardos iniciales y finales pueden tener una influencia importante en el sistema de control. La variabilidad que sufre una tarea en sus distintas activaciones se corresponde con su *jitter* de entrada, mientras que la de su finalización y, por lo tanto el envío de la acción de control, con su *jitter* de salida. Para un sistema planificable, esta variabilidad puede afectar al resultado sobre el proceso a controlar. El algoritmo de control programado en la tarea es, normalmente, diseñado sin tener en cuenta estos retardos debidos a la ejecución. Aunque el sistema es correcto desde el punto de vista temporal, estos retardos pueden producir en determinados casos pérdida de prestaciones y funcionamiento degradado en el sistema de control.

En los últimos años se ha realizado un esfuerzo conjunto entre grupos de control y de tiempo real para reducir los efectos de estos retardos sobre el sistema a controlar mediante un codiseño del sistema teniendo en cuenta los aspectos de control y los condicionantes de la implementación. En (ARTIST2, 2005*b*) se realiza una descripción detallada de los aspectos ligados a la relación entre la planificación y el control describiéndose un conjunto de técnicas utilizadas para reducir los efectos de los retardos sobre el sistema de control. A continuación se detallan algunos de los temas que están ligados a esta problemática.

Consideración del modelo del controlador. Uno de los primeros artículos en el codiseño del control y la planificación fue (Seto *et al.*, 1996). En este artículo se realizaba un diseño óptimo de los periodos de las tareas teniendo en cuenta una función de coste bajo condiciones de planificabilidad. Otra propuesta es la de medir el *esfuerzo de control* (Albertos and Olivares, 1999) de cada bucle de control que determina su sensibilidad ante retardos e incorporar este criterio para asignar la prioridad de cada tarea bajo la restricción de la planificabilidad del sistema (Albertos *et al.*, 2000). Otros trabajos en esta línea son los de (Seto *et al.*, 1998), (Eker *et al.*, 2000) y (Ryu *et al.*, 1997).

Reducción de los retardos. La reducción de retardos desde el punto de vista de planificación ha sido objeto de varios trabajos. En (Audsley *et al.*, 1993*a*) se propone un esquema de prioridades duales para que algunas tareas reduzcan su tiempo de finalización incrementando su prioridad a partir de un cierto tiempo. En (Baruah *et al.*, 1997) se propone una minimización del *jitter* de salida de tareas.

Particionado de tareas. Otras propuestas para la reducción de los retardos proponen un particionado de tareas asignando esquemas de prioridades asociados a cada partición. En (Crespo *et al.*, 1999) se propone la partición de tareas en tres clases (entrada, cálculo y salida) asignado bandas de prioridades a cada clase. En (Cervin, 1999) se propone un método heurístico para la asignación de prioridades a las distintas subtareas.

Aunque no directamente relacionado con las prestaciones de control, otra línea de trabajo en esta temática consiste en incluir algoritmos de control en el planificador para ajustar alguno de los recursos del sistema. Uno de ellos es el uso del procesador a través de la medida de los parámetros de le ejecución (carga del sistema, uso del procesador por un conjunto de tareas, pérdidas de plazo, etc.). Este aspecto, llamado *feedback scheduling*, ha sido objeto de un gran número de trabajos. Una visión detallada de esta problemática se puede encontrar en (ARTIST2, 2005*a*).

4. SISTEMAS OPERATIVOS DE TIEMPO REAL

Un sistema operativo de tiempo real es una clase de sistemas operativos que cumple unas determinados requisitos para dar soporte a aplicaciones de tiempo real. Un sistema operativo de tiempo real (RTOS) o Núcleo de tiempo real (RTK) tiene que cumplir las siguientes características:

Multiprogramación. Tiene que dar soporte a varias tareas de aplicación. Las aplicaciones de tiempo real son multitarea y requieren un núcleo que permita crear tareas y gestionarlas atendiendo a los plazos de éstas. La capacidad para expulsar a una tarea de menor prioridad

que otra que solicita su atención (planificación expulsiva) es una característica básica.

Predecibilidad de los servicios. Todos los servicios que ofrece deben ser predecibles y su coste debe ser conocido. Para poder aplicar cualquier análisis de planificabilidad de una aplicación de tiempo real se necesita conocer este coste y despreciarlo o no frente al coste de las tareas de aplicación.

Garantía de ejecución. Para poder garantizar la ejecución de las tareas, el RTOS tiene que poder seleccionar y ejecutar éstas atendiendo a criterios tales como la prioridad de una tarea. La política de planificación es la parte del núcleo encargada de la selección de la tarea. La prioridad de la tarea podrá ser determinada por el usuario (prioridad fija) o ser calculada por el sistema (prioridad dinámica) atendiendo a distintos criterios (urgencia, laxitud, etc.). El número de los niveles de prioridad soportados debe ser como mínimo de 32.

