UNIVERSIDAD POLITECNICA DE MADRID
FACULTAD DE INFORMATICA

EXTENSIONES A LOS METODOS DE PLANIFICACION DE SISTEMAS DE TIEMPO REAL CRITICOS BASADOS EN PRIORIDADES

TESIS DOCTORAL

ALEJANDRO ALONSO MUÑOZ

Licenciado en Informática

Febrero de 1994

DEPARTAMENTO DE ARQUITECTURA Y
TECNOLOGIA
DE SISTEMAS INFORMATICOS

FACULTAD DE INFORMATICA

Extensiones a los métodos de planificación de sistemas de tiempo real críticos basados en prioridades

Tesis Doctoral

Autor: Alejandro Alonso Muñoz
Licenciado en Informática
Director: Juan Antonio de la Puente Alfaro
Doctor Ingeniero Industrial

Febrero de 1994

Título:
Extensiones a los métodos de planificación
de sistemas de tiempo real críticos
basados en prioridades

Autor:
Alejandro Alonso Muñoz

Tribunal:

Presidente	:	Pedro de Miguel Anasagasti
Secretario	:	Eugenio Andrés Puente
Vocales	:	Pedro Albertos Pérez
		Michael González Harbour
		Jose Luis Villarroel Salcedo
Suplente	:	María Isabel García Clemente
		Fernando Pérez Costoya

Acuerdan otorgar la calificación de:
Apto con laude por unanimidad

Madrid, 22 de Marzo de 1994

A Ana,
a Alejandro,
a mis padres.

Agradecimientos.

Juan Antonio de la Puente es el principal responsable de que esta tesis se haya realizado. Su constante apoyo y consejo han supuesto una ayuda y un estímulo esenciales en el desarrollo de este trabajo. Los años transcurridos han sido para mí muy satisfactorios y fructíferos en todos los sentidos. Por todo ello, gracias.

A Pedro de Miguel y Antonio Pérez por la confianza depositada en este trabajo y el apoyo prestado para la presentación de esta tesis doctoral.

Julio Gutiérrez me dió la primera oportunidad de entrar en la universiad y entender lo que ésto significa.

Muchos compañeros han contribuido de muy diferente forma en la realización de esta tesis y, en general, a que mi estancia como profesor en la universidad haya sido muy satisfactoria, tanto desde el punto de vista personal como científico. Aún sabiendo que la lista es incompleta y que olvidaré algunos nombres importantes, me gustaría citar aquí a Angel Alvarez, Jesús Carretero, Luis Gómez, Fernando Pérez, Juan L. Redondo, Rafael Rodríguez, Santiago Rodríguez, José I. Tortosa y Juan Zamorano. A todos vosotros muchas gracias.

A Carlos, José y a Luis, por que sin ellos todo ésto valdría menos la pena.

Finalmente, quiero recordar que gran parte de esta tesis se ha realizado en el marco de los proyectos *Biblioteca de Componentes Ada* y *Planificación de procesos en sistemas informáticos con restricciones de tiempo real críticas*. Quiero agradecer a todos aquellos que han hecho posible que esto haya sucedido.

Resumen de la tesis doctoral.

Los sistemas de tiempo real tienen un papel cada vez más importante en nuestra sociedad. Constituyen un componente fundamental de los sistemas de control, que a su vez forman parte de diversos sistemas de ingeniería básicos en actividades industriales, militares, de comunicaciones, espaciales y médicas.

La planificación de recursos es un problema fundamental en la realización de sistemas de tiempo real. Su objetivo es asignar los recursos disponibles a las tareas de forma que éstas cumplan sus restricciones temporales. Durante bastante tiempo, el estado de la técnica en relación con los métodos de planificación ha sido rudimentario. En la actualidad, los métodos de planificación basados en prioridades han alcanzado un nivel de madurez suficiente para su aplicación en entornos industriales. Sin embargo, hay cuestiones abiertas que pueden dificultar su utilización.

El objetivo principal de esta tesis es estudiar los métodos de planificación basados en prioridades, detectar las cuestiones abiertas y desarrollar protocolos, directrices y esquemas de realización práctica que faciliten su empleo en sistemas industriales.

Una cuestión abierta es la carencia de esquemas de realización de algunos protocolos con núcleos normalizados. El resultado ha sido el desarrollo de esquemas de realización de tareas periódicas y esporádicas de tiempo real, con detección de fallos de temporización, comunicación entre tareas, cambio de modo de ejecución del sistema y tratamiento de fallos mediante grupos de recuperación. Los esquemas se han codificado en Ada 9X y se proporcionan directrices para analizar la planificabilidad de un sistema desarrollado con esta base. Un resultado adicional ha sido la identificación de la funcionalidad mínima necesaria para desarrollar sistemas de tiempo real con las características enumeradas.

La capacidad de adaptación a los cambios del entorno es una característica deseable de los sistemas de tiempo real. Si estos cambios no estaban previstos en la fase de diseño o si hay módulos erróneos, es necesario modificar o incluir algunas tareas. La actualización del sistema se suele realizar estáticamente y su instalación se lleva a cabo después de parar su ejecución. Sin embargo, hay sistemas cuyo funcionamiento no se puede detener sin producir daños materiales o económicos.

Una alternativa es diseñar el sistema como un conjunto de unidades que se pueden reemplazar, sin interferir con la ejecución de otras unidades. Para tal fin, se ha desarrollado un protocolo de reemplazamiento dinámico para sistemas de tiempo real crítico y se ha comprobado su compatibilidad con los métodos de planificación basados en prioridades. Finalmente se ha desarrollado un esquema de realización práctica del protocolo.

Abstract.

Real-time systems are very important now a days. They have become a relevant issue in the design of control systems, which are a basic component of several engineering systems in industrial, telecommunications, military, spatial and medical applications.

Resource scheduling is a central issue in the development of real-time systems. Its purpose is to assign the available resources to the tasks, in such a way that their deadlines are met. Historically, hand-crafted techniques were used to develop real-time systems. Recently, the priority-based scheduling methods have reached a sufficient maturity level to be feasible its extensive use in industrial applications. However, there are some open questions that may decrease its potential usefulness.

The main goal of this thesis is to study the priority-based scheduling methods, to identify the remaining open questions and to develop protocols, implementation templates and guidelines that will make more feasible its use in industrial applications.

One open question is the lack of implementation schemes, based on commercial real-time kernels, of some of the protocols. POSIX and Ada 9X has served to identify the services usually available. A set of implementation templates for periodic and sporadic tasks have been developed with provision for timing failure detection, intertask communication, change of the execution mode and failure handling based on recovery groups. Those templates have been coded in Ada 9X. A set of guidelines for checking the schedulability of a system based on them are also provided. An additional result of this work is the identification of the minimal functionality required to develop real-time systems based on priority scheduling methods, with the above characteristics.

A desirable feature of real-time systems is their capacity to adapt to changes in the environment, that cannot be entirely predicted during the design, or to misbehaving software modules. The traditional maintenance techniques are performed by stopping the whole system, installing the new application and finally resuming the system execution. However this approach cannot be applied to non-stop systems.

An alternative is to design the system as a set of software units that can be dynamically replaced within its operative environment. With this goal in mind, a dynamic replacement protocol for hard real-time systems has been defined. Its compatibility with priority-based scheduling methods has been proved. Finally, a execution template of the protocol has been implemented.

Índice General

Índice de figuras. **xvii**

Abreviaturas. **xix**

1 Planteamiento y objetivos de la tesis. 1
- 1.1 Ámbito de la tesis. 1
- 1.2 Objetivos de la tesis. 2
- 1.3 Contenido de la tesis. 3

2 Sistemas de tiempo real. 5
- 2.1 Naturaleza de los sistemas de tiempo real. 5
  - 2.1.1 Sistemas de control por computador. 5
  - 2.1.2 Otros sistemas de tiempo real. 7
  - 2.1.3 Necesidad de los sistemas de tiempo real. 7
- 2.2 Sistemas de tiempo real. 9
  - 2.2.1 Definición de sistemas de tiempo real. 9

3 Planificación de sistemas de tiempo real 11
- 3.1 Los sistemas de tiempo real como sistemas concurrentes 11
  - 3.1.1 Modelo de sistema. 11
  - 3.1.2 Tareas de tiempo real. 13
  - 3.1.3 Planificación de sistemas de tiempo real. 15
- 3.2 Planificación síncrona. 16
  - 3.2.1 Descripción del ejecutivo cíclico. 16
  - 3.2.2 Evaluación 18
- 3.3 Planificación basada en prioridades. 19
  - 3.3.1 Método de prioridad a la tarea más frecuente. 19
  - 3.3.2 Evaluación. 21
- 3.4 Planificación basada en plazos. 22
  - 3.4.1 Método basado en la proximidad del plazo de respuesta. 22
  - 3.4.2 Evaluación 24
- 3.5 Conclusiones 24

4 Planificación basada en prioridades. 25
- 4.1 Prioridad a la tarea más frecuente. 25

4.1.1 Análisis de planificabilidad. 25
4.1.2 Sobrecarga transitoria del sistema 27
4.2 Prioridad a la tarea más urgente. 28
4.2.1 Planteamiento. 28
4.2.2 Enunciado del método 29
4.2.3 Análisis de planificabilidad 29
4.2.4 Sobrecarga transitoria del sistema. 31
4.3 Comunicación entre tareas. 32
4.3.1 Planteamiento. 32
4.3.2 Herencia de prioridades. 33
4.3.3 Protocolo del techo de prioridad. 35
4.3.4 Protocolo de herencia inmediata del techo de prioridad. 37
4.3.5 Protocolo de control de semáforos. 38
4.3.6 Análisis de la planificabilidad. 38
4.3.7 Conclusiones. 41
4.4 Tareas aperiódicas. 41
4.4.1 Planteamiento. 41
4.4.2 Protocolo del servidor esporádico. 41
4.4.3 Análisis de la planificabilidad 44
4.5 Cambio de modo de ejecución del sistema. 44
4.5.1 Planteamiento. 44
4.5.2 Protocolo básico de cambio de modo 45
4.5.3 Determinación precisa de los tiempos de respuesta. 46
4.5.4 Comunicación entre tareas. 47
4.5.5 Conclusión 47
4.6 Otros estudios. 47
4.7 Conclusiones. 48

5 Sistemas de tiempo real distribuidos. 49
5.1 Introducción. 49
5.1.1 Sistemas distribuidos de control. 49
5.1.2 Sistemas distribuidos de tiempo real. 50
5.2 Sistemas síncronos. 52
5.2.1 Planteamiento. 52
5.2.2 *MARS*. 52
5.2.3 Comentarios 55
5.3 Sistemas dinámicos. 55
5.3.1 Planteamiento. 55
5.3.2 *Spring* 56
5.3.3 Comentarios 58
5.4 Sistemas distribuidos basados en prioridades. 59
5.4.1 Planteamiento. 59
5.4.2 Generalización de la planificación basada en prioridades. 59
5.4.3 Sistema desarrollado en la Universidad de York. 62

5.4.4 Comentarios 64
5.5 Consideraciones finales. 65

6 Desarrollo de sistemas de tiempo real basados en prioridades. 67
6.1 Planteamiento. 67
6.1.1 Requisitos de los sistemas de tiempo real. 69
6.1.2 Entornos de ejecución. 72
6.1.3 Directrices de codificación. 76
6.1.4 Características generales de los sistemas. 79
6.2 Tareas de tiempo real crítico. 80
6.2.1 Funcionalidad básica de las tareas. 80
6.2.2 Comunicación entre tareas. 88
6.2.3 Detección de fallos de ejecución. 91
6.3 Cambio de modo. 103
6.3.1 Planteamiento. 103
6.3.2 Cambio de modo síncrono. Consideraciones de diseño. 104
6.3.3 Algoritmo de cambio de modo síncrono. 109
6.3.4 Esquema de realización del algoritmo. 110
6.4 Tratamiento de fallos. 127
6.4.1 Planteamiento. 127
6.4.2 Grupo de recuperación. 128
6.4.3 Consideraciones de diseño. 129
6.4.4 Esquema de realización. 130
6.5 Análisis de la planificabilidad 140
6.5.1 Cálculo del tiempo de cómputo de una tarea. 140
6.5.2 Cálculo del tiempo de bloqueo de una tarea. 141
6.5.3 Activación autónoma de la tarea supervisora. 144
6.6 Conclusiones 144

7 Reemplazamiento dinámico de software en sistemas de tiempo real críticos. 147
7.1 Planteamiento del problema. 147
7.2 Conceptos básicos 149
7.2.1 Requisitos del reemplazamiento dinámico. 149
7.2.2 Unidad de reemplazamiento 150
7.2.3 Apoyo del sistema operativo 152
7.3 Protocolo de reemplazamiento dinámico de software 153
7.4 Protocolo de reemplazamiento dinámico de software en sistemas de tiempo real críticos. 155
7.4.1 Modelo del sistema. 155
7.4.2 Crítica al protocolo de reemplazamiento general. 155
7.4.3 Protocolo de reemplazamiento dinámico. 156
7.5 Realización práctica del protocolo. 163
7.5.1 Comunicación entre unidades. 165
7.5.2 Unidad supervisora del reemplazamiento. 168

7.5.3 Unidad reemplazable de software. 177
7.6 Ampliaciones. 181
7.7 Conclusiones. 182

8 Aplicación: Biblioteca de Componentes Ada 183
8.1 Planteamiento 184
8.1.1 Reutilización de software 184
8.1.2 Planteamiento general del proyecto 185
8.1.3 Ejecutivo multitarea 185
8.1.4 Inconvenientes de Ada. 186
8.2 Descripción general del ejecutivo multitarea 187
8.2.1 Descripción funcional. 187
8.2.2 Componentes reusables 189
8.2.3 Estructura de una aplicación 191
8.3 Diseño del ejecutivo multitarea 191
8.3.1 Proceso supervisor del sistema 191
8.3.2 Procesos de tiempo real 193
8.3.3 Tratamiento de errores 198
8.3.4 Cambio de modo 201
8.3.5 Comunicación entre procesos 204
8.3.6 Análisis de la planificabilidad 205
8.4 Desarrollo de sistemas de tiempo real 205
8.4.1 Directrices metodológicas 205
8.4.2 Especificación de la aplicación ejemplo 207
8.4.3 Asignación de prioridades 209
8.4.4 Análisis de planificabilidad 211
8.4.5 Creación de la aplicación 214
8.5 Consideraciones finales. 214

9 Conclusiones y líneas de trabajo futuro. 217
9.1 Conclusiones. 217
9.2 Líneas de trabajo futuro. 221

A Análisis detallado del protocolo de cambio de modo. 225

B Desarrollo de sistemas con el ejecutivo multitarea 233
B.1 Modos 233
B.2 Perfiles de ejecución 233
B.3 Supervisor del sistema 234
B.4 Procesos de recuperación 235
B.5 Procesos Periódicos 239
B.6 Procesos Esporádicos 241
B.7 Monitores 245
B.8 Procesos de segundo plano 247
B.9 Manejadores de interrupciones 248

B.10 Procedimiento Principal 249

Bibliografía **251**

Índice de Figuras

3.1 Estructura de un sistema de tiempo real. 12
3.2 Estados de una tarea. 13
3.3 Estructura de una planificación cíclica. 16

4.1 Inversión de prioridad. 33
4.2 Protocolo de herencia de prioridades. 34
4.3 Interbloqueo al aplicar el protocolo de herencia de prioridades. 34
4.4 Interbloqueo al aplicar el protocolo de herencia de prioridades. 35
4.5 Protocolo del techo de prioridad. Prevención de interbloqueos 36
4.6 Protocolo del techo de prioridad. Prevención de bloqueos encadenados . 37
4.7 Ejecución de un servidor esporádico. 43

6.1 Fallo de temporización. 93
6.2 Caso más desfavorable de detección de cambio de modo. 107

7.1 Fase de carga del protocolo de reemplazamiento en sistemas de tiempo real. 158
7.2 Fase de inicio del protocolo de reemplazamiento en sistemas de tiempo real. 159
7.3 Fase de intercambio del protocolo de reemplazamiento en sistemas de tiempo real. 161
7.4 Fase de limpieza del protocolo de reemplazamiento en sistemas de tiempo real. 162
7.5 Esquema de las comunicaciones entre unidades durante un operación de reemplazamiento. 166

8.1 Esquema de un sistema creado con el ejecutivo multitarea 192

A.1 τ_i y τ_j comparten un instante crítico. 227
A.2 τ_j se activa inmediatamente después que el cambio de modo 228

Abreviaturas.

A continuación se enumeran las abreviaturas utilizados en este documento. Si en algún capítulo se cambiara el significado de alguna de ellas, se comentará convenientemente.

BCA: Proyecto *Biblioteca de Componentes Ada.*
B_i: Tiempo de bloqueo que sufre la tarea τ_i.
C_i: Tiempo de cómputo máximo de la tarea τ_i.
CP: Ciclo principal de un planificador síncrono.
CS: Ciclo secundario de un planificador síncrono.
c_m: Prioridad techo del monitor M.
D_i: Plazo de respuesta de la tarea τ_i.
DMS: *Deadline Monotonic Scheduling.*
d_i: Retardo de transmisión de un mensaje.
E_p: Energía en un punto en el algoritmo de distribución de carga (Apartado 5.4.3).
F: Función de enfriado en el algoritmo de distribución de carga (Apartado 5.4.3).
H_i: Tiempo durante el que el nodo i puede enviar sus mensajes.
I_i: Interferencia de la tarea τ_i.
LM_i: Longitud de un mensaje.
M_i: Un monitor de comunicación el sistema.
ME_i: Un mensaje.
MI_I: Un manejador de interrupciones.
np_i: Un nivel de prioridad del sistema.
P_i: Punto de energí en el algoritmo de distribución de carga (Apartado 5.4.3).
PE_i: Proceso esporádico en el capítulo 8.
PP_i: Proceso periódico en el capítulo 8.
PR_i: Proceso de recuperación en el capítulo 8.
PSP_i: Proceso de segundo plano en el capítulo 8.
PTR_i: Proceso de tiempo real en el capítulo 8.
pr_i: Prioridad de la tarea τ_i.

R_i:	Tiempo de respuesta de la tarea τ_i.
RR:	Capacidad de red consumida por la rotación del testigo.
RT_i:	Instante de tiempo en el que se restaura el tiempo de cómputo consumido por un servidor esporádico que ejecuta en el nivel de prioridad np_i.
RMS:	*Rate Monotonic Scheduling.*
S_i:	Separación entre eventos de la tarea acíclica τ_i.
Su:	Supervisor de reemplazamiento.
SE_i:	Servidor esporádico.
t_i:	Valor de tiempo.
T_i:	Período de la tarea periódica τ_i.
Te:	Factor de temperatura en el algoritmo de distribución de carga (Apartado 5.4.3).
$TDMA$:	*Time Division Multiple Access.*
$TTRT$:	Tiempo de rotación del testigo.
τ_i:	Una tarea del sistema.
Φ_i:	Fase de la tarea τ_i.

Capítulo 1

Planteamiento y objetivos de la tesis.

1.1 Ámbito de la tesis.

Los sistemas de tiempo real tienen un papel cada vez más importante en nuestra sociedad. Constituyen un componente fundamental de los sistemas de control, que a su vez forman parte de diversos sistemas de ingeniería básicos en actividades industriales, militares, de comunicaciones, espaciales y médicas. Por todo ello, es preciso producir sistemas de tiempo real más eficientes, seguros, complejos y económicos que los actuales.

La característica diferenciadora de los sistemas de tiempo real es la obligación de completar sus actividades en plazos de tiempo determinados. El incumplimiento de estos requisitos supone el funcionamiento incorrecto del sistema. Esta propiedad hace que su desarrollo sea más complejo que el de los sistemas informáticos sin restricciones temporales. En consecuencia, sus técnicas de especificación, programación, prueba y, en general, de desarrollo son, en muchos casos, rudimentarias y más propias de una actividad artesanal que ingenieril.

Este panorama se agrava en el caso de los sistemas de tiempo real críticos. En estos sistemas, el incumplimiento de uno de los plazos de respuesta es intolerable, pues se pueden producir daños materiales y/o económicos graves. Este es el caso, por ejemplo, de los sistemas de control de vuelo o los sistemas de control de robots. El nivel de fiabilidad que se exige a estos sistemas es mucho mayor, con las repercusiones que ello implica en su desarrollo.

Los trabajos realizados ultimamente en relación con los métodos de planificación basados en prioridades, han permitido disponer en la actualidad de unos sólidos fundamentos analíticos, que están cambiando el panorama descrito. De hecho, la posibilidad de determinar analíticamente la planificabilidad de un sistema, permite, entre otras ventajas, aumentar el grado de abstracción de su diseño.

El enunciado básico del método de planificación basado en prioridades y la demostración de sus propiedades más interesantes datan del año 1973 [Liu&73]. Sin embargo, en la formulación del método se establecen un conjunto de restricciones funcionales a las tareas que han impedido su aplicación industrial en gran escala. En los últimos años,

se ha dedicado mucho esfuerzo en mejorar este método, lo que ha supuesto importantes avances, como el desarrollo de métodos de planificación que permiten especificar plazos de respuesta más flexibles [Leung&82] [Audsley&91], procedimientos de análisis de planificabilidad más precisos [Lehoczky&89] [Audsley&91], protocolos para la comunicación entre tareas [Rajkumar91], tratamiento de eventos esporádicos [Sprunt&88], cambio dinámico del modo de ejecución [Sha&89c] [Tindell&92], etc.

Gracias a este esfuerzo se ha producido una maduración de estos métodos hasta el punto de que es factible su utilización generalizada en sistemas industriales. Así, se están empleando en el desarrollo de sistemas de tiempo real en la NASA, la ESA, y son los métodos recomendado por *IBM Federal Sector Division* [Sha&91a]. Además, estos resultados han influido notoriamente en el desarrollo de normas internacionales como POSIX [POSIX.1b 93] [POSIX.1c 93], Ada9X [Ada9XRM93] y Futurebus 1+ [Futurebus91].

Los sistemas distribuidos de tiempo real serán necesarios para abordar la complejidad y para satisfacer los requisitos de los sistemas de tiempo real de la siguiente generación. Los métodos de planificación basados en prioridades se pueden emplear en este contexto, como lo muestran, entre otros, los trabajos llevados a cabo en el *Software Engineering Institute* de la Universidad de Carnegie-Mellon [Sha&93] y en la Universidad de York [Tindell&92a].

Sin embargo, a pesar de los considerables avances que se han producido, aún quedan cuestiones pendientes, como diseñar mecanismos de tolerancia o tratamiento de fallos, establecer directrices de realización práctica de algunas extensiones al método original, incorporar al análisis de planificación detalles de bajo nivel, que en la formulación del método no se consideran, diseñar y realizar herramientas específicas y metodologías de diseño y desarrollo que guíen su utilización, etc.

En conclusión, los métodos de planificación basados en prioridades han alcanzado un grado de madurez suficiente para su aplicación industrial. Sin embargo, hay algunas cuestiones pendientes de resolución que dificultan la obtención del máximo beneficio posible y que, en determinados casos, pueden desaconsejar su utilización.

1.2 Objetivos de la tesis.

El objetivo principal de esta tesis es estudiar en profundidad los métodos de planificación basados en prioridades, detectar las cuestiones abiertas existentes y desarrollar protocolos, directrices y esquemas de realización que faciliten su utilización en sistemas industriales de tiempo real crítico. La funcionalidad básica que se debe proporcionar para satisfacer los requisitos generales de este tipo de aplicaciones es:

- Coexistencia de tareas periódicas, esporádicas y de segundo plano (*background*).
- Comunicación entre tareas.
- Comprobación analítica de la planificabilidad del sistema.
- Detección de fallos de temporización.

- Tratamiento de fallos de ejecución.
- Capacidad de adaptación a cambios internos o del entorno.

Una principio de diseño en este trabajo es el empleo de núcleos de ejecución normalizados, para que los componentes desarrollados se puedan adaptar diréctamente a otros entornos de desarrollo. De hecho, un inconveniente de algunos esquemas de realización publicados es que se basan en núcleos con características especiales y orientadas a un protocolo concreto y que, por esta razón, no están disponibles en productos comerciales.

La capacidad de adaptación a los cambios se basa en dos características principales:

- Cambio dinámico del modo de ejecución de las tareas.
- Reemplazamiento dinámico de software.

Mientras que han aparecido métodos que permiten realizar sistemas con varios modos de funcionamiento, no ha ocurrido lo mismo en relación a la segunda característica. Por lo tanto, un objetivo adicional de esta tesis ha sido desarrollar un protocolo de reemplazamiento dinámico de software para sistemas de tiempo real crítico y comprobar su compatibilidad con el método de planificación basado en prioridades.

1.3 Contenido de la tesis.

La presente tesis está dividida en nueve capítulos y tres apéndices, cuyo contenido se describe a continuación:

- Capítulo 1: *Planteamiento y objetivos de la tesis.* El presente capítulo, en el que se expone la motivación y el origen de la tesis y se presenta la estructura en capítulos de este documento.
- Capítulo 2: *Sistemas de tiempo real.* El objetivo de este capítulo es introducir los sistemas de tiempo real. En primer lugar se justifica la necesidad de los sistemas de tiempo real, mediante la identificación de sistemas cuyo correcto funcionamiento obliga a satisfacer ciertos requisitos temporales. Posteriormente, se define rigurosamente lo que se entiende por sistemas de tiempo real.
- Capítulo 3: *Planificación de sistemas de tiempo real.* Con objeto de acotar el marco de este trabajo, se define un modelo básico de sistemas de tiempo real. A continuación se describe el problema de la planificación de sistemas de tiempo real y se presentan los tres métodos generales de planificación en sistemas monoprocesadores.
- Capítulo 4: *Planificación basada en prioridades.* Este método de planificación y sus extensiones principales son descritas en profundidad.

- Capítulo 5: *Sistemas de tiempo real distribuidos.* La distribución será una característica imprescindible en los sistemas de tiempo real de la siguiente generación. Como reflejo de esta tendencia se muestran tres enfoques diferentes y alternativos al desarrollo de este tipo de sistemas.

- Capítulo 6: *Desarrollo de sistemas de tiempo real basados en prioridades.* Este capítulo y los dos siguientes contienen la mayoría de las aportaciones originales de la tesis. El objetivo de este apartado es tratar de resolver algunas cuestiones prácticas relacionadas con la aplicación industrial de los sistemas de tiempo real basados en prioridades. Para tal fin, se han desarrollado una serie de esquemas de tareas de tiempo real que satisfacen los requisitos básicos del método de planificación e incorporan mecanismos para detección y tratamiento de fallos y cambio de modo de ejecución. En paralelo, se han analizado los requisitos funcionales mínimos que debe proporcionar un núcleo de ejecución para su realización práctica.

- Capítulo 7: *Reemplazamiento dinámico de software en sistemas de tiempo real crítico.* Esta funcionalidad resuelve ciertos problemas que se presentan cuando hay que cambiar módulos software sistemas cuya detención no es conveniente. En este apartado se presenta un protocolo de reemplazamiento dinámico de software y se comprueba su compatibilidad con los métodos de planificación basados en prioridades. Finalmente, se desarrolla un esquema de realización práctica del protocolo.

- Capítulo 8: *Aplicación: Biblioteca de Componentes Ada.* El proyecto industrial *Biblioteca de Componentes Ada* constituyó un marco ideal para el desarrollo de un primera versión de los esquemas de tareas presentados anteriormente y para comprobar prácticamente su validez. Por este motivo, se ha considerado útil dedicar este capítulo a describir su planteamiento general y los resultados más interesantes relativos a este trabajo. En concreto, se describe el *ejecutivo multitarea*, que se basa en los mencionados esquemas de tareas.

- Capítulo 9: *Conclusiones y líneas de trabajo futuro.* Es el momento de sintetizar las conclusiones más importantes del trabajo realizado y de exponer las líneas más interesantes de continuación de esta tesis.

- Apéndice A: *Análisis detallado del protocolo de cambio de modo.* En este apéndice se presenta el análisis de planificabilidad detallado del protocolo de cambio de modo descrito en el capítulo 4. Su complejidad y la poca aportación conceptual al protocolo aconsejan su inclusión en este apartado.

- Apéndice B: *Desarrollo de sistemas con el ejecutivo multitarea.* Con objeto de ilustrar la utilización del ejecutivo multitarea para desarrollar aplicaciones de tiempo real, se muestra la secuencia de uso de los componentes reutilizables disponibles.

- *Bibliografía.* Citas bibliográficas referenciadas en la tesis.

Capítulo 2

Sistemas de tiempo real.

2.1 Naturaleza de los sistemas de tiempo real.

2.1.1 Sistemas de control por computador.

Los sistemas de control tienen una importancia vital en la sociedad actual. Suelen estar integrados en sistemas más complejos y su objetivo es gobernar el funcionamiento de éstos, para que su funcionamiento sea uno determinado. Forma una parte vital en sistemas de aviónica, centrales nucleares, robots y son un componente básico en los procesos[1] industriales y de fabricación modernos. Una definición básica de un sistema de control es un sistema que mide un conjunto de variables de salida de un proceso físico y modifica las variables de entrada para corregir o limitar desviaciones de las variables medidas respecto a los valores deseados [Ogata90] [Gayakwad&88].

Tradicionalmente, los sistemas de control se realizaban con tecnología analógica. A finales de los años cincuenta se dieron los primeros casos de empleo de computadores digitales. Inicialmente eran sistemas de supervisión que se empleaban para la determinación de los parámetros óptimos de los controladores analógicos. A partir del año 1962 se comienzan a emplear ordenadores para la realización de funciones de control digital directo. La creciente potencia de los ordenadores motivó que se usaran para realizar simultaneamente labores de control y supervisión de la planta, lo que complicó paulatinamente el desarrollo del software. Esta tendencia se ha mantenido hasta nuestros días, y actualmente la gran flexibilidad y el incremento continuo de la potencia de los sistemas informáticos ha permitido incluir un elevado número de funciones en un sólo computador y motivar el desarrollo de métodos de control muy sofisticados, como son los sistemas de control basados en el conocimiento [Payton&91] [Chandrasekaran&91] [Sanz90] [Velasco91].

Los sistemas de control se pueden agrupar como sigue:

[1] El término *proceso* se utiliza para referirse a la transformación de materia y energía o a la unidad de concurrencia en un sistema informático. En general, es fácil comprender el sentido en que se usa por el contexto en que aparece. En cualquier caso y mientras no se diga lo contrario, se usará el término *proceso* con el primer significado y se utilizará el término *tarea* para referirse al segundo.

- **Control de sistemas continuos.** Los sistemas continuos son sistemas que varían continuamente en el tiempo. En general, estos sistemas se representan en términos de ecuaciones diferenciales lineales o no lineales. En muchos casos las ecuaciones no lineales se pueden linealizar, bajo ciertas suposiciones.

  El diseño y análisis de este tipo de sistemas de control es el objeto de la teoría clásica de control, p.e. [Dorf89] [Ogata90]. Estas técnicas se han diseñado con la hipótesis de que los controladores serían también sistemas continuos. La realización con computadores de estos elementos presenta problemas específicos. El computador recibe las señales de salida del proceso y actúa sobre las entradas en instantes de tiempo discretos. Por tanto se debe describir el cambio de las señales de instante de muestreo en instante de muestreo y sin información sobre comportamiento del sistema entre tanto. Este problema ha sido objeto de intenso trabajo de investigación y se han desarrollado métodos teóricos y prácticos para realizar controladores analógicos por computador y métodos de diseño específicos de controladores de tiempo discreto, p.e. [Ogata87] [Astrom&90].

  Los modelos matemáticos que se emplean suelen suponer que hay una sincronización entre el instante de muestreo y el accionamiento correspondiente al muestreo previo. Si esta condición no se cumple, el funcionamiento de controlador no será el deseado y, por lo tanto, el comportamiento del proceso controlado tampoco lo será. En consecuencia, cuando estos controladores se realizan por computador, se debe asegurar que la respuesta del programa ocurrirá de acuerdo con esta hipótesis.

- **Control de sistemas discretos.** Los sistemas discretos son sistemas que sólo cambian en instantes discretos de tiempo y como respuesta a un evento externo [Silva85] [Olsson&92]. Por esta razón, a estos sistemas también se les conoce como sistemas de eventos discretos. Los métodos control de estos sistemas son radicalmente diferentes a los de sistemas continuos. Se suelen realizar mediante métodos basados en representar de alguna forma el estado del sistema y en especificar los cambios de estado en función de las variaciones en las entradas. Por ejemplo, los grafos de estado y las Redes de Petri son dos métodos que responden a esta descripción. La utilización de sistemas informáticos para realizar estos tipos de sistemas es aún mayor que los anteriores.

  La realización práctica de este tipo de sistemas también requiere una respuesta del sistema en un intervalo de tiempo que comienza cuando se produce el evento que la activa. En caso de no cumplirse estos requisitos, el sistema no funcionará adecuadamente.

Algunos controladores forman parte de sistemas de seguridad crítica. Estos son sistemas en los que un fallo puede tener consecuencias catastróficas en término de daños económicos o incluso en vidas humanas. Un ejemplo son los sistemas de control por computador forman una parte vital de los modernos sistemas de aviónica [Middleton89] [Spitzer87]. Las consecuencias que un fallo podría provocar son fácilmente imaginables.