Gestión del tiempo. La gestión del tiempo es fundamental para las aplicaciones de tiempo real. El sistema debe ser capaz de gestionar un reloj que sea siempre creciente (monotónico) y gestionar un número de temporizadores adecuado para permitir el control de los distintos requisitos temporales de las tareas. Funciones para acceder al reloj, operar con él, programar retardos absolutos y relativos, etc., son necesarios para que las aplicaciones especifiquen adecuadamente sus restricciones temporales. Adicionalmente, es aconsejable disponer de relojes de alta resolución y relojes de ejecución asociados a cada tarea para conocer y tomar decisiones sobre el tiempo de ejecución de éstas.

Comunicación y sincronización. Las tareas requieren un modelo para comunicarse y sincronizarse. El sistema operativo debe ofrecer los mecanismos básicos y los protocolos adecuados (basados en herencia de prioridad) para sincronizar y comunicar de forma segura y eficiente las tareas de la aplicación. Estos modelos se basan en esquemas de memoria compartida y paso de mensajes.

Gestión de memoria. Por un lado es importante que el espacio de direccionamiento del núcleo esté protegido respecto a la aplicación. Esto es fundamental para evitar que fallos en la aplicación afecten al propio sistema operativo. Por otro lado, el núcleo tiene que suministrar mecanismos predecibles para la gestión de memoria dinámica que puedan utilizar las aplicaciones.

Interfaz de programación (API). La interfaz de programación determina en gran medida la portabilidad de las aplicaciones. Una de las más usadas es POSIX (IEEE, 1997) que ofrece una criterio estándar de acceso a los servicios del RTOS.

Un RTOS debe ofrecer comportamiento determinista en todos los servicios del sistema. Son especialmente significativas las medidas de ciertas operaciones de bajo nivel que realiza muy frecuentemente el sistema y que requiren, además de estar acotadas, un coste temporal bajo. Algunas de estas medidas críticas son:

- Latencia de la gestión de interrupciones garantizada
- Cambio de contexto acotado
- Baja sobrecarga introducida por el planificador
- Gestión eficiente del reloj y los temporizadores
- Tiempo de respuesta del hardware

Un RTOS facilita la ejecución de un sistema de tiempo real pero no garantiza que el producto final cumpla las restricciones temporales. Esto require un desarrollo del software de aplicación correcto y analizable.

Para seleccionar un RTOS para una aplicación determinada es necesario tener en cuenta no sólo sus características de tiempo real, sino que además se deben considerar otros factores tales como:

- Lenguajes soportados
- Tipo de interfaz de llamadas al sistema (API) siguiendo estándares (POSIX, OSE, ARINC-563, etc.) y por tanto, portabilidad de las aplicaciones
- Certificación
- Estructura basada en componentes que permita la generación de sistemas seleccionando aquellos elementos necesarios para una aplicación determinada (sistemas empotrados).
- Middleware en el que se integra
- Disponibilidad de amplia gama de manejadores de dispositivos
- Otros aspectos: sistema de ficheros, pila de comunicaciones, etc.

Existen un gran número de núcleos o sistemas operativos de tiempo real que cumplen los requisitos anteriormente citados. Dentro de esta amplia gama, se pueden distinguir aquellos que son comerciales y los que son de código abierto. Respecto a los RTOS comerciales, en (Timmerman, 2000) se puede encontrar una amplia lista de RTOS y una evaluación completa de ellos. En una reciente publicación (Melanson and Tafazoli, 2003) se realizó una evaluación de distintos aspectos: núcleo, planificación, modelo de tareas, API, etc., puntuando distintas características. Los resultados de esta evaluación muestran a QNX/Neutrino y OS-9 como los mejores, seguidos de OSE, Lynx OS y VxWorks.

En cuanto a los sistemas operativos de tiempo real basados en código abierto, su número es también numeroso. En (Ripoll *et al.*, 2002) se puede encon-

trar una lista de criterios y una comparación de las características de los basados en Linux con los más relevantes comerciales. De entre los RTOS de código abierto basados en Linux destacan RTLinux-GPL y RTAI, que incluyen el núcleo como módulo en el sistema operativo Linux permitiendo que coexistan un entorno de ejecución de tiempo real y otro, Linux, de propósito general. Otros, como KURT y TimeSys Linux, añaden mejoras al núcleo estándar de Linux para reducir la latencia e incrementar la resolución de los relojes y planificación de tiempo real. De los no basados en Linux destacan principalmente RTEMS y eCOS.