2.1.2 Otros sistemas de tiempo real.

Los sistemas multimedia son sistemas informáticos que tratan de forma integrada información de diversa naturaleza, como imágenes, sonido y texto. Los avances tecnológicos en informática y redes de telecomunicación ha estimulado la aparición de diversas aplicaciones multimedia distribuidas [Bellcore93], como el trabajo cooperativo asistido por computador [Greenberg90], conferencias multimedia [Schooler91] [Vin&91], aplicaciones médicas [Goldberg&89] y educación a distancia [Smith88].

Algunas de estas aplicaciones se basan en la transmisión y sincronización de imágenes y sonido. El flujo de los datos de voz e imágenes son isócronos por naturaleza y se pueden tratar como una secuencia de muestras de tamaño finito que se generan, transmiten y reciben en intervalos de tiempo fijo, imponiendo una serie de restricciones temporales que se deben cumplir [Nicolau90]. Esta restricción se impone al conjunto de la transmisión, es decir desde que se muestrea la señal hasta que se presenta al usuario.

Así pues, el sistema distribuido sobre el que se desarrollan las aplicaciones multimedia, debe asegurar estos plazos de respuesta. De otra forma, el usuario no recibirá imágenes realistas o el sonido no se corresponderá con la imagen.

2.1.3 Necesidad de los sistemas de tiempo real.

Los tipos de sistemas que se han presentado en los apartados anteriores imponen a los sistemas informáticos dos tipos de requisitos específicos:

- El programa no controla completamente el flujo de ejecución, sino que debe seguir la pauta marcada por los acontecimientos que acontecen en el sistema físico.
- El programa debe realizar estrictamente determinadas operaciones en un intervalo fijo de tiempo.

Las técnicas de diseño y de desarrollo de software de propósito general, no consideran estos requisitos. Un objetivo de estas aplicaciones es conseguir que los resultados sean lógicamente correctos, pero el instante en que se producen no condiciona su validez (en todo caso se trata de conseguir tiempos medios de respuesta aceptables). Por otro lado, las arquitecturas software que se emplean no son generalmente adecuadas para diseñar sistemas que tienen que tratar eventos físicos.

Por estas razones, ha surgido la necesidad de los sistemas de tiempo real para tratar adecuadamente estos requisitos específicos. Su problemática es muy amplia y han dado lugar a diversas áreas de investigación para dar respuesta a las necesidades actuales y para producir sistemas más fiables, potentes y cuya complejidad sea manejable. Las áreas en las que se ha detectado la necesidad de disponer de métodos específicos para sistemas de tiempo real son [Stankovic88a]:

- *Especificación y verificación.* El reto fundamental en la especificación y verificación de sistemas de tiempo real es cómo incorporar el tiempo en la descripción del

sistema. Es necesario desarrollar métodos que permitan describir las restricciones de tiempo del sistema y para analizar si se cumplen.

- *Teoría de planificación.* El objetivo principal de la teoría de planificación es ordenar la ejecución de las actividades de un sistema, de forma que se cumplan los requisitos temporales. Hay que desarrollar métodos que permitan comprender, predecir y facilitar el mantenimiento del comportamiento temporal del sistema.

- *Sistemas operativos.* Deben proporcionar un alto nivel de abstracción a los programadores y, a la vez, cumplir las restricciones temporales, que dependen de detalles de bajo nivel de la implementación y del entorno. El factor tiempo y los principios de planificación deben ser elementos centrales en el diseño de aspectos del sistema operativo como los métodos de gestión de recursos, las tareas de aplicación, etc.

- *Lenguajes de programación.* El aumento de la complejidad de los sistemas de tiempo real demanda abstracciones de alto nivel en los lenguajes de programación. Algunas características deseables son mecanismos para gestión del tiempo, análisis de planificabilidad, módulos reutilizables de tiempo real y mecanismos para desarrollar programas distribuidos y tolerantes a fallos.

- *Metodologías de diseño.* Son necesarias metodologías de diseño que incorporen y permitan razonar sobre el tiempo desde las primeras fases de desarrollo.

- *Bases de datos distribuidas.* Los datos en los sistemas de tiempo real sólo son válidos durante un cierto período de tiempo. Para satisfacer las restricciones de tiempo se debe aumentar el paralelismo de las transacciones y se deben integrar los algoritmos de control de acceso a la base de datos y mecanismos de planificación.

- *Inteligencia artificial.* La utilización de métodos basados en el conocimiento en sistemas de tiempo real pasa por controlar el tiempo de cómputo de su aplicación. Se debe proporcionar la mejor respuesta posible en un plazo de tiempo que puede ser variable.

- *Tolerancia a fallos.* Los sistemas de tiempo real se emplean en sistemas que requieren un nivel alto de fiabilidad. En este sentido, son necesarios métodos generales de detección de errores, localización de fallos, reconfiguración dinámica del sistema y recuperación de fallos.

- *Arquitecturas de computadores.* Las arquitecturas de computadores disponibles no satisfacen los requisitos deseables para su aplicación en sistemas de tiempo real. Las arquitecturas para tiempo real deben proporcionar mecanismos hardware de apoyo al tratamiento de fallos, a los algoritmos de planificación, a los sistemas operativos y a los lenguajes de programación de tiempo real. La topología de interconexión de los procesadores con los dispositivos de entrada/salida y comunicaciones rápidas y fiables, también son cuestiones pendientes de soluciones adecuadas.

- *Comunicación.* Los mecanismos de comunicación son básicos para la siguiente generación de sistemas distribuidos de tiempo real. Las características que se requieren son predicibilidad, fiabilidad y altas prestaciones.

2.2 Sistemas de tiempo real.

2.2.1 Definición de sistemas de tiempo real.

El término **sistemas de tiempo real** se suele utilizar con ambigüedad. Por tanto, la primera cuestión es definir con precisión su significado en el contexto en que se desarrolla este trabajo. En la literatura técnica hay multitud de definiciones. La que proporciona el diccionario Oxford de computadores es:

> Cualquier sistema en el que el tiempo en el que se produce la salida es significativo. Esto se debe habitualmente a que la entrada corresponde a algún movimiento en el mundo físico, y la salida está relacionada con este movimiento. El intervalo entre el instante de la entrada y el de salida debe ser suficientemente pequeño para que sea oportuno.

Un gran número de sistemas responden a esta definición, como es un sistema operativo de propósito general que debe responder a un comando de un usuario. Sin embargo, si la respuesta se retrasa, no ocurre nada grave. Estos sistemas se deben diferenciar de aquellos en los que una respuesta producida tarde es una respuesta errónea. La característica más importante de un sistema de tiempo real es [Stankovic88a][Burns&89]:

> la corrección de un sistema de tiempo real depende tanto de la validez lógica de la respuesta, como del instante de tiempo en que se produce.

Este requisito es la principal causa de la mayor dificultad en el desarrollo de estos sistemas respecto a los sistemas informáticos clásicos.

Los sistemas de tiempo real, suelen especificarse como un conjunto de actividades, parte de las cuales deben completarse en un intervalo de tiempo, denominado plazo de respuesta. Dependiendo de las características de estos plazos, estos sistemas se dividen en:

- *Críticos*: El incumplimiento de un requisito temporal supone un fallo intolerable por sus consecuencias en el sistema. En un sistema de control de la trayectoria del brazo de un robot, cuando se produce un evento que indica que se ha llegado a una posición determinada, el sistema debe detener o cambiar la dirección del movimiento en un plazo de tiempo determinado. Si no es así, el brazo podría impactar sobre algún objeto cercano.

- *Acríticos*: Se puede tolerar el incumplimiento ocasional de algunos requisitos temporales. En algunos sistemas de multimedia las imágenes y el sonido se deben

transmitir dentro de unos plazos determinados. Si no se cumple éste y una trama de sonido llega tarde, no es importante, ya que puede ser descartada sin que el usuario lo perciba.

Una característica básica de los sistemas de tiempo real radica en los mecanismos empleados para iniciar la ejecución sus actividades. Atendiendo a estos mecanismos, los sistemas de tiempo real se dividen en:

- *Dirigidos por tiempo*: El reloj es el único elemento que inicia las actividades del sistema. Las condiciones de inicio están constituidas por instantes precisos de tiempo. Cuando el valor del tiempo del sistema alcanza este valor, las actividades correspondientes comienzan a ejecutarse.
- *Dirigidos por eventos* : Las actividades del sistema se inician como respuesta a eventos externos o en instantes precisos de reloj. Una interrupción de un dispositivo externo o un evento detectado por el sistema, pueden iniciar la ejecución de una actividad.

Los sistemas resultantes a partir de estas concepciones son radicalmente diferentes. En el capítulo siguiente se analizarán algunas de las características diferenciadoras resultantes de estos enfoques.

Los primeros sistemas de tiempo real eran *centralizados*. Estos sistemas se caracterizan por disponer de un sólo procesador que centraliza todos los cálculos y las decisiones. La evolución de los computadores y las necesidades de las aplicaciones de control reclaman otros tipos de sistemas. La respuesta a estas necesidades la proporcionan los *sistemas distribuidos*, compuestos por un conjunto de procesadores conectados entre sí y que colaboran para satisfacer la funcionalidad requerida. Estos sistemas se presentan en el capítulo 5.

Capítulo 3

Planificación de sistemas de tiempo real

El objetivo de este capítulo es presentar las técnicas de planificación de sistemas de tiempo real críticos en sistemas monoprocesador más significativas. En primer lugar se describe el modelo de sistema de tiempo real que se considera. A continuación, se introducen los métodos de planificación síncrona, basada en prioridades y basada en plazos. El estudio detallado de los métodos de planificación basada en prioridades se acomete en el capítulo 4.

3.1 Los sistemas de tiempo real como sistemas concurrentes

3.1.1 Modelo de sistema.

Un sistema de tiempo real centralizado típico está compuesto por un procesador, memoria y una serie de dispositivos de entrada/salida. En general, su función será controlar y tratar un proceso físico, como se muestra esquematicamente en la figura 3.1. Los elementos de entrada/salida suelen ser:

- sensores, para medir parámetros significativos del proceso físico.
- actuadores, para modificar el comportamiento del proceso, según el comportamiento deseado.
- dispositivos de almacenamiento masivo, para almacenar información histórica del comportamiento del sistema o referente a parámetros de funcionamiento del sistema.
- elementos de interfaz hombre-máquina.

La interacción con los dispositivos externos se realiza de dos formas diferentes:

- Interrupciones: cuando se produce un evento determinado, el dispositivo interrumpe al procesador, que ejecutará una rutina de tratamiento. Este método optimiza

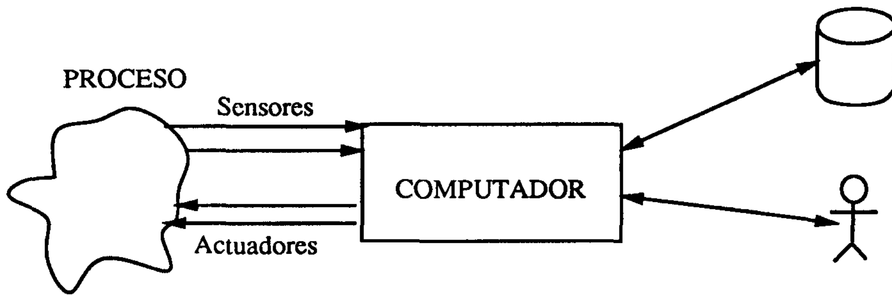

Figura 3.1: Estructura de un sistema de tiempo real.

el uso del procesador, dado que sólo se ejecutan las operaciones de entrada/salida cuando se necesitan. Sin embargo, su carácter asíncrono complica la estructura y el desarrollo de la aplicación.

- Consulta: cada cierto tiempo, el procesador comprueba el estado del dispositivo externo para determinar si el evento en cuestión ha ocurrido. Sus ventajas e inconvenientes son complementarios al método anterior. La estructura de la aplicación es más sencilla y su comportamiento determinista. Sin embargo, consume tiempo de procesador aunque no haya ocurrido evento alguno en el sistema.

Los sistemas de tiempo real son, normalmente, sistemas empotrados, que se caracterizan porque están incluidos físicamente en el sistema con el que interactúan o controlan. En estos casos, el computador no suele disponer de dispositivos de almacenamiento masivo, ni de interfaz hombre-máquina.

Los sistemas de tiempo real se suelen diseñar y realizar como un conjunto de tareas secuenciales autónomas, que ejecutan lógicamente en paralelo. Este enfoque se ha revelado como un método apropiado para desarrollar estos sistemas por varias razones, entre ellas:

- la abstracción de tarea es natural e intuitiva.
- refleja la concurrencia real de los procesos físicos sobre los que se actúa.
- simplifica la descomposición del problema en subproblemas más fácilmente abordables.

En la figura 3.2 se muestra un esquema sencillo de los diferentes estados por los que transita una tarea en su ciclo de vida. La tarea se *crea* y pasa al estado *ejecutable*, que se descompone en varios subestados. Inicialmente, la tarea estará *preparada* para ejecutar. Permanecerá en este estado hasta que el procesador esté disponible y entonces se *ejecutará*. En esta situación el sistema puede decidir ejecutar otra tarea, con lo que volverá al estado *preparada* o puede realizar operaciones bloqueantes, como algunas operaciones de entrada/salida, detenerse mientras no se cumpla alguna condición o hasta que ocurra un evento, y entonces quedará *bloqueada*. Cuando desaparezca la causa del bloqueo la tarea

pasará a preparada y, eventualmente, se volverá a ejecutar. En algunas circunstancias, cuando el transito de una tarea de bloqueada a preparada coincide con el inicio de un ciclo lógico de una operación, se dice que se *activa.* De igual forma, cuando el bloqueo de una tarea coincide con el fin de un ciclo lógico de una operación, se dice que la tarea *completa la activación.* Por ejemplo, esta nomenclatura se usa cuando una tarea periódica comienza o termina la ejecución correspondiente a un período. La tarea terminará en algún momento, bien porque se complete la ejecución de su código, sea abortada o se detenga el funcionamiento del sistema.

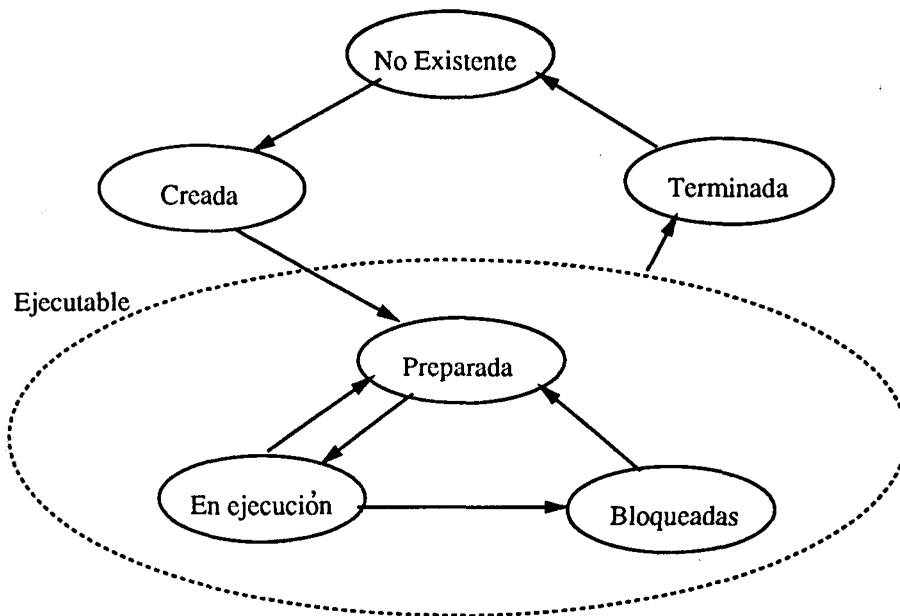

Figura 3.2: Estados de una tarea.

El sistema operativo o núcleo de ejecución realiza las operaciones de creación, terminación, bloqueo y selección de la tarea que debe ejecutar. En particular, el planificador es el encargado de seleccionar entre las tareas preparadas cuál es la que debe ejecutar. Si no hay tareas preparadas para ejecutar ni ejecutándose, se dice que el procesador está *ocioso.*

3.1.2 Tareas de tiempo real.

Las tareas de tiempo real tienen unos requisitos temporales que delimitan un intervalo de tiempo en que deben completar unas actividades concretas. Atendiendo a los requisitos temporales, una tarea de tiempo real puede ser:

- *Crítica:* El incumplimiento de un requisito temporal supone un fallo intolerable por sus consecuencias en el sistema.
- *Acrítica:* Se puede tolerar el incumplimiento ocasional de un requisito temporal.

La funcionalidad requerida en un sistema de tiempo real es muy diversa. En consecuencia, existen varios tipos de tareas, dependiendo sus características de ejecución:

- *Tareas periódicas*: son tareas que se activan con una periodicidad fija. Tienen un plazo de tiempo en el que se debe completar su ejecución cada vez que se activan.

  Los características temporales de una tarea periódica se representan mediante una terna (T, D, C), donde:

  - T: período de activación de la tarea.
  - D: plazo de respuesta máximo. Una vez activado, la tarea debe terminar en D unidades de tiempo.
  - C: tiempo máximo de cómputo de la tarea en cada activación.

  En algunas ocasiones, la tarea periódica no se activa hasta que transcurre un cierto tiempo desde el momento en que se inicia la ejecución del sistema. Este parámetro se denomina *fase*, Φ.

  Una aplicación de la tarea periódica es la realización de controladores de sistemas con un conjunto de entradas y salidas. El período de la tarea dependerá de la dinámica del sistema físico controlado y será el período de muestreo calculado teóricamente. En cada activación de la tarea, se leen las variables de entrada de los sensores, se aplica el algoritmo de control y, finalmente, se actúa sobre las variables de salida.

- *Tareas aperiódicas*: son tareas que sólo se ejecutan como respuesta a un evento que ocurre en el sistema, en instantes aleatorios. Las características temporales de una tarea aperiódica se representan mediante una terna (S, D, C), donde:

  - S: separación mínima entre dos activaciones sucesivas de la tarea.
  - D: plazo de respuesta máximo. Una vez activado, la tarea debe terminar en D unidades de tiempo.
  - C: tiempo máximo de cómputo de la tarea en cada activación.

  A las tareas aperiódicas con plazos de respuesta críticos, se les denomina *tareas esporádicas*.

  El objetivo de diseño de sistemas con estos dos tipos de tareas debe ser minimizar, en lo posible, los tiempos de respuesta de las tareas aperiódicas y asegurar el cumplimiento de los plazos de respuesta de las tareas esporádicas.

  Las tareas esporádicas se utilizan en sistemas de control para realizar alarmas. En el ejemplo mostrado en el apartado 2.1.1, la actividad que debe cerrar las válvulas de entrada al contenedor cuando el nivel alcanza un valor determinado, se realizaría mediante una tarea esporádica.

- *Tareas de segundo plano*: son tareas acríticas que sólo se ejecutan si no hay ninguna tarea de los tipos anteriores preparada para ejecutar.

  Las tareas de segundo plano, típicamente, realizan funciones de prueba del hardware, recopilación de datos para generar informes, etc.

3.1.3 Planificación de sistemas de tiempo real.

Uno de los problemas centrales en la construcción de sistemas de tiempo real es determinar si existe una planificación *admisible*, es decir, si existe una asignación de los recursos del sistema a las tareas de tiempo real, tal que éstas cumplan sus plazos de respuesta. Entre los recursos que hay que administrar están el procesador, la memoria, los dispositivos de entrada/salida, etc.

La gestión y planificación de recursos en sistemas operativos convencionales se suelen realizar con el objetivo primordial de obtener su máximo aprovechamiento y distribuir su uso entre los usuarios con equidad [Tanenbaum92][Peterson&89]. No suele haber requisitos temporales, y si los hay se expresan en términos de tiempos de respuesta medios. Por este motivo, estos tipos de planificadores no son aplicables a sistemas de tiempo real. Así pues, ha sido imprescindible desarrollar métodos específicos de planificación, cuyo objetivo principal sea asegurar el cumplimiento de los plazos de respuesta de las tareas [Stankovic&88b].

La planificación de sistemas de tiempo real es un problema complejo y en el caso general es NP-completo [Mok83]. Se puede simplificar considerando únicamente la planificación del procesador. En este caso, se supone que los recursos restantes estarán disponibles cuando una tarea los necesite. Esta aproximación es realista en sistemas monoprocesador. En estos casos, el sistema se diseña y desarrolla de forma que las tareas dispongan de la memoria que necesitan y la mayoría de los dispositivos externos en este tipo de sistemas, como sensores o actuadores, suelen ser utilizados por una tarea con exclusividad. En sistemas compuestos por varios procesadores, hay dispositivos que son por naturaleza compartidos, como es el caso del medio de comunicación, por lo que no se puede obviar su planificación.

En sistemas monoprocesador, si hay recursos comunes, se puede tratar su planificación encapsulando el acceso en una región crítica. Entonces el problema es asimilable al de comunicación entre tareas y se puede resolver fácilmente con los métodos que se mostrarán posteriormente.

La planificación de recursos puede ser:

- *Sin expulsión*: cuando se asigna un dispositivo a una tarea, esta lo utiliza hasta completar la operación.
- *Con expulsión*: el planificador puede desalojar a una tarea de un dispositivo, antes de completar su uso.

La planificación del procesador es más sencilla de realizar, si se hace expulsiva. Se ha demostrado que muchos problemas de planificación sin expulsión son NP-duros. Por

ejemplo, la planificación de un conjunto de tareas sin expulsión con tiempos de activación arbitrarios, es NP-duro incluso en sistemas monoprocesador [Lenstra&77].

La optimalidad y la estabilidad son dos propiedades deseables de los planificadores. Un planificador es *óptimo* cuando es capaz de encontrar una planificación admisible, si ésta existe. Un planificador es *estable* si en el caso de que no se puedan cumplir todos los plazos de respuesta de las tareas, fallarán las tareas menos prioritarias o menos críticas del sistema.

En el resto del capítulo se presentarán los principales métodos de planificación para sistemas monoprocesador.

3.2 Planificación síncrona.

3.2.1 Descripción del ejecutivo cíclico.

Un planificador *síncrono* se caracteriza porque el único evento que activa la ejecución de las tareas es el reloj. El ejecutivo cíclico es el método de planificación síncrona más extendido y el más utilizado industrialmente para desarrollar sistemas de tiempo real crítico.

Un ejecutivo cíclico [Baker&89] [Locke92] [Zamorano&92] consiste en un procedimiento que activa cíclicamente un conjunto de tareas según un orden prefijado, que se representa mediante un plan principal.

Este plan se determina estáticamente a partir de las características temporales del conjunto de tareas. El plan principal describe el conjunto de tareas y el orden en que se deben ejecutar durante un ciclo principal, CP. Su duración es el máximo común divisor de los períodos de las tareas, para que el número de activaciones de cada una sea fijo. Es decir, se cumple:

$$\forall \tau_i, \exists k \in \mathcal{N} | k \times T_i = CP$$

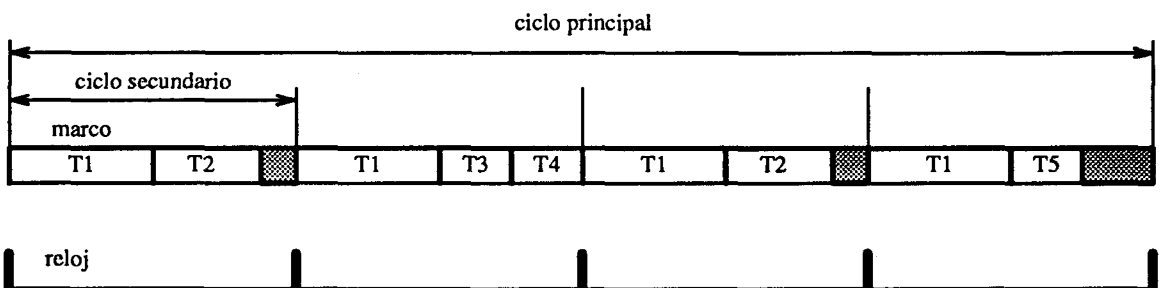

Figura 3.3: Estructura de una planificación cíclica.

El plan principal está formado por una serie de planes secundarios, los cuales determinan la secuencia de tareas a ejecutar durante un ciclo secundario, CS. En la figura 3.3 se muestra un esquema de un plan.

El ciclo secundario debe cumplir las siguientes condiciones [Baker&89]:

1. CS debe ser menor o igual que el plazo de ejecución de cualquier tarea.

$$\forall \tau_i : CS \leq D_i$$

2. CS debe ser mayor o igual que el mayor de los tiempos de cómputo máximo de los procesos.

$$\forall \tau_i : CS \geq \max(C_i)$$

3. CS debe ser un divisor de la duración del ciclo principal, CP.

$$\exists k \in \mathcal{N} \mid CP = k \times CS$$

4. CS debe cumplir:

$$\forall i : CS + (CS - mcd(CS, T_i)) \leq D_i$$

   Esta condición incluye a la anterior. Exige que entre el instante de activación de cada tarea y su plazo límite haya un ciclo secundario completo. Esta condición permite detectar fallos en el cumplimiento del plazo de respuesta.

La duración de los planes secundarios se ajusta con temporizadores. Cuando ocurre el instante de inicio de un plan secundario se comprueba si la tarea que está ejecución pertenece al plan secundario anterior. Evidentemente esta situación es errónea y los métodos de tratamiento de fallos del sistema deberán ocuparse de esta situación.

Los planes secundarios se dividen en marcos, que determinan el tiempo de cómputo máximo permitido a una tarea. La duración de los marcos también está controlada por temporizadores. De esta forma se detecta cuándo una tarea utiliza más tiempo de cómputo que el máximo asignado. En este caso, la tarea se expulsa y se toman las acciones necesarias para tratar el fallo.

La generación de los planes a partir de la especificación de las tareas es un problema complejo. En la práctica se suele simplificar con objeto hacer más manejable el diseño del plan. La simplificación más común consiste en ajustar los períodos originales de las tareas para que sean armónicos. De esta forma se reduce la complejidad de la selección del tamaño de los ciclos principal y secundario. Sin embargo, este enfoque aumenta artificialmente la carga del sistema, ya que en esta transformación los períodos se reducen, sin que los tiempo de cómputo se modifiquen.

Es evidente que todos los eventos que ocurren en un sistema de tiempo real no son síncronos. El tratamiento de los eventos asíncronos se realiza mediante servidores de consulta. Son tareas periódicas que al inicio de cada activación comprueban si se ha producido el evento asociado y, si es así, se ejecuta el código de tratamiento.

El ejecutivo cíclico permite definir tareas de segundo plano, que se ejecutarán cuando el procesador esté ocioso. Es decir, si las tareas de un plan secundario han completado la

activación y aún no se ha llegado al instante de inicio del siguiente plan secundario, este tiempo se puede emplear para ejecutar las tareas acríticas. Cuando llegue el instante de inicio del siguiente plan secundario, la tarea acrítica que se esté ejecutando se expulsará del planificador y se ejecuta la primera tarea de este plan.

La comunicación entre tareas en el ejecutivo cíclico es sencilla. Como se sabe con absoluta certeza qué tarea se va a ejecutar en cada momento, se puede asegurar que no habrá problemas de carreras que produzcan datos inconsistentes.

3.2.2 Evaluación

El ejecutivo cíclico presenta ciertas ventajas que han favorecido su uso, especialmente en sistemas con requisitos críticos de seguridad:

- Es un método determinista y predecible. Se sabe con exactitud lo que está ocurriendo en el procesador en cada instante.
- Está muy experimentado.
- Su implementación es muy sencilla y la sobrecarga del planificador es muy pequeña. Cuando una tarea termina su activación, el planificador sólo tiene que programar el temporizador con la duración del siguiente marco, y ceder el control a la siguiente tarea del plan.
- En suma, es un planificador robusto y fiable.

Sin embargo, su estructura y concepción conlleva ciertos problemas:

- El diseño de los planes a partir de las especificaciones de las tareas es muy laborioso, complicándose según aumenta el número de tareas.
- Su mantenimiento es costoso. Cambios pequeños en las especificaciones del sistema suelen implicar la necesidad de replantearse el conjunto del plan.
- Como el ciclo mayor es el máximo común divisor de los períodos de las tareas, el desarrollo del plan se complica enormemente cuando los períodos de las tareas no son armónicos. La transformación de los períodos en armónicos, supone el aumento artificial de la carga del procesador, contrarrestando en parte las ventajas derivadas de la pequeña sobrecarga que introduce el planificador.

En conclusión, su bajo nivel de abstracción, los elevados costes que implica su desarrollo y la aparición de otros métodos con similar nivel de fiabilidad y predecibilidad, hacen que el ejecutivo cíclico no sea la alternativa más recomendable para desarrollar sistemas de tiempo real críticos. Sin embargo, su pequeña sobrecarga y tamaño pueden ser factores determinantes para ser usado en aplicaciones específicas, como las espaciales que emplean computadores con limitaciones de tamaño o peso y de generaciones tecnológicas anteriores.

3.3 Planificación basada en prioridades.

Los métodos de planificación basada en el concepto de prioridad son muy utilizados en el desarrollo de sistemas concurrentes. La prioridad, *pr*, es una indicación de la urgencia relativa de la ejecución de una tarea. Es un concepto muy intuitivo y eficaz, que se ha aplicado con éxito en el desarrollo de sistemas concurrentes. En este contexto, el planificador elegirá para ejecutar la tarea preparada con prioridad más urgente.

El uso tradicional de este concepto no es directamente aplicable a los sistemas de tiempo real. En sistemas de propósito general, la asignación de prioridades a tareas se suele realizar en función de su importancia o criticidad. Este criterio no es válido en sistemas de tiempo real. En efecto, supóngase un sistema compuesto por dos tareas periódicas con las siguientes características (las unidades de estos parámetros son unidades de tiempo genéricas):

$$\tau_1 : \; T_1 = 50; \quad C_1 = 5; \quad D_1 = 50$$
$$\tau_2 : \; T_2 = 5; \quad C_2 = 1; \quad D_2 = 5$$

Si τ_1 es más importante y se le asigna una prioridad mayor que a τ_2, entonces ésta no cumplirá su plazo de respuesta en la primera activación. Si por el contrario, se asigna mayor prioridad a τ_2, ambas tareas cumplirán sus plazos de respuesta.

La planificación de tareas en sistemas de tiempo real es comúnmente expulsiva. Esto quiere decir que si una tarea bloqueada pasa a preparada y es más prioritaria que la que se está ejecutando actualmente, entonces se expulsa a ésta y se ejecuta la primera.

3.3.1 Método de prioridad a la tarea más frecuente.

Uno de los primeros trabajos de planificación de tareas periódicas en sistemas monoprocesador fue publicado por Liu y Layland en el año 1973 [Liu&73]. En este artículo se enuncia un método de planificación basado en prioridades que ha influido decisivamente en los desarrollos posteriores sobre este tema. Encontrar una planificación admisible de un conjunto de tareas de tiempo real es un problema complejo. Para tratarle analíticamente, formularon las siguientes suposiciones sobre el conjunto de tareas [1]:

1. Todas las tareas críticas del sistema son tareas periódicas.
2. El plazo de respuesta de cada tarea coincide con su período.
3. Las tareas son independientes, es decir, no se comunican, ni comparten recursos, ni hay relaciones de precedencia entre ellas.
4. El tiempo de cómputo máximo de cada tarea está acotado.
5. Las tareas no se suspenden mientras ejecutan el código correspondiente a una activación.

[1] Mientras no se diga lo contrario, se supone que las tareas se ajustan a esta descripción

6. Si en el sistema hay tareas aperiódicas, no tienen plazos de respuesta críticos.
7. Las tareas se ejecutan mediante un planificador guiado por prioridades y expulsivo.

En un sistema con las características anteriores, se enuncia el método de planificación de prioridad a la tarea más frecuente (*Rate-monotonic scheduling, RMS*):

Teorema 3.1 (Liu y Layland) *Si existe una asignación de prioridades a un conjunto de tareas con las características enumeradas que produzca una planificación admisible, la asignación de prioridades más urgentes a las tareas más frecuentes también será admisible.*

Este método tiene las siguientes propiedades [Sha&86]:

- *Optimalidad*: Si existe una asignación de prioridades que sea admisible para un conjunto de tareas, la asignación de prioridades obtenida con este método también será admisible
- *Estabilidad*: Si un sistema está sobrecargado y no todas las tareas pueden cumplir sus plazos de respuesta, fallarán las tareas menos prioritarias.
- *Determinista*: Si se dispone de las características de las tareas y ciertos parámetros del entorno de ejecución, se puede comprobar analíticamente la planificabilidad de un conjunto de tareas.

Análisis de planificabilidad

El objetivo del análisis de planificabilidad es comprobar si un conjunto de tareas cumple sus plazos de respuesta en cualquier circunstancia. Esto implica que se debe comprobar si el conjunto es planificable teniendo en cuenta el caso más desfavorable.

Teorema 3.2 (Liu y Layland) *Sea $\tau_1, \tau_2, \ldots \tau_n$ un conjunto de tareas. El tiempo de respuesta más largo para cualquier tarea ocurre cuando se activan todas las tareas al mismo tiempo. Ocurre al iniciar el sistema, si se cumple $\Phi_1 = \Phi_2 = \ldots = \Phi_n = 0$*

A este instante se le denomina instante crítico.

Este resultado permite determinar la ejecución más desfavorable de un conjunto de tareas bajo cualquier relación de fases. A partir de este resultado, Liu y Layland [Liu&73] enunciaron una condición suficiente para que un conjunto de tareas sea planificable:

Teorema 3.3 (Liu y Layland) *Sea $\tau_1, \tau_2, \ldots \tau_n$ un conjunto de tareas con prioridades. El conjunto es planificable, para cualquier relación de fases, si:*

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1) \tag{3.1}$$

Es decir, si la ocupación máxima del procesador es menor que un cierto umbral, se puede garantizar que las tareas cumplirán sus plazos de respuesta. Según este método, cuando n tiende a infinito, la ocupación máxima posible del procesador es, aproximadamente, $0,693$.