Es de destacar los desarrollos de núcleos por grupos españoles y su apuesta por el código abierto. El grupo de Sistemas de Tiempo Real de la Universidad Politécnica de Madrid, ha desarrollado un núcleo que cumple el perfil Ravenscar mínimo para aplicaciones de alta integridad. El Open Ravenscar Real-Time Kernel (ORK) (Puente *et al.*, 2000) es un núcleo de tamaño reducido y baja complejidad que puede ser usado para el desarrollo de aplicaciones de tiempo real crítico y que ha sido desarrollado con un subconjunto del lenguaje Ada-95 compatible con el perfil Ravenscar. El grupo de Sistemas de Tiempo Real de la Universidad de Cantabria ha desarrollado un núcleo totalmente en Ada que cumple el perfil de POSIX mínimo. El núcleo desarrollado, denominado MaRTE OS (Aldea and González-Harbour, 2001), ha sido ampliamente reconocido y utilizado para una amplia gama de desarrollos. Finalmente, el grupo de Sistemas de Tiempo Real de la Universidad Politécnica de Valencia ha mejorado las prestaciones de RTLinux-GPL y ha desarrollado versiones de este núcleo para empotrar en sistemas sin la necesidad de incluir el sistema operativo Linux. Esta versión del núcleo, denominado embedded RTLinux (eRTL), ha sido desarrollado en el marco del proyecto europeo OCERA (OCERA, 2002).

5. LENGUAJES DE TIEMPO REAL

La programación de sistemas de tiempo real se ha hecho clásicamente mediante lenguajes sin características específicas para sistemas de tiempo real. Para poder utilizar las abstracciones adecuadas para su programación es necesario utilizar llamadas a un sistema operativo que las proporcione. El uso de lenguajes específicos de tiempo real presenta algunas ventajas sobre este enfoque:

- Portabilidad: la codificación del programa y su coportamiento no depende del sistema operativo.
- Fiabilidad: el compilador puede realizar comprobaciones de buen uso de las construcciones de tiempo real del lenguaje.

En la literatura científica se pueden encontrar varias referencias a lenguajes de tiempo real, que en la gran mayoría de los casos no se emplearon en sistemas industriales. En la actualidad, hay dos lenguajes que tienen definido soporte para desarrollo de sistemas de tiempo real y usados en la industria: Ada y Java (Burns and Wellings, 2001) (Wellings, 2004).

Uno de los objetivos del diseño del lenguaje Ada fue la programación de sistemas de tiempo real. Aunque la primera versión del lenguaje incluía mecanismos para este fin, tenía importantes carencias (Ada, 1983). La versión actual del lenguaje, aprobada en 1995, mejora las funciones para sistemas de tiempo real, que se incluyen en los anexos de programación de sistemas y de tiempo real (Ada, 1995). Actualmente, hay una revisión que se espera que sea norma ISO próximamente y que incluye mejoras adicionales, como soporte del perfil Ravenscar (orientado al desarrollo de sistemas de alta integridad), relojes de tiempo de cómputo, cuotas de uso de procesador, sucesos temporizados y la posibilidad de definir políticas adicionales de planificación. Estas mejoras deben permitir desarrollar sistemas de tiempo real más sofisticados, aunque esto dependerá de que los fabricantes de compiladores incluyan estas funcionalidades en sus productos.

La definición original de Java no incluía mecanismos para el desarrollo de sistemas de tiempo real. Dado el interés en el lenguaje, hubo investigadores que estudiaron la forma de resolver esta carencia. Estos esfuerzos cristalizaron en dos iniciativas para definir extensiones normalizadas para el desarrollo de sistemas de tiempo real:

- "Real-Time Specification for Java (RTSJ)", desarrollada por el *Real-Time for Java Expert Group*, liderado por Greg Bollella (Bollella *et al.*, 2000) (RTSJ, 2002).
- "Real-Time Core Extension (RTCE)" desarrollada por el *Real-Time Java WG (J-Consortium)*, liderado por Kelvin Nielsen (J-Consortium, 2000).

Ambas versiones presentaron sus normas correspondientes en el año 2000. RTCE ha tenido muy poca aceptación y difusión. Se puede considerar que RTSJ es la norma más empleada. Hay implementaciones comerciales, algunas de las cuales se han empleado en el desarrollo de aplicaciones industriales. Sin embargo, teniendo en cuenta el tiempo transcurrido desde su aprobación, era de esperar un soporte comercial más amplio. Por otro lado, algunas de la extensiones más novedosas eran opcionales y no están soportadas en todas la máquinas virtuales comerciales.

Ada y Java proporcionan los mecanismos fundamentales requeridos a un lenguaje de programación para desarrollar sistemas de tiempo real:

Programación concurrente. Ambos lenguajes permiten la creación de hebras o tareas. Java proporciona un tipo de tarea de tiempo real y permite su caracterización mediante la definición de parámetros de activación, de planificación y de memoria. Así se facilita la programación de tareas periódicas, aperiódicas o esporádicas.