Ampliaciones del método.

A pesar de su interés el método RMS no se ha utilizado intensamente por las restricciones que impone a las tareas. Sin embargo, los trabajos de investigación realizados, especialmente en el *Software Engineering Institute* de la *Universidad de Carnegie-Mellon* (SEI-CMU) y en la *Universidad de York*, han permitido desarrollar métodos que permiten extender el tipo de aplicaciones en las que se puede aplicar. En concreto, se han desarrollado métodos para realizar análisis de planificabilidad precisos [Lehoczky&89], comunicar tareas [Sha&87] [Goodenough&88] [Sha&89a] [Audsley91b] [Klein&90] [Rajkumar&88], tratar sobrecargas transitorias [Sha&86], incluir tareas acíclicas [Lehoczky&87] [Sprunt&88] [Sprunt&89] [González&91], cambiar el modo del sistema [Sha&89c] [Tindell&92], reducir los plazos de respuesta [Leung&82] [Audsley&91] [Audsley91a], etc.

En el capítulo 4 se presentan estas ampliaciones en detalle.

3.3.2 Evaluación.

El resultado final de este intenso trabajo es la suficiente madurez y generalidad del método de planificación para su empleo en aplicaciones reales. Es un método utilizado por la ESA (*European Space Agency*) [Burns&91] y por la NASA en el desarrollo de la estación espacial FREEDOM [Gafford90]. Además las normas IEEE Futurebus+ [Futurebus91], IEEE POSIX 1003.1b [POSIX.1b 93], IEEE POSIX 1003.1c [POSIX.1c 93] y Ada9X [Ada9xMap92] [Ada9XMaS92] [Ada9XAnS92] disponen de mecanismos adecuados para desarrollar aplicaciones con este método de planificación.

Las ventajas principales que comporta su uso son [Locke92] [Puente&92]:

- La principal ventaja, que influye positivamente en el coste total del ciclo de vida de la aplicación, es la posibilidad de determinar analíticamente que un conjunto de tareas cumplirá sus plazos de respuesta. Esto es aplicable desde las primeras fases de diseño para determinar si el sistema es planificable a partir de una primera estimación de los tiempo de cómputo. El mantenimiento se simplifica, pues si se cambia la especificación del conjunto de tareas, se puede determinar fácilmente si sigue siendo planificable.

- Los períodos pueden ser armónicos o no, sin que este factor afecte a la complejidad del desarrollo del sistemas.

- Este modelo es ampliamente aplicable, pues requiere un planificador expulsivo y con prioridades, que está ampliamente difundido en productos comerciales y normas internacionales [AdaLRM83] [Ready86] [POSIX.1b 93].

Las desventajas que presenta este modelo de planificación son:

- El método RMS no define con precisión el intervalo de tiempo que terminará la activación de una tarea. Sólo asegura que lo hará antes del siguiente período. En determinadas aplicaciones este comportamiento puede ser problemático. Utilizando técnicas de modificación de período (sección 4.1.2) se puede resolver el problema.
- El planteamiento teórico del método no considera ciertos problemas de bajo nivel que es necesario resolver para realizar sistemas que funcionen correctamente. Así por ejemplo, el cálculo del tiempo de cómputo máximo de una tarea no es un problema obvio [AdaPI90], Este problema se simplificaría silos compiladores y herramientas de diseño proporcionaran alguna ayuda.
- Una característica de los sistemas de la siguiente generación es su dinamicidad ante cambios en el entorno. Los métodos basados en prioridades son poco dinámicos, por naturaleza. Paradigmas como el cambio de modo o el reemplazamiento de software dinámico palían parcialmente este problema.

3.4 Planificación basada en plazos.

3.4.1 Método basado en la proximidad del plazo de respuesta.

Los métodos de planificación basada en plazos seleccionan la tarea que debe ejecutar en cada instante, a partir de parámetros relativos al estado de ejecución de la tarea en cada momento.

El primer método de asignación de prioridades dinámicas enunciado está basado en la proximidad de los plazos de respuesta (*earliest deadline first, EDF*). Consiste en ejecutar en cada momento la tarea con el plazo de respuesta más cercano. Los instantes de planificación son aquéllos en los que una tarea completa o inicia la ejecución de una activación.

Este método también fue introducido por Liu y Layland [Liu&73] quienes demostraron que si un conjunto de tareas cumple las condiciones enumeradas en el apartado 3.3, el método es óptimo, en el sentido que si existe una planificación admisible, la encuentra. El siguiente teorema, enunciado por Liu y Layland, constata esta característica:

Teorema 3.4 (Liu y Layland) *Sea $\tau_1, \tau_2, \ldots \tau_n$, planificadas según el método de prioridad a la tarea más urgente.*

El conjunto de tareas será planificable si y sólo si:

$$\sum_{i=1}^{n} \frac{C_j}{T_j} \leq 1$$

La característica más interesante de este método es su habilidad para encontrar una planificación admisible si existe. Evidentemente, es imposible construir una planificación

admisible de un conjunto de tareas, si la capacidad de cómputo que solicitan es mayor que la disponible.

Sin embargo, este método aumenta la complejidad y, consecuentemente, la sobrecarga del planificador en tiempo de ejecución, al tener que examinar los plazos de respuesta de las tareas para determinar cuál debe ejecutarse. La forma de incluir esta sobrecarga en el análisis de planificabilidad consiste en añadirla al tiempo de cómputo de las tareas, junto con el coste de los cambios de contexto. Por tanto, como se debe considerar el caso más desfavorable, se pierde capacidad de cómputo.

El método no es estable, es decir, si a causa de una sobrecarga transitoria alguna tarea falla, ésta puede ser cualquiera del conjunto. En especial, puede ser alguna de las tareas críticas del sistemas. Por tanto, el enunciado original del método obliga a que las únicas tareas que pueden formar parte del sistema son las garantizadas. Si los tiempos de cómputo máximos son mucho mayores que los tiempos medios, el resultado será que el procesador estará durante mucho tiempo ocioso.

Este problema se puede resolver parcialmente si se dividen las tareas en críticas y acríticas y el planificador siempre da preferencia a las tareas críticas. Sin embargo, soluciones de este tipo, aumentan aún más la sobrecarga de planificación.

Otro método de planificación basado en plazos es el **método de prioridad a la tarea con la menor holgura** [Mok84a], donde la holgura de una tarea, τ_i, en un instante t se define como la diferencia entre el tiempo que resta hasta el plazo de respuesta $(D_i - t)$, y el tiempo de cómputo que le falta para completar su ejecución. Las propiedades de este método son similares a las del anterior. Sin embargo, la sobrecarga en ejecución para determinar la siguiente tarea a ejecutar es aún mayor.

Comunicación entre tareas.

Dado su interés práctico, varios trabajos de investigación han desarrollado métodos de comunicación de tareas en sistemas de tiempo real con planificación basada en plazos [Mok84a] [Chen&89] [Chetto&90].

El primer trabajo [Mok84a] trata la comunicación entre tareas como sincronización y lo limita a tareas periódicas con períodos tales que el menor es múltiplo del mayor. De esta forma, hay un número fijo de sincronizaciones durante el período de la mayor. El método divide el código de las tareas en bloques, delimitados por sincronizaciones con otras tareas. A continuación se establece un grafo de precedencia entre los bloques de cada tarea y se determinan los plazos de respuesta de los bloques, para que las tareas dependientes cumplan sus plazos. Este método es farragoso y limita las formas de comunicar tareas. Aunque en trabajos posteriores se depura este método [Chetto&90], sus desventajas siguen siendo importantes.

El método del protocolo del techo de prioridad dinámico [Chen&89] intenta compatibilizar planificación basada en plazos con los métodos de comunicación basados en el protocolo del techo de prioridades (ver apartado 4.3.3), que se define como la prioridad mayor entre las tareas que acceden a un objeto compartido. El método ordena las tareas, según su urgencia y las asigna prioridades. Estas prioridades son dinámicas, pues varían

durante la ejecución del sistema. El techo de prioridad también varía y hay que recalcularlo a la vez que aquellas. Este enfoque previene bloqueos encadenados e interbloqueos. Sin embargo, la necesidad de reordenar frecuentemente las tareas es un inconveniente importante.

3.4.2 Evaluación

Los métodos de planificación basada en plazos presentados permiten obtener una planificación admisible, si existe y su capacidad de adaptarse a cambios dinámicos en el entorno es mayor. Sin embargo, las desventajas que se enumeran, han impedido una implantación más amplia:

- La sobrecarga del planificador es mayor que en los casos anteriores, al tener que comprobar los plazos de las tareas preparadas para decidir la que debe ejecutar.
- El método no es estable, con los inconvenientes comentados.
- La utilización de estos métodos suelen implicar diseñar un planificador a medida, ya que los planificadores comerciales suelen estar basados en prioridades.

3.5 Conclusiones

En este capítulo se han revisado tres métodos de planificación de sistemas de tiempo real críticos en monoprocesador. Teniendo en cuenta el estado de la tecnología y sus respectivos méritos, los métodos de planificación basada en prioridades son los más adecuados actualmente para desarrollar sistemas de tiempo real.

Por esta razón, el trabajo realizado en esta tesis está relacionado con estos métodos. En el capítulo siguiente se presentan en detalle los trabajos de ampliación de la planificación basada en prioridades y se identifican los temas pendientes de resolución que tratará este trabajo.

Capítulo 4

Planificación basada en prioridades.

Los primeros estudios de planificadores expulsivos basados en prioridades para desarrollar sistemas de tiempo real críticos, dieron como resultado el enunciado del método de prioridad a la tarea más frecuente. El enunciado original del método (teorema 3.1) imponía restricciones severas a las características de las tareas que pueden formar parte de sistema.

Este capítulo contiene las ampliaciones más relevantes de este método de planificación. En concreto, se incluyen el método de prioridad a la tarea más urgente y métodos de análisis de planificación más precisos, de comunicación entre tareas, de tratamiento de tareas aperiódicas y de cambio de modo.

4.1 Prioridad a la tarea más frecuente.

4.1.1 Análisis de planificabilidad.

En el apartado 3.3 se enunció el método de prioridad a la tarea más frecuente (RMS) y una condición necesaria para garantizar la planificabilidad de un conjunto de tareas de tiempo real. Este cálculo es pesimista, ya que considera el caso con las características más desfavorables de las tareas. Por ejemplo, si el conjunto de tareas es armónico, se puede conseguir una planificación admisible con una utilización del procesador del 100%. Como consecuencia, se han investigado métodos más precisos para analizar la planificabilidad de un conjunto de tareas, basados en sus características específicas.

Lehoczky, Sha y Ding [Lehoczky&89] propusieron un método sistemático para comprobar la planificabilidad de un conjunto de tareas. Básicamente, se estudia si en algún instante entre la activación de la tarea y el plazo de respuesta, la carga solicitada al procesador es menor que la unidad. Esta condición implica que todas las tareas han completado la ejecución correspondiente a la última activación.

Teorema 4.1 (Lehoczky, Sha y Ding) *Sea $\tau_1, \tau_2, \ldots \tau_n$, un conjunto de tareas en orden decreciente de prioridades.*
τ_i cumplirá su plazo de respuesta si y sólo si:

$$\min_{0 \leq t \leq T_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \leq 1 \tag{4.1}$$

El conjunto de tareas será planificable para cualquier relación de fases si y sólo si:

$$\max_{1 \leq i \leq n} \min_{0 \leq t \leq T_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \leq 1 \tag{4.2}$$

Esta función es discontinua y tiene un número finito de puntos de discontinuidad. Por esta razón, no es directamente aplicable para determinar la planificabilidad de un sistema. Sin embargo, se puede refinar teniendo en cuenta que no es necesario comprobar la validez del algoritmo para cualquier valor de la variable t, sino en una serie de puntos muy concretos, llamados *puntos de planificación.* Estos puntos coinciden con valores de t múltiplos del período de alguna de las tareas y en ellos se obtiene un mínimo local de la función que representa la carga solicitada al procesador por las tareas.

Teorema 4.2 (Lehoczky, Sha y Ding) *Sea $\tau_1, \tau_2, \ldots \tau_n$, un conjunto de tareas en orden decreciente de prioridades.*

Sea S_i el conjunto de los puntos de planificación *para el conjunto de tareas dado, definido como sigue:*

$$S_i = \left\{ k.T_j \mid j = 1, \ldots, i; \ k = 1, \ldots, \left\lfloor \frac{T_i}{T_j} \right\rfloor \right\} \tag{4.3}$$

τ_i cumplirá todos sus plazos de respuesta para cualquier relación de fases si y sólo si:

$$\min_{t \in S_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \leq 1 \tag{4.4}$$

El conjunto de tareas será planificable si y sólo si:

$$\max_{1 \leq i \leq n} \min_{t \in S_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \leq 1 \tag{4.5}$$

Este método es fácilmente programable, lo que permite su aplicación práctica. Para cada tarea, el algoritmo debe comprobar si en un instante crítico cumple su plazo de respuesta. En el momento en que se cumpla esta condición, la tarea será planificable. La aplicación del método continúa hasta el plazo de respuesta de la tarea. Si en este instante no se ha completado, entonces la tarea no es planificable.

4.1.2 Sobrecarga transitoria del sistema

En la mayoría de las aplicaciones, los tiempos de cómputo de las tareas son estocásticos, y el caso peor son mayores que los tiempos medios de ejecución. Si sólo se incluyeran en los sistemas aquellas tareas que pueden ser planificadas con seguridad, la utilización del procesador sería en media muy baja.

Si se incluyen más tareas de las que se puede asegurar que cumplirán sus requisitos bajo cualquier carga, puede ocurrir que haya tareas críticas para la aplicación que incumplan sus restricciones temporales, con posibles efectos peligrosos en el sistema controlado.

Para resolver el problema es necesario que el algoritmo de planificación, además de ser óptimo, sea estable, y el método de planificación RMS lo es. Por tanto, si todas las tareas no pueden cumplir sus plazos de respuesta, fallarán las tareas con menor prioridad. En consecuencia, cuando se diseña un sistema de tiempo real con este método, se debe asegurar que el subconjunto de tareas que siempre cumplirán sus plazos, incluye a las tareas críticas para el correcto funcionamiento del sistema.

Por las características del método de asignación de prioridades, puede ocurrir que en un sistema particular el período de una tarea crítica sea mayor que el de una tarea acrítica y, por tanto, su prioridad sea menor. En esta situación, la tarea crítica podría incumplir su plazo de respuesta, mientras que la tarea acrítica lo cumple. A continuación se presenta una solución sistemática que, en esta situación y hay capacidad de procesamiento suficiente, permite obtener un conjunto de tareas equivalente, tal que se pueda garantizar los plazos de respuesta de las tareas.

Método de transformación del período.

El objetivo de este método es conseguir que las tareas más críticas del sistema tengan los períodos menores y, por tanto, tengan mayor prioridad [Sha&86]. Para tal fin se transforma el período de algunas tareas como sigue:

- Se reduce el período de una tarea τ_i a T_i/k ($k = 1, 2, \ldots$) y se divide el código de la tarea en porciones de duración máxima C_i/k. En cada activación de la tarea se ejecutan sucesivamente cada una de las porciones de código.

- Se aumenta el período de una tarea τ_i creando k tareas iguales a la original pero con período $k \cdot T_i$ y desfasadas entre ellas en T_i. Esta modificación no es siempre factible, dado que al aumentar el período de una tarea τ_i que comienza a ejecutar en el instante t, su plazo de respuesta será $t + k \cdot T_i$ y es posible que este plazo sea incompatible con la especificación del sistema.

Un procedimiento sistemático para conseguir que las tareas críticas de un conjunto sean las que tienen mayor prioridad es el siguiente:

1. Se identifica el conjunto de tareas críticas que son planificables en las peores condiciones.

2. Dado que el conjunto de tareas está divido en críticas y acríticas, se ordenan ambos conjuntos según la prioridad asignada. Se denomina $pr_{c,min}$ a la menor prioridad del conjunto de tareas críticas y $pr_{a,max}$ a la mayor prioridad entre las tareas no críticas.

   - Si $pr_{c,min} > pr_{a,max}$ entonces no es necesario transformar el período.
   - Si $pr_{c,min} = pr_{a,max}$ entonces se debe deshacer la igualdad asignando a la tarea $\tau_{c,min}$ una prioridad mayor, y no habrá que realizar transformación de período.
   - Si $pr_{c,min} < pr_{a,max}$ entonces se comprueba si la tarea $\tau_{a,max}$, junto al conjunto de tareas críticas, es ahora planificable. Si es así, se incluye la tarea en el conjunto de tareas críticas. Se repite este procedimiento hasta que se cumpla una de las dos condiciones siguientes:
     (a) En la partición actual $pr_{c,min} \geq pr_{a,max}$, entonces no es necesario transformar el período.
     (b) Si aún $pr_{c,min} < pr_{a,max}$, no se puede incluir $\tau_{a,max}$ en el conjunto crítico, habrá que transformar los períodos. En primer lugar se aumenta el período de tareas acríticas, pues así se incrementa la capacidad de planificación de tareas críticas. En concreto, se intenta aumentar el período de $\tau_{a,max}$, hasta que sea mayor que el de la tarea $\tau_{c,min}$. Entonces se selecciona la siguiente $\tau_{a,max}$ y se comprueba si $pr_{c,min} > pr_{a,max}$. Si se cumple esta condición, ya se ha logrado el objetivo, sino se repite este procedimiento. Puede ocurrir que el período de una determinada tarea $\tau_{a,max}$ no pueda ser aumentado. En este caso se deberá reducir el período de todas las tareas críticas cuyo período sea mayor que $\tau_{a,max}$.

4.2 Prioridad a la tarea más urgente.

4.2.1 Planteamiento.

La teoría presentada en el apartado anterior forma una base adecuada para desarrollar sistemas de tiempo real críticos, cuando los plazos de respuesta coinciden con los períodos. Si bien esta restricción es en general admisible para tareas periódicas, cuando se trata de tareas esporádicas la relación entre el plazo de respuesta y la separación entre eventos es difícil de establecer.

La eliminación de esta restricción permite resolver problemas adicionales [Burns&91]. En particular, las tareas pueden satisfacer otros requisitos como:

- *Determinar el instante de terminación de una tarea con mayor precisión.* Con el esquema del algoritmo de prioridad a la tarea más frecuente, una tarea completa su activación en el intervalo $[t_a + C_{min}, t_a + T]$; donde C_{min} es el tiempo de cómputo mínimo y t_a es el instante de activación. Si fuera necesario determinar con mayor

precisión el final de una activación y mantener el período, sólo hay que reducir el plazo de respuesta y así se puede garantizar que la tarea acabará en el intervalo $[t_a + C_{min}, t_a + D]$.

- *Incorporar tareas esporádicas de forma natural.* Los plazos de respuesta las tareas esporádicas pueden ser menores que la separación entre eventos. El análisis de planificabilidad correspondiente no considera que éste sea un caso particular.
- *Proporcionar una abstracción más útil para sistemas distribuidos.* Un enfoque al desarrollo de sistemas distribuidos es considerar que las operaciones que realiza una tarea en una activación son la lectura de los datos de entrada, el cálculo de los resultados y su envío a una tarea remota. Entonces, el plazo de respuesta se referiría al instante en que el mensaje llega a su destino. Si d es el tiempo necesario para transmitirle entonces la ejecución del cálculo debería completarse antes de $T - d$.

A continuación se presenta la teoría básica de planificación de prioridad a la tarea más urgente, junto con las extensiones adecuadas para su empleo en el desarrollo de sistemas de tiempo real industriales.

4.2.2 Enunciado del método

La teoría de planificación de prioridad a la tarea más urgente fue enunciada por Leung y Whitehead [Leung&82] (*Deadline-Monotonic Scheduling, DMS*), para un conjunto de tareas con las mismas características especificadas en el apartado 3.3.1, excepto que los plazos de respuesta de las tareas pueden ser menores que sus períodos.

Teorema 4.3 (Leung y Whitehead) *Si existe una asignación de prioridades a un conjunto de tareas con las características enumeradas que produzca una planificación admisible, la asignación de prioridades mayores a las tareas con los plazos de respuestas menores, es decir más urgentes, también será admisible.*

Este método es óptimo, estable y determinista [Leung&82] [Burns&91].

En el caso particular de que todos los plazos de respuesta de las tareas coincidan con los períodos, las asignaciones de prioridades resultantes de la aplicación de los métodos de prioridad al más frecuente (RMS) y prioridad al más frecuente (DMS) serán idénticas. En concreto, el método RMS es un caso particular del método DMS.

4.2.3 Análisis de planificabilidad

En este apartado se presentan tres métodos de análisis de planificabilidad de un conjunto de tareas. El primero es una generalización del teorema 4.1 a situaciones en que $D_i \leq T_i$ [Lehoczky&91]:

Teorema 4.4 (Lehoczky, Sha y Ding) *Sea $\tau_1, \tau_2, \ldots \tau_n$, un conjunto de tareas en orden decreciente de prioridades.*

τ_i cumplirá su plazo de respuesta si y sólo si:

$$\min_{0 \le t \le D_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \le 1$$

El conjunto de tareas será planificable para cualquier relación de fases si y sólo si:

$$\max_{1 \le i \le n} \min_{0 \le t \le D_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \le 1$$

En [Audsley91a] [Audsley&91] [Audsley&92] se enuncian dos métodos de análisis de la planificabilidad. Uno de ellos establece una condición necesaria para que un conjunto de tareas sea planificable y el segundo determina una condición necesaria y suficiente. Para enunciarlos se define la *interferencia*, I_i de una tarea τ_i, como el tiempo durante el que τ_i permanece preparada para ejecutar por existir otras tareas preparadas más prioritarias.

Teorema 4.5 (Audsley) *Sea $\tau_1, \tau_2, \ldots \tau_n$ un conjunto de tareas, en orden decreciente de prioridades.*

El conjunto de tareas será planificable, para cualquier relación de fases, si:

$$\forall i : 1 \le i \le n :$$

$$\frac{C_i}{D_i} + \frac{I_i}{D_i} \le 1$$

donde I_1 vale 0 y para $i > 1$:

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{T_j} \right\rceil C_j$$

Teorema 4.6 (Audsley) *Sea $\tau_1, \tau_2, \ldots \tau_n$ un conjunto de tareas en orden decreciente de prioridades. El conjunto es planificable, para cualquier relación de fases, si y sólo si todas las tareas son planificables según el algoritmo que se presenta a continuación. Esta prueba se debe repetir hasta que se compruebe que la tarea cumple o no el primer plazo de respuesta:*

$$\frac{I_i^{t_0}}{t_0} + \frac{C_i}{t_0} \le 1$$

$$donde \;\; t_0 = \sum_{j=1}^{i} C_j$$

$$\frac{I_i^{t_1}}{t_1} + \frac{C_i}{t_1} \leq 1$$

$$donde \ t_1 = I_i^{t_0} + C_i$$

$$\ldots$$

$$\frac{I_i^{t_k}}{t_k} + \frac{C_i}{t_k} \leq 1$$

$$donde \ t_k = I_i^{t_{k-1}} + C_i$$

y donde

$$I_y^x = \sum_{z=1}^{y-1} \left\lceil \frac{x}{T_z} \right\rceil C_z$$

El teorema 4.5 es el más sencillo, por lo que se suele aplicar en primer lugar. Si el resultado es negativo, puede ocurrir que el conjunto de tareas sea planificable, ya que sólo se comprueba una condición necesaria. En este caso se debe aplicar alguno de los otros para asegurar que el sistema no es planificable. Los teoremas 4.4 y 4.6 son equivalentes. Los criterios para seleccionar uno de ellos estarán relacionados con cuestiones de eficiencia de su automatización.

4.2.4 Sobrecarga transitoria del sistema.

El método de asignación estática de prioridades a las tareas más urgentes es estable. Por tanto, es posible diseñar un sistema con una carga de procesador mayor que la planificable. Mientras que se garanticen los plazos de respuesta de las tareas críticas, la seguridad del sistema no estará comprometida. En el apartado 4.1.2 se presentó un método para que un sistema cumpla esta condición, cuando sea posible.

En este apartado se proponían dos formas de modificar el período y, consecuentemente, la prioridad. En relación al método que nos ocupa, lo que habrá que modificar es el plazo de respuesta. A continuación se analizan las dos técnicas que se propusieron a la luz de esta teoría de planificación:

- Reducción del plazo de respuesta. Una tarea, τ, se transforma en k iteraciones de una nueva tarea, τ'. Esta tarea tendrá un período T/k, un plazo de respuesta de $T/k - (T - D)$. El código de la tarea se divide en k partes de duración C/k y que se ejecutan ordenadamente en activaciones sucesivas.

- Aumento del plazo de respuesta. Cuando se aplica está reducción, es porque la tarea en cuestión no es crítica. Por consiguiente, inicialmente se debería aumentar el plazo de respuesta hasta igualarlo con el período. Si fuera necesario aumentar más aún el plazo de respuesta, se podría hacer de forma análoga a la planteada en la sección 4.1.2.

El procedimiento sistemático se puede aplicar de forma análoga al presentado anteriormente.

4.3 Comunicación entre tareas.

4.3.1 Planteamiento.

Los métodos de planificación basada en prioridades que se han presentado, presuponen que no hay comunicación entre las tareas. Esta restricción evita los problemas generados por la sincronización de tareas al acceder a dispositivos físicos o lógicos compartidos. Sin embargo, estos métodos no serán útiles para diseñar sistemas de interés práctico si impide que las tareas se comuniquen.

El bloqueo de una tarea es una situación común cuando se solicita acceso a un recurso compartido con requisitos de exclusión mutua y que está siendo utilizado por otra tarea. Un ejemplo típico acontece cuando dos tareas comparten datos. Para mantener su consistencia el acceso se debe serializar. Aunque este modo de operar es el único posible, puede producir *inversión de prioridades*, que ocurre cuando una tarea es bloqueada por otra tarea de menor prioridad. Así por ejemplo, una tarea será bloqueada si intenta acceder a los datos mientras una tarea de menor prioridad está operando con ellos. Esta situación es peligrosa, ya que si la inversión de prioridades no se controla, la tarea más prioritaria podría incumplir su plazo de respuesta. Si la duración del bloqueo no se puede acotar, es imposible garantizar el cumplimiento de los plazos de respuesta de un conjunto de tareas.

El problema anterior es independiente del mecanismo de comunicación entre tareas. En lo sucesivo se supone que las tareas se comunican mediante monitores. Los protocolos que se van a presentar se pueden extender a otros mecanismos de comunicación entre tareas como son los semáforos.

Para ilustrar este problema, supóngase que se tienen tres tareas τ_1 τ_2 y τ_3 y que τ_1 y τ_3 se comunican mediante un monitor M. Las prioridades de las tareas cumplen la siguiente relación:

$$pr_1 > pr_2 > pr_3$$

Si se activan las tareas en orden inverso a su prioridad y τ_3 accede a M antes de activar τ_2 y τ_1, entonces se puede producir inversión de prioridades (figura 4.1).

La inversión de prioridad que se produce cuando τ_3 bloquea a τ_1, no se puede evitar si se quiere mantener la consistencia de los datos. En cualquier caso, no es muy grave si el tiempo de ejecución del servicio es pequeño. El bloqueo realmente grave es el que provoca τ_2, pues su duración es tan grande como sea el tiempo de ejecución de τ_2.

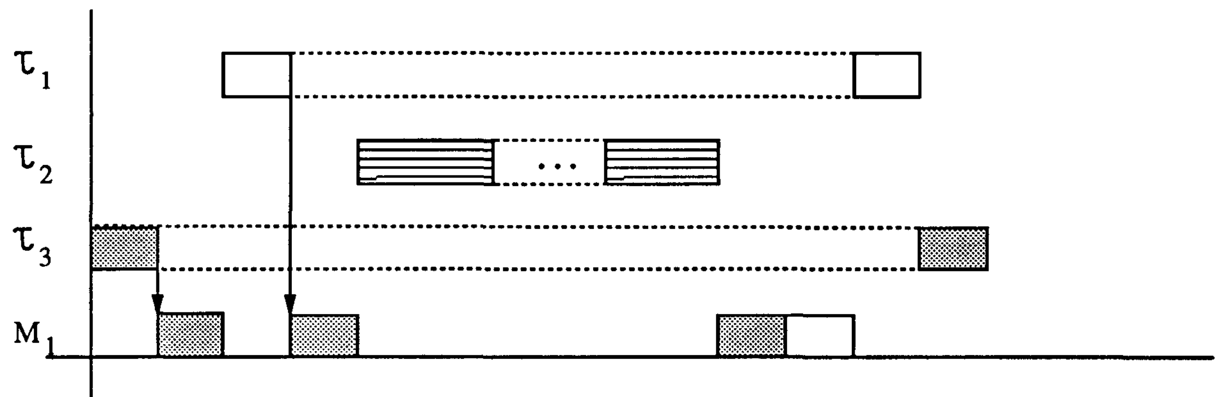

Figura 4.1: Inversión de prioridad.

Este panorama podría empeorar si hubiera un conjunto mayor de tareas preparadas para ejecutar con prioridades entre pr_1 y pr_3. τ_1 quedaría bloqueado durante la ejecución de todas ellas.

En este ejemplo se está suponiendo que las colas de espera del monitor están ordenadas por prioridades. Si no es así, se produce otra situación potencial de inversión de prioridades. En efecto, supóngase que son colas ordenadas según el orden de llegada (FIFO), que una tarea accede al monitor y en la cola hay una tarea con prioridad mayor esperando. Si una tarea mas prioritaria que ambas intentara acceder al monitor, debería esperar a que completaran el acceso la tarea que está en el monitor y la que está encolada.

4.3.2 Herencia de prioridades.

La herencia de prioridades [Sha&87] es la base del desarrollo de un conjunto de protocolos para tratar el problema de inversión de prioridades. La idea central es que si una tarea, τ', bloquea a otra tarea, τ, más prioritaria, entonces τ' hereda dinamicamente la prioridad de τ.

En el ejemplo de anterior 4.1, τ_3 heredaría la prioridad de τ_1 cuando ésta intenta acceder al monitor, M, que está ocupado por τ_3. En este caso, τ_2 no bloquea a τ_3 y τ_1 sólo estará bloqueada durante el tiempo que τ_3 use el monitor (figura 4.2).

Este método permite acotar la duración de la inversión de prioridades. En concreto, si una tarea accede a m monitores y hay n tareas con menor prioridad que usan éstos, el número máximo de veces que τ sufrirá inversión de prioridades en cada activación es $\min(n, m)$. La duración máxima del bloqueo en cada activación será la suma de los tiempos de bloqueo más desfavorables al acceder a los $\min(n, m)$ monitores.

Los bloqueos que acontecen cuando se emplea este protocolo provienen de dos causas:

- *Bloqueo directo*: Se produce cuando una tarea debe esperar a que otra tarea menos prioritaria libere un monitor.
- *Bloqueo transitivo*: Ocurre cuando una tarea con prioridad media está bloqueado

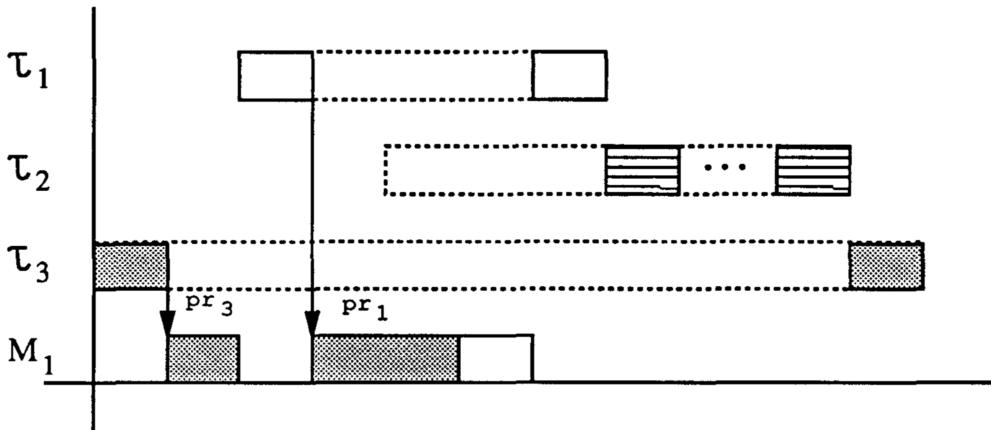

Figura 4.2: Protocolo de herencia de prioridades.

por una tarea con prioridad menor que ha heredado la prioridad de una tarea con prioridad alta.

El protocolo básico de herencia de prioridades tiene desventajas que dificultan su aplicación práctica:

- *Interbloqueos*. Este protocolo no evita los interbloqueos en el acceso a monitores. En efecto, sea τ_1 una tarea que accede al monitor M_2 y, antes de liberarlo, a M_1 y sea τ_2 una tarea menos prioritaria, que accede a los mismos monitores, pero en el orden inverso. En la figura 4.3 se muestra una secuencia de ejecución que producen un interbloqueo. τ_1 ha bloqueado M_2 e intenta acceder a M_1, mientras que τ_2 ha bloqueado M_1 e intenta acceder a M_2. Las dos tareas están esperando a que la otra libere el monitor correspondiente.