Planificación. La planificación de tareas adecuada es fundamental para desarrollar sistemas de tiempo real predecibles. Ada y Java definen un planificador expulsivo y con prioridades fijas. El estándar nuevo de Ada (Ada 2005) permite al usuario definir políticas de planificación adicionales.

Comunicación entre tareas. Estos lenguajes proporcionan mecanismos de comunicación entre tareas, que se basan en mecanismos de herencia de prioridades para acotar la inversión de prioridades.

Servicios de gestión de tiempo. Ada y Java proporcionan relojes con precisión suficiente y primitivas para establecer retardos. Ada proporciona temporizadores de tiempo absoluto para programar tareas periódicas. Java incluye mecanismos de más alto nivel de abstracción para este fin, que permiten bloquear una tarea hasta el siguiente instante de activación.

Programación de bajo nivel. El desarrollo de sistemas empotrados requiere acceder y programar dispositivos de entrada/salida. Para poder realizar estas operaciones, estos lenguajes permiten interactuar con los registros de los dispositivos o con posiciones de memoria concretas.

El estándar RTSJ incluye algunas funciones avanzadas para el desarrollo de sistemas de tiempo real. A partir de los parámetros de activación de las tareas de tiempo real se plantea la realización de un análisis de planificabilidad automática que permite determinar si las tareas cumplen sus requisitos temporales cuando se crea una nueva. También permite definir el tiempo de cómputo de una tarea, de forma que se puede impedir que use más tiempo de procesador que el declarado. Los parámetros de activación permiten especificar el plazo de terminación de una tarea y el sistema de ejecución detecta y notifica los incumplimientos del mismo.

En conclusión, Ada y Java son dos lenguajes adecuados para el desarrollo de sistemas de tiempo real. Las extensiones de Ada son más maduras y están mejor soportadas en compiladores comerciales. Java proporciona algunas extensiones más innovadoras, aunque el estándar presenta algunas indefiniciones y el número de compiladores comerciales disponibles es limitado.

6. SISTEMAS DE TIEMPO REAL ACRÍTICOS

La mayoría del trabajo de investigación sobre sistemas de tiempo real se ha centrado en los sistemas críticos. Sin embargo, hay muchos sistemas con requisitos temporales en los que el incumplimiento esporádico de plazos de respuesta no constituye un problema grave.

Durante los últimos años, el uso de dispositivos de electrónica de consumo basados en procesadores ha crecido espectacularmente (teléfonos móviles, PDA, reproductores de DVD, etc). Esta tendencia se espera que continúe en sistemas futuros que integren los computadores en nuestras vidas, de forma que éstos no se perciban.

Este tipo de dispositivos se caracterizan por ser dinámicos y tener recursos limitados, requisitos temporales, y requisitos de calidad muy exigentes. Para referirse a estos dispositivos se ha introducido el nombre de sistemas críticos para el negocio (*bussiness-critical*). Su fallo no provoca pérdidas humanas pero, en cambio, pueden producirse cuantiosas pérdidas económicas y de imagen (industria del automóvil o electrónica de consumo). Este es un área de trabajo bastante activo durante los últimos años (Gill *et al.*, 2004) (Zhang *et al.*, 2002) (Welch *et al.*, 1998) (Valls *et al.*, 2002) (Otero-Pérez *et al.*, 2003).

Un problema evidente en el desarrollo de este tipo de sistemas es cómo conjugar la capacidad limitada del hardware, con la necesidad de que las aplicaciones proporcionen una calidad estable en situaciones potenciales de sobrecarga. Un enfoque generalmente aceptado se basa en un modelo de contrato, por el que las aplicaciones proporcionan una cierta calidad mientras que el sistema le garantiza unos recursos mínimos. Es obvio que no es posible que una aplicación proporcione una salida estable y de calidad si no dispone de unos recursos de ejecución determinados.

Una estructura adecuada para realizar este enfoque consiste en disponer de una entidad que se encarga de gestionar el sistema y las aplicaciones. Así, por ejemplo, se pueden identificar dos niveles diferentes de operación del gestor:

Gestión de calidad de servicio. El objetivo es maximizar la calidad que proporciona el sistema. Para ello, hay que decidir la cuota de uso de los recursos que se asigna a las aplicaciones, la cuál dependerá de la importancia de las aplicaciones, los recursos que necesitan para ejecutar y los recursos disponibles. Esta decisión

se basa en un proceso de negociación. En este nivel se pueden incluir operaciones adicionales para monitorizar el correcto funcionamiento del sistema y adaptar su comportamiento a cambios en el entorno.