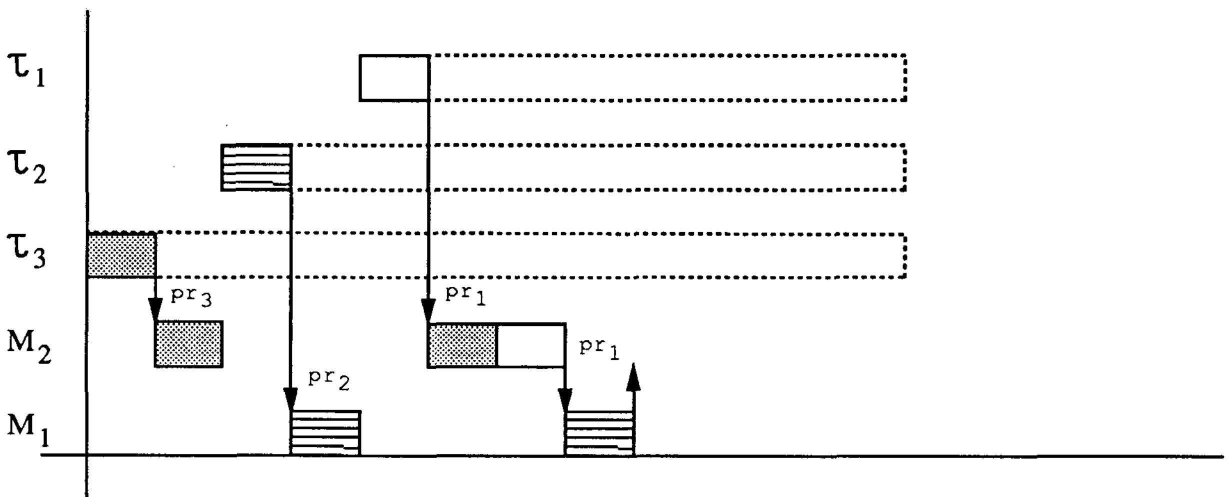

Figura 4.3: Interbloqueo al aplicar el protocolo de herencia de prioridades.

- *Bloqueos encadenados.* El caso más desfavorable de bloqueo de una tarea puede ser excesivamente grande. Esto es debido a la posible existencia de cadenas de bloqueos. En la figura 4.4 se muestra un ejemplo de esta situación. La tarea τ_1 comparte el monitor M_1 con τ_2 y el monitor M_2 con τ_3. Si cuando se activa τ_1 los dos monitores están ocupados, τ_1 estará bloqueada por inversión de prioridades durante la suma de los tiempos de acceso de τ_2 y τ_3 a los monitores correspondientes.

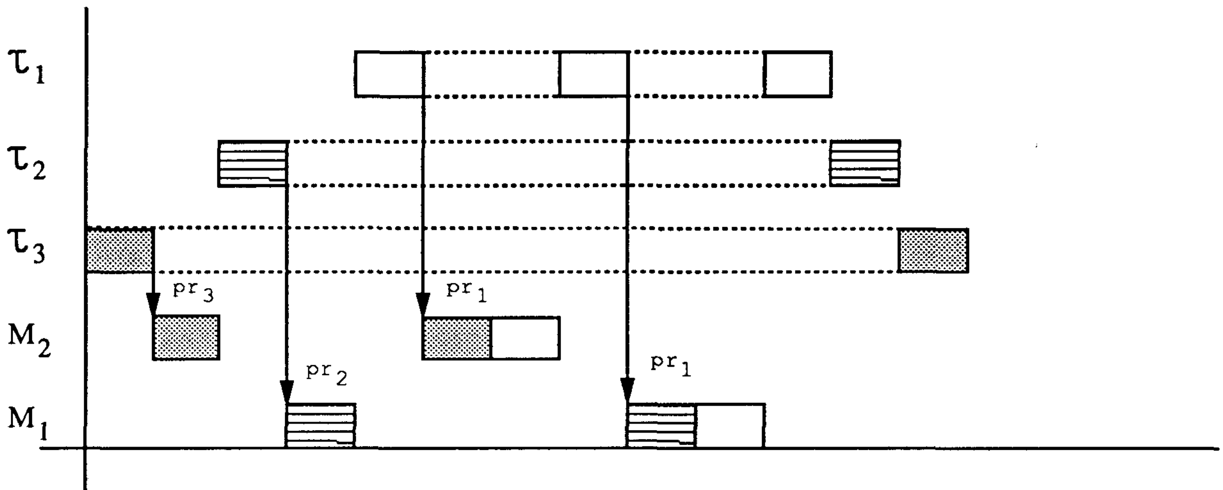

Figura 4.4: Interbloqueo al aplicar el protocolo de herencia de prioridades.

4.3.3 Protocolo del techo de prioridad.

El problema básico del método anterior radica en que los monitores se conceden a las tareas, sin tener en cuenta la relación con prioridades de otras tareas que han bloqueado o pueden intentar acceder a los monitores. El protocolo del techo de prioridad [Sha&87] [Goodenough&88] [Rajkumar91] no concede acceso a un monitor si esta acción puede provocar interbloqueos o bloqueos encadenados.

El techo de prioridad de un monitor es la **prioridad máxima de sus clientes potenciales**. El protocolo del techo de prioridad se define como sigue:

1. Sea τ una tarea en ejecución y sea $M_>$ el monitor con la prioridad mayor entre todos los monitores que están ocupados por tareas distintas de τ. τ intenta acceder a un monitor M. La tarea τ quedará bloqueado y la petición de acceso no tendrá efecto, si la prioridad de τ no es mayor que el techo de prioridad del monitor $M_>$. En este caso se dice que τ ha sido bloqueada por $M_>$. Cuando éste sea liberado, todas las tareas bloqueadas en el monitor pasarán a estar preparadas para ejecutar, y se ejecutará la más prioritaria.

2. Una tarea τ_c cuando está dentro de un monitor M ejecutará con su prioridad, a no ser que bloquee a tareas más prioritarias. En este caso, heredará la prioridad mayor

de las tareas a las que bloquea. Cuando libere el monitor, recuperará su prioridad original. La herencia de las prioridades es transitiva. Las operaciones de herencia de prioridad y de recuperación de la prioridad original deben ser atómicas.

3. Una tarea cuando no accede a un monitor expulsa a cualquier tarea con prioridad, propia o heredada, menor.

El protocolo del techo de prioridad genera un nuevo tipo de bloqueo, además del directo y transitivo presentados en el apartado anterior. Es el *bloqueo indirecto* y se produce cuando una tarea queda bloqueada porque su prioridad es menor que el techo de prioridad de algún monitor bloqueado. Sin embargo, el caso peor de bloqueo para cada tarea se reduce considerablemente.

Las propiedades más importantes de este protocolo son [Sha&91a] [Rajkumar91]:

- *El protocolo del techo de prioridad evita los interbloqueos.* En la figura 4.3 se mostraba un ejemplo de un interbloqueo cuando se comunican tareas mediante el protocolo de herencia de prioridad. Si en este ejemplo se aplica el protocolo del techo de prioridad, no se producirá interbloqueo (figura 4.5). En efecto, cuando τ_2 intenta acceder al monitor M_2 se bloqueará porque su prioridad es menor que el techo de prioridad del monitor bloqueado M_1, pr_1.

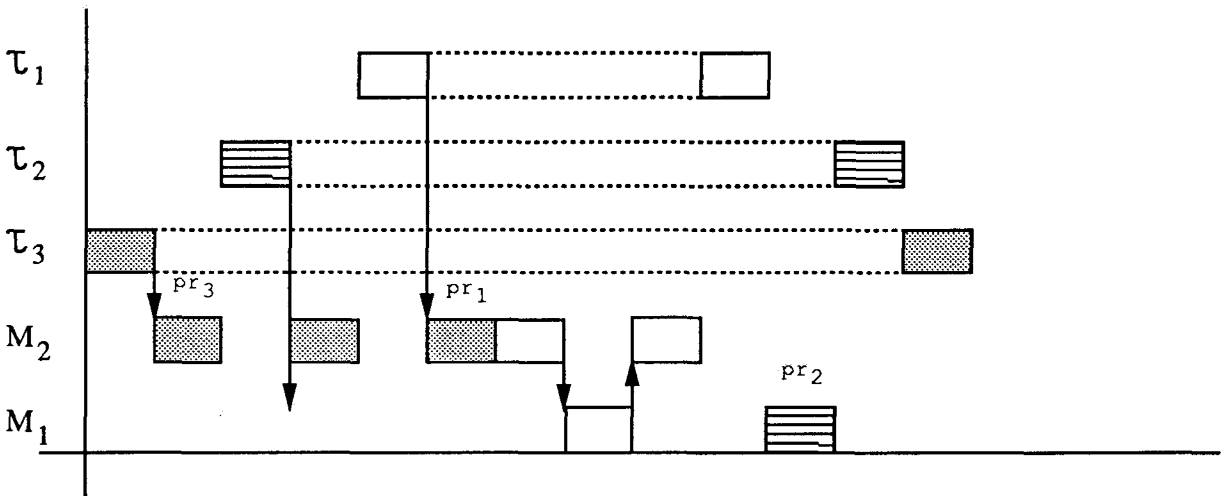

Figura 4.5: Protocolo del techo de prioridad. Prevención de interbloqueos

- El tiempo máximo de bloqueo por inversión de prioridades en una activación de una tarea que se comunique con otras mediante este protocolo, será la mayor duración del acceso de una tarea con menor prioridad a un monitor con techo de prioridad mayor o igual que la prioridad de la tarea. Esta reducción en el tiempo máximo de bloqueo se justifica porque el protocolo del techo de prioridad evita los bloqueos encadenados.

  En el ejemplo ilustrado por la figura 4.4, el techo de prioridad de los servidores M_1 y M_2 será la prioridad de pr_1. Si se aplica el protocolo del techo de prioridad a este

ejemplo (figura 4.6), se observa que no se producen bloqueos encadenados porque a τ_2 no se le permite acceder al monitor M_2 mientras que M_1 esté bloqueado, dado que el techo de prioridad de este monitor, pr_1, es mayor que pr_2. En este caso, el tiempo máximo de bloqueo de τ_1 es la duración del acceso τ_3 a M_2.

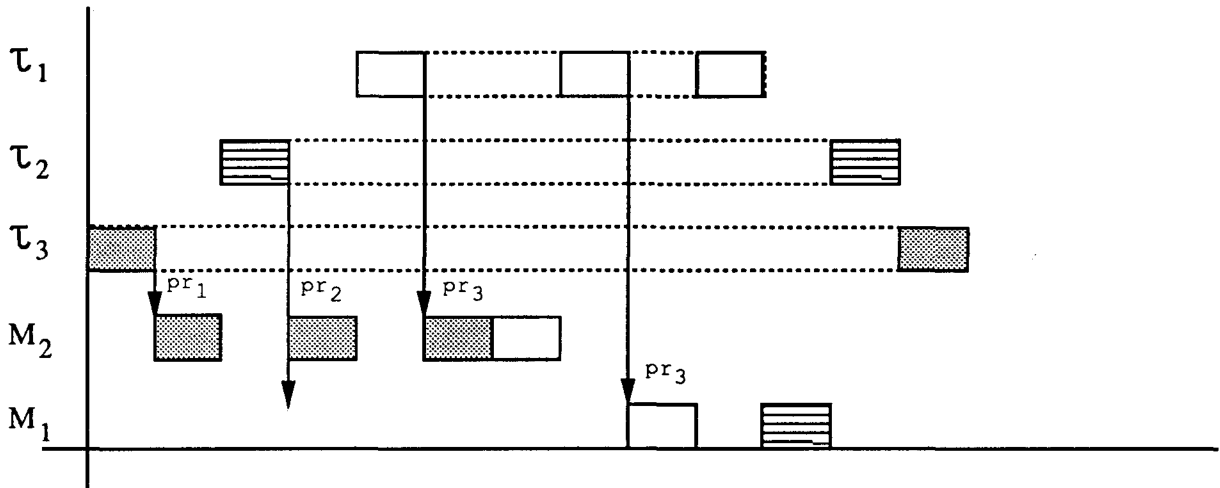

Figura 4.6: Protocolo del techo de prioridad. Prevención de bloqueos encadenados

- Una tarea sólo se puede bloquear por inversión de prioridades en cada activación la primera vez que accede a un monitor.

4.3.4 Protocolo de herencia inmediata del techo de prioridad.

El protocolo de herencia inmediata del techo de prioridad [Sha&89a] [Audsley91b] [Klein&90] es una variante del protocolo del techo de prioridad. Cuando una tarea accede a un monitor, hereda inmediatamente el techo de prioridad del monitor, sin necesidad de que bloquee a otras tareas más prioritarias. Este protocolo también impide bloqueos múltiples e interbloqueos y el caso más desfavorable de bloqueo por inversión de prioridad es el mismo. Sin embargo, las condiciones en las que se niega acceso de una tarea a un monitor son más restrictivas. El número de veces que una tarea será bloqueada por este motivo será mayor.

Esta variante del protocolo de herencia del techo de prioridad es la más útil desde el punto de vista de su empleo en aplicaciones prácticas. Las normas IEEE POSIX.1c [POSIX.1c 93] y Ada9X [Ada9xMap92] [Ada9XMaS92] [Ada9XAnS92] proporcionan mecanismos de comunicación basados en este enfoque. Sus ventajas son:

- Su implantación práctica es muy sencilla. La única operación necesaria es elevar la prioridad de una tarea cuando accede a un monitor, sin necesidad de comprobaciones adicionales.

- En sistemas monoprocesadores, si se emplea este protocolo, este esquema de prioridades es suficiente para asegurar la exclusión mutua en el acceso a las regiones

críticas.

- Una ventaja adicional, consiste en que el número de cambios de contexto que origina es menor que en el caso anterior.

4.3.5 Protocolo de control de semáforos.

El protocolo del techo de prioridad define condiciones suficientes para evitar bloqueos encadenados e interbloqueos. Sin embargo, en algunas situaciones es innecesariamente restrictivo. El protocolo de control de semáforos [1] [Rajkumar&88] [Rajkumar91] relaja las condiciones en que se impide el acceso a un monitor libre.

La definición del protocolo es como sigue:

Sea una tarea, τ_i que quiere acceder al monitor M. Si M está bloqueado por una tarea τ_m, ésta heredará la prioridad de τ_i. Si M no está bloqueado, τ_i podrá acceder a M si y sólo si se cumple alguna de las siguientes condiciones:

1. Si la prioridad de τ_i es mayor que el techo de prioridad mayor entre los monitores bloqueados.
2. Si la prioridad de τ_i es igual que el techo de prioridad mayor entre los monitores bloqueados y τ no necesita usar ninguno de éstos.
3. Si la prioridad de τ_i es igual que el techo de prioridad mayor entre los monitores bloqueados y las tareas que están accediendo a monitores con techo de prioridad mayor, no acceden a M.

El caso peor de bloqueo por inversión de prioridades es el mismo que con los protocolos anteriores. Sin embargo, las tareas serán bloqueadas menos veces, como consecuencia de las condiciones menos restrictivas de bloqueo indirecto de este algoritmo.

Una desventaja de este método es su dificultad de implementación y el coste de las comprobaciones en tiempo de ejecución. Por esta razón no se suele usar en aplicaciones reales.

4.3.6 Análisis de la planificabilidad.

Los métodos para analizar la planificabilidad de un conjunto de tareas propuestos en las secciones 4.1.1 y 4.2.3, suponían que no había comunicación entre tareas. Para enunciar métodos correspondientes adecuados, se define el tiempo de bloqueo por inversión de prioridades, B_i, como el tiempo máximo durante el que cualquier tarea menos prioritaria puede bloquear a una tarea con prioridad igual o mayor que τ_i. El valor de B_i en los protocolos basados en el concepto del techo de prioridad es la duración mayor de la ejecución

[1] Esta es la traducción directa del nombre original del protocolo. Evidentemente se puede aplicar a cualquier método de comunicación de tareas

las regiones críticas de los monitores cuyo techo de prioridad es mayor que la prioridad de τ_i. Este valor incluye los diversos tipos de bloqueo.

Cabe señalar que el instante crítico para un conjunto de tareas con comunicación no es exactamente el enunciado en el apartado 3.3.1, ya que no tiene en cuenta el factor de bloqueo. El enunciado correcto en este caso es:

Teorema 4.7 *Sea $\tau_1, \tau_2, \ldots \tau_n$ un conjunto de tareas ordenadas por prioridades decrecientes y que se comunican según el protocolo del techo de prioridad. El tiempo de respuesta más largo para cualquier tarea ocurre cuando se activan todas las tareas al mismo tiempo y la tarea menos prioritaria que produce el mayor tiempo de bloqueo comienza el acceso al monitor que produce éste.*

A partir de estas definiciones se pueden generalizar los métodos anteriores.