Gestión de recursos. Este nivel se encarga de comprobar que la asignación de recursos a las aplicaciones es factible, en el sentido de que son suficientes. Otra operación fundamental de este gestor es garantizar el cumplimiento de las cuotas asignadas a las aplicaciones. Esta funcionalidad se basa en la capacidad para medir en tiempo de ejecución los recursos utilizados por las aplicaciones e impedir que empleen más que los asignados.

Las técnicas de planificación de sistemas de tiempo real han constituido la base sobre la que las funciones mencionadas se apoyan. El análisis de planificabilidad de un algoritmo de planificación se puede ver como la comprobación de si el sistema tiene recursos suficientes para permitir que un conjunto de procesos con unos requisitos de cómputo y de comunicación pueden cumplir sus plazos de entrega. Este enfoque se puede emplear para comprobar si una asignación de recursos (procesador) a aplicaciones es factible.

Las aplicaciones tienen que diseñarse de forma diferente para comportarse óptimamente en un entorno como el mencionado. Por una lado, deben ser conscientes de los recursos que necesitan para completar una operación. En general, los requisitos de cómputo pueden ser muy diferentes de una activación a la siguiente, por lo que la aplicación debe ser capaz de adaptar su ejecución a los recursos disponibles en un momento dado. Por otro lado, es conveniente que puedan configurarse para poder proporcionar diferentes niveles de calidad. Cada uno de éstos se puede caracterizar por la calidad que proporcionan, los recursos de cómputo necesarios para proporcionarla y los requisitos temporales. Las capacidades de adaptación de las aplicaciones a situaciones imprevistas o transitorias permitirían un comportamiento más adecuado en este tipo de entornos. El futuro potencial de este enfoque parece muy prometedor, teniendo en cuenta el crecimiento previsto para el tipo de sistemas cuyas características han motivado su planteamiento.

7. SISTEMAS DISTRIBUIDOS DE TIEMPO REAL

Un sistema distribuido de tiempo real está compuesto por un conjunto de computadores conectados en red, donde se ejecutan un conjunto de aplicaciones con requisitos temporales. Este tipo de sistemas se emplean por que es necesaria una capacidad de cómputo que no puede proporcionar un sólo computador, para desarrollar sistemas tolerantes a fallos, por la distribución geográfica del sistema, etc.

Los sistemas distribuidos de tiempo real presentan problemas adicionales a los sistemas centralizados, como:

Planificación de la red. En un sistema centralizado, el procesador suele ser el único recurso que se planifica. En un sistema distribuido los mensajes que intercambian las aplicaciones tienen requisitos temporales. Por tanto, es necesario emplear protocolos de comunicación que permitan acotar el tiempo de entrega de los mismos.

Asignación de tareas a procesadores. El planificador tiene que decidir qué tareas se ejecutan en qué procesador. Una asignación incorrecta puede impedir que algunos procesos cumplan sus plazos de respuesta.

Plazos de respuesta globales. En un sistema distribuido no sólo hay que fijarse en los plazos de respuesta de un proceso o mensaje aislado. Una aplicación distribuida puede ejecutar partes de su código en distintos procesadores y usar mensajes para coordinar la ejecución de las mismas. En este caso, es importante calcular el tiempo de respuesta desde que comienza a ejecutar la aplicación distribuida (también llamada transacción) hasta que se completa su ejecución.

Sincronización de relojes. Los relojes hardware de distintos procesadores pueden tener diferentes valores, motivados por pequeñas derivas del hardware local. Estas diferencias pueden llegar a ser suficientemente grandes como para que lo que en un procesador se considere que un suceso ocurre a tiempo, en otro se interprete que ocurre fuera de plazo. En aquellos sistemas en los que se deba evitar este tipo de interpretaciones divergentes hay que garantizar que la diferencia entre los valores esté acotada.

Tolerancia a fallos. Uno de los motivos de usar un sistema distribuido puede ser la necesidad de garantizar un comportamiento correcto en presencia de fallos. La tolerancia a fallos en sistemas distribuidos de tiempo real es bastante compleja. La razón es que las técnicas de tolerancia a fallos ejecutan una serie de operaciones adicionales para recuperarse o detectar fallos. Dado que no es posible determinar cuando aparecerán, puede ocurrir que estas operaciones impidan a algún proceso cumplir sus requisitos temporales.

A continuación se van a presentar dos enfoques de desarrollo de sistemas distribuidos de tiempo real crítico. En sistemas distribuidos acríticos se puede emplear la aproximación descrita en la sección 6. En este caso, la red es un recurso a gestionar.

Se asigna ancho de banda a las aplicaciones siguiendo principios similares a los usados para el procesador. Las aplicaciones deben especificar los recursos de red necesarios para proporcionar unos resultados con calidad suficiente.

7.1 Sistemas dirigidos por tiempo o cíclicos

Estos sistemas se caracterizan por la planificación estática de la red y los procesadores. Los instantes de activación de las tareas o de envío de los mensajes se almacenan en tablas cíclicas, que se construyen a priori.

El protocolo de comunicación TTP (*Time Triggered Protocol*) es un protocolo adecuado para este tipo de sistemas. Es del tipo TDMA (*Time Division Multiple Access*), dado que el acceso al medio se divide en rodajas de tiempo. A cada mensaje se le asigna el instante en que debe transmitirse, que se almacena en una tabla. De esta forma, todos los nodos saben cuándo tienen que transmitir o recibir un mensaje de un nodo determinado.

Esta característica se puede emplear para saber cuándo un nodo ha fallado. Si un mensaje no se recibe cuando se debía, de acuerdo con las tablas de comunicación, es que hay un problema con el nodo emisor o con la red. A partir de la detección de este problema, se pueden activar acciones de recuperación de fallos.

En (Kopetz, 1997) y (Kopetz, 1998) se describen las técnicas básicas, el protocolo de comunicación TTP y un enfoque de desarrollo de sistemas global basado en estos principios. Los sistemas así construidos tendrán un comportamiento predecible y tolerante a fallos. Sin embargo, puede ser muy complicado construir las tablas en sistemas de tamaño medio o grande.

7.2 Sistemas basados en prioridades

El principio fundamental de este enfoque es planificar todos los recursos empleando prioridades estáticas. Para ello, es necesario disponer de protocolos de comunicación conformes a este principio.

El protocolo de comunicación CAN (*Controller Area Network*) cumple estos requisitos. Es un protocolo de comunicaciones en serie, radiado y sensible a portadora. Está orientado a la transmisión de mensajes de tamaño pequeño en ambientes ruidosos. Los mensajes se identifican mediante una etiqueta o identificador. El protocolo garantiza que siempre se transmite el mensaje más prioritario. El identificador describe el contenido del mensaje y se corresponde con su prioridad.

Dado un protocolo de comunicaciones con este comportamiento, es posible emplear los mismos métodos de análisis que para la planificación del procesador. En general, se tendrán mensajes que se transmiten periódicamente y con una prioridad determinada. La transmisión de mensajes no es expulsiva, por lo que hay una inversión de prioridades igual al tiempo máximo de transmisión de un mensaje. AFDX (ARINC, 2005) y RT-EP (Martínez and González-Harbour, 2005) son dos protocolos de comunicación que también tienen las características para ser usados en estos tipos de sistemas.

Estos sistemas distribuidos se pueden modelar como un conjunto de transacciones (periódicas o esporádicas). Cada una de ellas está compuesta por un conjunto de acciones que se ejecutan en secuencia. Cada acción puede ser una tarea, que se ejecutará en uno de los procesadores, o un mensaje. Cada acción se activa cuando se completa la anterior y activa a la siguiente al terminar su ejecución. El objetivo de un método de análisis es calcular el tiempo de respuesta de acciones y, sobre todo, de la transacción completa.

El primer método de análisis fue llamado holístico (Tindell and Clark, 1994). Trabajos posteriores presentaron métodos de análisis más precisos (Gutiérrez, 1995) (Palencia and González-Harbour, 1999), adaptados a modelos de transacciones más complejos (Gutiérrez *et al.*, 2000) y usando planificación dinámica (Palencia and González-Harbour, 2005).

8. CONCLUSIONES

En este artículo se ha realizado una revisión de los aspectos más importante que tienen que ver con el diseño, desarrollo e implantación de los sistemas de tiempo real. En este momento se puede constatar que el diseño y desarrollo de este tipo de sistemas tiene unos fundamentos claramente asentados e integra aspectos que van desde la ingenieria del software a los sistemas operativos. Uno de los retos que tiene la comunidad de los sistemas de tiempo real, en la que los grupos españoles son muy activos, es contribuir a que estos sistemas permitan integrar los nuevos desafios que las aplicaciones de tiempo real requerirán en los nuevos tiempos. Aspectos como seguridad (en sus distintas acepciones de fiabilidad, robustez, inmunidad a accesos no deseados), gestión de recursos con flexibilidad, soporte para multiples sistemas operativos especializados sobre un mismo procesador, flexibilidad para adaptarse a los entornos, etc., son algunos de estos retos que están en este momento en desarrollo.

REFERENCIAS

Abeni, L. and G. Buttazzo (1998). Integrating multimedia applications in hard real-time systems. In: *IEEE Real-Time Systems Symposium*. pp. 4–13.

Ada (1983). *Military Standard Ada Programming Language*. ANSI/MIL-STD-1815A.

Ada (1995). *Ada Reference Manual. ISO/IEC/ANSI 8652:1995*.

Albertos, P., A. Crespo, I. Ripoll, M. Vallés and P. Balbastre (2000). RT control scheduling to reduce control performance degrading. In: *Proc. 39th IEEE Conference on Decision and Control*. Sydney, Australia.

Albertos, P. and M. Olivares (1999). Time delay limitations in control implementation. In: *European Control Conference*. Karslrue. Germany.

Aldea, M. and M. González-Harbour (2001). MaRTE OS: An Ada kernel for real-time embedded applications. In: *Reliable Software Technologies: Ada Europe 2001*. pp. 305–316. http://marte.unican.es/.

ARINC (2005). *Avionics full duplex switched ethernet (AFDX) network*. Aeronautical radio, inc.

ARTIST2 (2005*a*). Roadmap on control of real-time computing systems.Control Cluster Deliverable. IST FP6 Artist2 NoE.

ARTIST2 (2005*b*). Roadmap on real-time control techniques. Control Cluster Deliverable. IST FP6 Artist2 NoE.

Audsley, N. (1990). Deadline monotonic scheduling.

Audsley, N., A. Burns, M. Richardson, K. Tindell and A. Wellings (1993*a*). Applying new scheduling theory to static preemptive scheduling. *Software Engineering Journal*. **8**(5), 285–292.

Audsley, N.C., A. Burns, M. Richardson, K. Tindell and A. Wellings (1993*b*). Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*. **8**(25), 284292.

Baker, T. P. (1990). A stack-based resource allocation policy for realtime processes. In: *IEEE Real-Time Systems Symposium*. pp. 191–200.

Baker, T. P. (1991). Stack-based scheduling of real-time processes. *Real-Time Systems*. **3**(1), 67–100.

Baker, T.P. and A.C. Shaw (1989). The cyclic executive model and Ada. *Real-Time Systems*. **1**(1), 7–25.

Baruah, S.K., A.K. Mok and L.E. Rosier (1990). Preemptively scheduling hard-real-time sporadic tasks on one processor. In: *IEEE Real-Time Systems Symposium*. pp. 182–190.

Baruah, S.K., D. Chen and A.K. Mok (1997). Jitter concerns in periodic task systems. In: *Proc. 18th Real-Time Systems Symposium*.

Bollella, Greg, Ben Brosgol, Peter Dibble, Steve Furr, James Gosling, David Hardin, Mark Turnbull and Rudy Belliardi (2000). *The Real-Time Specification for Java*. Addisson-Wesley.

Bouyssounouse, B. and Sifakis, J., Eds. (2005). *Embedded Systems Design: The ARTIST Roadmap for Reasearch and Development*. number 3436 In: *LNCS*. Springer-Verlag.

Burns, Alan and Andy Wellings (2001). *Real-Time Systems and Programming Languages (Third Edition)*. Addison Wesley Longmain.

Cervin, Anton (1999). Improved scheduling of control tasks. In: *Proceedings of the 11th Euromicro Conference on Real-Time Systems*. York, UK. pp. 4–10.

Chen, M. and K. Lin (1990). Dynamic priority ceilings: A concurrency control protocol for real-time. *Real-Time Systems*. **2**(4), 325–346.

Crespo, A., I. Ripoll and P. Albertos (1999). Reducing delays in RT control: The control action interval. In: *Proc. 14th IFAC World Congress*. pp. 257–262.

Dertouzos, M.L. and A.K. Mok (1989). Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Trans. Software Engineering*. **15**(12), 1497–1506.

Eker, J., P. Hagander and K.E. Årzén (2000). A feedback scheduler for real-time control tasks. *Control Engineering Practice*. **8**(12), 1369–1378.

Ghazalie, T. M. and T.P. Baker (1995). Aperiodic servers in a deadline scheduling environment. *Real-Time Systems*. **9**(1), 31–67.

Gill, Christopher, Jeanna Gossett, David Corman, Joseph Loyall, Richard Schantz, Michael Atighetchi and Douglas Schmidt (2004). Integrated adaptive qos management in middleware: A case study. In: *10th IEEE Real-Time and Embedded Technology and Applications Symposium*.

González-Harbour, M., M.H. Klein, R. Obenza, B. Pollak and T. Ralya (1993). *A Practitioner's Handbook for Real-Time Analysis*. Kluwer.

Gutiérrez, J. (1995). Planificación, análisis y optimización de sistemas distribuidos de tiempo real estricto. PhD thesis. Universidad de Cantabria.

Gutiérrez, J., J.C. Palencia and M. González-Harbour (2000). Schedulability analysis of distributed hard real-time systems with multiple-event synchronization.. In: *12th Euromicro Conference on Real-Time Systems*.

IEEE (1997). IEEE standards project p1003.13, 1997. draft standard for information technology -standardized application environment profile- POSIX realtime application support (AEP). draft 8. The Institute of Electrical and Electronics Engineers.

J-Consortium (2000). *Real-Time Core Extensions*. International J-Consortium Specification.

Kopetz, Hermann (1997). *Real-Time Systems*. Kluwer Academic Publishers.

Kopetz, Hermann Kopetz (1998). The time-triggered model of computation. In: *IEEE RTSS* (IEEE Press, Ed.).

Lehoczky, John P., Lui Sha and Y. Ding (1989). The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In: *IEEE Real-Time Systems Symposium*. pp. 166–171.

Liu, C.L. and J.W . Layland (1973). Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of the ACM.* **20**, 40–61.

Liu, J. (2000). *Real-Time Systems*. Prentice-Hall.

Martínez, J. M. and M. González-Harbour (2005). RT-EP: A fixed-priority real time communication protocol over standard ethernet. In: *20th International Conference on Reliable Software Technologies, Ada-Europe*.

Melanson, P. and S. Tafazoli (2003). A selection methodology for the rtos market. In: *Data Systems in Aerospace (DASIA 2003) Conference*. Prague, Czech Republic.

OCERA (2002). OCERA: Open components for embedded real-time applications. 2002. IST 35102 European Research Project. (http://www.ocera.org).

Otero-Pérez, C.M., L. Steffens, P. van der Stok, S. van Loo, A. Alonso, J.E. Ruíz, R.J. Bril and M. García Valls (2003). *Ambient Intelligence: Impact on Embedded System Design*. Chap. QoS-based Resource Management for Ambient Intelligence, pp. 159–182. Kluwer Academic Publishers.

Palencia, J.C. and M. González-Harbour (1999). Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In: *20th Real-Time Systems Symposium* (IEEE Computer Press, Ed.).

Palencia, J.C. and M. González-Harbour (2005). Response time analysis of edf distributed real-time systems. *Journal of Embedded Computing*.

Puente, J.A. de la, J.F. Ruíz and J. Zamorano (2000). An Open Ravenscar real-time kernel for GNAT.. In: *Reliable Software Technologies - Ada-Europe 2000*. pp. 5–15. http://www.dit.upm.es/str/proyectos/ork/.

Ripoll, I., A. Crespo and A.K. Mok (1996). Improvement in feasibility testing for real-time tasks. *Real-Time Systems* **11**(1), 19–39.

Ripoll, I., P. Pisa, L. Abeni, P. Gai, A. Lanusse and S. Saez (2002). RTOS state of the art analysis. http://www.ocera.org.

RTSJ (2002). Real-time specification for java. http://www.rtsj.org/.

Ryu, M., S. Hong and M. Saksena (1997). Streamlining real-time controller design: From performance specifications to end-to-end timing constraints. In: *Proc. 3rd IEEE Real-Time Technology and Applications Symposium*. pp. 91–99.

Seto, D., J. P. Lehoczky and L. Sha (1998). Task period selection and schedulability in real-time systems. In: *Proc. 19th IEEE Real-Time Systems Symposium*.

Seto, D., J. P. Lehoczky, L. Sha and K. G. Shin (1996). On task schedulability in real-time control systems. In: *Proc. 17th IEEE Real-Time Systems Symposium*.

Sha, L., R. Rajkumar and J.P. Lehoczky (1990). Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. on Computers.* **39**(9), 1175–1185.

Sha, L., T.F. Abdelzaher, K.E. Arzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J.P. Lehoczky and A.K. Mok (2004). Real time scheduling theory: A historical perspective. *Real-Time Systems.* **28**(2-3), 101–155.

Sprunt, B., L. Sha and J.P. Lehoczky (1989). Aperiodic task scheduling for hard real-time systems. *Real-Time Systems.* **1**(1), 27–60.

Spuri, M. and G. Buttazzo (1994). Efficient aperiodic service under earliest deadline scheduling. In: *IEEE Real-Time Systems Symposium*. pp. 2–11.

Timmerman, M. (2000). RTOS market survey preliminary results. *Dedicated Systems Magazine - Special Edition.*

Tindell, K. and J. Clark (1994). Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming.*

Valls, M. García, A. Alonso, J.F. Ruíz and A. Groba (2002). An architecture for a quality of service resource manager middelware for flexible multimedia embedded systems. In: *the International Workshop on Software Engineering and Middleware*.

Welch, L. R., B. A. Shirazi, B. Ravindran and C. Bruggeman (1998). Desiderata: Qos management technology for dynamic, scalable, dependable real-time systems. In: *IFACs 15th Symposium on Distributed Computer Control Systems*.

Wellings, Andy (2004). *Concurrent and Real-Time Programming in Java*. Wiley.

Zhang, Ronghua, Chenyang Lu, Tarek F. Abdelzaher and John A. Stankovic (2002). Controlware: a middleware architecture for feedback control of software performance. In: *International Conference on Distributed Computing Systems*.