Teorema 4.8 (Sha, Rajkumar y Lehoczky) *Sea $\tau_1, \tau_2, \ldots \tau_n$ un conjunto de tareas ordenadas por prioridades decrecientes y que se comunican según el protocolo del techo de prioridad. El conjunto es planificable, para cualquier relación de fases, si:*

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + \frac{B_i}{T_i} \leq n(2^{1/n} - 1)$$

Teorema 4.9 (Sha, Rajkumar y Lehoczky) *Sea $\tau_1, \tau_2, \ldots \tau_n$ un conjunto de tareas ordenadas por prioridades decrecientes y que se comunican según el protocolo del techo de prioridad.*

Sea S_i el conjunto de los puntos de planificación para el conjunto de tareas dado, definido como sigue:

$$S_i = \left\{ k.T_j \mid j = 1, \ldots, i; \; k = 1, \ldots, \left\lfloor \frac{T_i}{T_j} \right\rfloor \right\}$$

τ_i cumplirá todos sus plazos de respuesta si y sólo si:

$$\min_{t \in S_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil + \frac{B_i}{T_i} \leq 1$$

El conjunto de tareas es planificable, para cualquier relación de fases, si y sólo si:

$$\max_{1 \leq i \leq n} \; \min_{t \in S_i} \left(\sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil + \frac{B_i}{T_i} \right) \leq 1$$

En el contexto del método de planificación de prioridad a la tarea más urgente [Burns&91] se han enunciado los siguientes teoremas:

Teorema 4.10 (Burns y Wellings) *Sea $\tau_1, \tau_2, \dots \tau_n$ un conjunto de tareas ordenadas por prioridades decrecientes y que se comunican según el protocolo del techo de prioridad. El conjunto es planificable, para cualquier relación de fases, si:*

$$\forall i : 1 \leq i \leq n :$$

$$\frac{C_i}{D_i} + \frac{I_i}{D_i} + \frac{B_i}{D_i} \leq 1$$

donde B_n y I_1 valen 0 y para $i > 1$

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{T_j} \right\rceil C_j$$

Teorema 4.11 (Burns y Wellings) *Sea $\tau_1, \tau_2, \dots \tau_n$ un conjunto de tareas ordenadas por prioridades decrecientes prioridades asignadas y que se comunican según el protocolo del techo de prioridad. El conjunto es planificable, para cualquier relación de fases, si y sólo si todas las tareas son planificables según el algoritmo que se presenta a continuación. Esta prueba se debe repetir hasta que se compruebe que la tarea cumple o no el primer plazo de respuesta:*

$$\frac{I_i^{t_0}}{t_0} + \frac{C_i}{t_0} + \frac{B_i}{t_0} \leq 1$$

$$donde \;\; t_0 = \sum_{j=1}^{i} C_j + B_i$$

$$\frac{I_i^{t_1}}{t_1} + \frac{C_i}{t_1} + \frac{B_i}{t_1} \leq 1$$

$$donde \;\; t_1 = I_i^{t_0} + C_i + B_i$$

$$\dots$$

$$\frac{I_i^{t_k}}{t_k} + \frac{C_i}{t_k} + \frac{B_i}{t_k} \leq 1$$

$$donde \;\; t_k = I_i^{t_{k-1}} + C_i + B_i$$

y donde

$$I_y^x = \sum_{z=1}^{y-1} \left\lceil \frac{x}{T_z} \right\rceil C_z$$

4.3.7 Conclusiones.

El conjunto de protocolos que se ha presentado permiten tratar el problema de inversión de prioridades. En particular, acotan la duración del bloqueo que sufre una tarea y el valor de la cota sólo depende de la duración de la operación. El efecto de este bloqueo se puede incluir en los métodos de análisis de planificabilidad presentados.

Entre los protocolos que se han descrito, el de herencia inmediata del techo de prioridad es el más interesante. Esto se debe a que el caso más desfavorable es el mismo que el óptimo y su coste de implantación y de ejecución es menor que en los otros casos.

4.4 Tareas aperiódicas.

4.4.1 Planteamiento.

La definición original del algoritmo de planificación de prioridad a la tarea más frecuente [Liu&73], no contemplaba la ejecución de tareas aperiódicas. Estas tareas tratan eventos que ocurren aleatoriamente en el tiempo. El problema es cómo tratar estos eventos de forma que las tareas aperiódicas cumplan sus restricciones temporales, sin interferir en el cumplimiento de los plazos de respuesta de las tareas periódicas. En efecto, si se trata un evento inmediatamente después de ocurrir, se puede impedir que una tarea periódica cumpla sus restricciones temporales.

Este problema ha sido estudiado en profundidad, en especial en el *Software Engineering Institute* de la Universidad de Carnegie-Mellon, por la necesidad de disponer de tareas aperiódicas para tratar eventos aleatorios. La mayoría de los mecanismos descritos en la literatura sólo permiten implementar tareas aperiódicas. Algunos de ellos ofrecen buenos tiempos de respuesta en el tratamiento de los eventos y no interfieren con la ejecución de las tareas periódicas. Sin embargo, no garantizan el cumplimiento de plazos de respuesta de las tareas esporádicas. Entre estos métodos destacan los protocolos de segundo plano, por consulta, servidor diferido e intercambio de prioridades[Lehoczky&87] [Sprunt&88].

4.4.2 Protocolo del servidor esporádico.

El protocolo del servidor esporádico [Sprunt&89] [González&91] es el único que permite realizar tareas esporádicas, es decir, tareas aperiódicas con restricciones críticas de tiempo. Este protocolo se basa en crear tareas servidoras, a las que se les asigna un tiempo de cómputo máximo durante un cierto intervalo, para tratar eventos aperiódicos y en definir un mecanismo de restauración del tiempo de cómputo consumido. Al intervalo de tiempo respecto al que se le define el tiempo de cómputo máximo se le llama *período del servidor*. Esta tarea preserva su tiempo de cómputo con la prioridad original hasta que aparece una petición aperiódica y sólo se restaura tiempo de cómputo cuando se ha consumido todo o en parte.

A continuación se presentan algunos términos que se utilizarán para explicar formal-

mente este protocolo:

np_s: Nivel de prioridad en el que el sistema está ejecutando una tarea en un instante dado.

np_i: Un nivel de prioridad del sistema. Los niveles de prioridad se numeran consecutivamente, siendo np_1 el nivel de prioridad menor.

Activo: Este término se utiliza para describir un intervalo de tiempo respecto a un nivel de prioridad. Un nivel de prioridad se considera activo, si la prioridad actual del sistema, np_s, es igual o mayor que la prioridad de np_i.

Ocioso: Este término tiene significado opuesto al de activo. Un nivel de prioridad se considera ocioso, si la prioridad actual del sistema, np_s, es menor que la prioridad de np_i.

RT_i: Instante de restauración del tiempo consumido por los servidores del nivel de prioridad np_i. Es el instante de tiempo cuando el tiempo de ejecución consumido por el servidor esporádico de prioridad np_i se restaura.

La restauración del tiempo de un servidor esporádico, que se ejecuta en el nivel de prioridad np_i, es como sigue:

- Si el servidor tiene tiempo de ejecución disponible, el instante de recuperación del tiempo consumido RT_i, se determina cuando el nivel de prioridad np_i pasa a activo. El valor RT_i será el instante de la transición del nivel de prioridad a activo más el período del servidor.

- Si la capacidad del servidor se ha terminado, el siguiente instante de recuperación RT_i se puede determinar cuando la capacidad del servidor sea mayor que cero y el nivel np_i esté activo.

- La recuperación de cualquier tiempo consumido por el servidor se planifica en el instante RT_i si el nivel de prioridad np_i está ocioso o el tiempo de ejecución del servidor se ha completado. La cantidad a recuperar es igual al tiempo de ejecución consumido.

El protocolo del servidor esporádico se puede emplear para diseñar tareas aperiódicas o esporádicas. Cuando se desarrollan tareas aperiódicas, se asignan varios eventos aperiódicos a un servidor esporádico. Mientras que haya tiempo de ejecución disponible, se atenderán las peticiones existentes. Cuando se acabe el tiempo de ejecución disponible, se parará la tarea aperiódica y cuando se reponga tiempo de cómputo, continuará su ejecución exactamente en el instante donde se interrumpió. El período del servidor y el tiempo de cómputo se asignan teniendo en cuenta los eventos que se tratan y el tiempo su tratamiento.

La aplicación del protocolo del servidor esporádico para desarrollar tareas esporádicas es más sencilla. Se crea un servidor esporádico por cada tarea esporádica. El período del

servidor coincide con la separación entre eventos de la tarea y el tiempo de cómputo es el mismo.

La asignación de prioridades a un servidor esporádico se hace de acuerdo a su período o a su plazo de respuesta, de forma análoga a una tarea periódica. A continuación se muestra un ejemplo de ejecución un servidor esporádico. Supóngase que se tienen las siguientes tareas:

Tarea	*Período*	*T. Cómputo*	*Prioridad*
$Tarea_1$	10	2	2
$Tarea_2$	14	6	1
$Servidor$	10	2	2

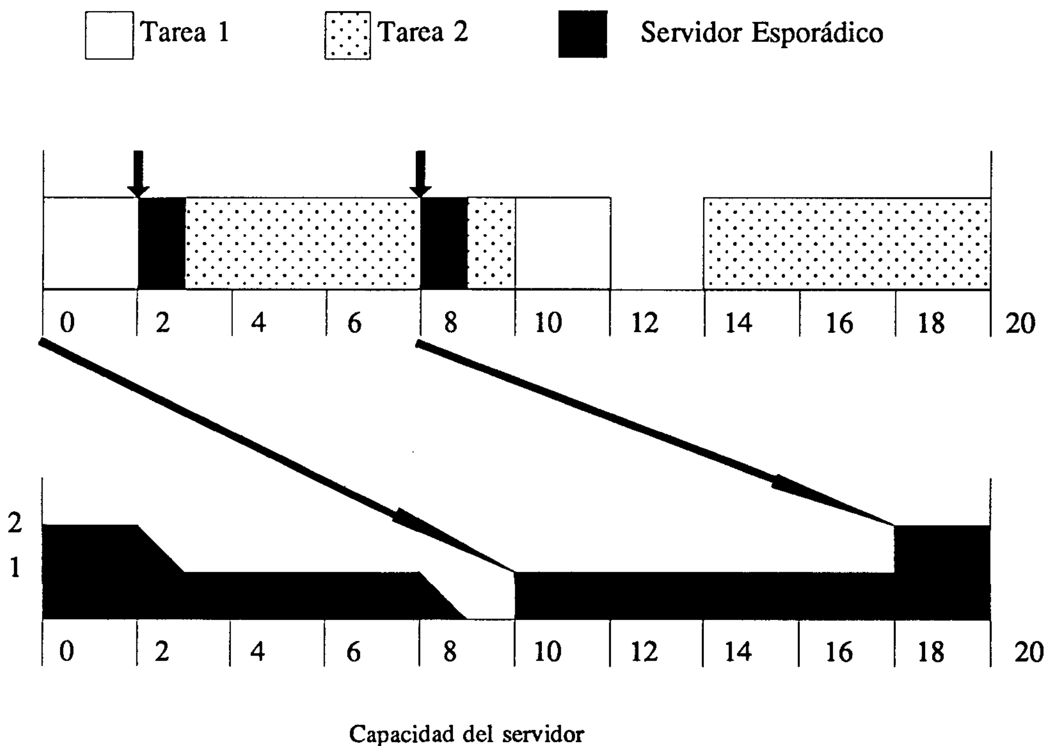

Figura 4.7: Ejecución de un servidor esporádico.

La ejecución de estos elementos se muestra en la figura 4.7. En ella se muestra la

ejecución de las tareas y la capacidad de cómputo disponible del servidor, en función de la aparición de eventos aperiódicos.

4.4.3 Análisis de la planificabilidad

El método de análisis de planificabilidad presentado en apartados anteriores supone únicamente la existencia de tareas periódicas. Es necesario determinar el impacto en este método de la inclusión de tareas aperiódicas realizadas según el algoritmo del servidor esporádico. A este respecto Sprunt, Sha y Lehoczky [Sprunt&89] han enunciado el siguiente teorema:

Teorema 4.12 (Sprunt, Sha y Lehoczky) *Un conjunto de tareas periódicas que es planificable con una tarea* τ_i*, es también planificable si se la sustituye por un servidor esporádico con el mismo período y tiempo de cómputo.*

La demostración del teorema se basa en determinar el caso más desfavorable del servidor esporádico; ocurre cuando se le solicitan servicios idénticos a los de una tarea periódica equivalente. Es decir, al comienzo de cada período del servidor, se solicita un servicio que requiere toda la capacidad de cómputo.

Por consiguiente, se puede aplicar el mismo análisis de planificabilidad cuando coexisten tareas periódicas y servidores esporádicos en un sistema de tiempo real.

4.5 Cambio de modo de ejecución del sistema.

4.5.1 Planteamiento.

En determinados sistemas de tiempo real, el entorno exterior es tan cambiante, que para adaptarse, puede ser necesario variar el conjunto de tareas que se ejecutan. Este es el caso en sistemas de aviónica donde las funciones que debe realizar el sistema en despegue o en vuelo estabilizado son diferentes. Un enfoque que se ha revelado válido, consiste en determinar un conjunto de modos de ejecución que se asocian a unas condiciones particulares del entorno. Para cada modo de ejecución se definen un conjunto de tareas con unas características específicas. Finalmente, es necesario un mecanismo que permita cambiar el modo de ejecución del sistema cuando se detecte que han variado significativamente las condiciones del entorno.

El cambio de modo de ejecución también es útil para situaciones en las que se producen errores de ejecución en el sistema. En este caso, esta operación permite degradar la funcionalidad del sistema de forma controlada y preconcebida.

Los objetivos básicos de un protocolo de cambio de modo son:

- *Integridad de los datos del sistema.* Un cambio de modo se puede producir en cualquier instante. Se debe asegurar que cualquiera que sean estas circunstancias, la integridad de los datos se mantendrá.

- *Cumplimiento de los plazos de ejecución de las tareas.* Es evidente que en cualquier modo de ejecución se debe garantizar el cumplimiento de los plazos de ejecución. El proceso de cambio de modo supone un estado transitorio del sistema en el que no se sabe con certeza las características de las tareas en ejecución en un momento dado. Puede ocurrir que como unas tareas aumentan su ocupación del procesador y otras disminuyen, el conjunto de tareas no sea planificable y se produzcan incumplimientos de los plazos de respuesta. Un método de cambio de modo útil debe permitir analizar el comportamiento del sistema, para garantizar la planificabilidad de las tareas.

Los requisitos de inmediatez de los cambios de modo resultan en dos tipos de enfoques a su implementación:

- *Asíncrono:* Cuando se produce un cambio de modo, se abortan todas las tareas cuyas características de ejecución cambian y se activa la versión de la tarea con las características correspondientes al nuevo modo. Esta enfoque se utiliza en sistemas muy específicos donde la necesidad de disponer de la funcionalidad del modo nuevo es crítica. Una desventaja evidente es la necesidad de que cada tarea incluya código específico para garantizar la consistencia de los datos.

  El sistema MARS [Reisinger89] proporciona esta funcionalidad y la transferencia asíncrona de control en Ada9x [Ada9xMap92] y las señales en POSIX.1b [POSIX.1b 93], permiten implementarla.

- *Síncrono:* Cuando se produce un cambio de modo, se deja que las versiones antiguas de las tareas terminen su ejecución con normalidad. Según van terminando, se activan las nuevas asegurando siempre que la carga del procesador sea tal que las tareas cumplan sus plazos de respuesta. Este enfoque complica el protocolo de cambio de modo y aumenta la duración de la operación. Sin embargo, la programación es más sencilla, al no tener que incluir código específico para salvaguardar la consistencia de los datos, en caso de cambio de modo.

  Se han desarrollado protocolos de cambio de modo para su aplicación con métodos de planificación basados en prioridades estática. La mayor sencillez del método, desde el punto de vista de la programación de la aplicación, su validez para la mayoría de las aplicaciones y su compatibilidad con los métodos de planificación basados en prioridades estática, aconseja su selección para profundizar en su estudio y aplicación.

4.5.2 Protocolo básico de cambio de modo

El cambio de modo es una técnica que se ha practicado desde hace tiempo en sistemas basados en el ejecutivo cíclico. Todo lo que hay que hacer es cambiar el plan principal al final de un ciclo mayor, si el cambio de modo es síncrono o inmediatamente, si es asíncrono. En el *Software Engineering Institute* de la Universidad de Carnegie-Mellon se ha desarrollado un protocolo de cambio de modo con funcionalidad y prestaciones

comparables con el ejecutivo cíclico. En [Sha&89c] [Sha&91a] se presenta el primer protocolo de cambio de modo compatible con la teoría de planificación de prioridad a la tarea más frecuente y a las ampliaciones anteriores. Este método establece unas normas concretas sobre cuando se pueden activar las tareas del nuevo modo y como se debe tratar el techo de prioridad de los monitores. Su enunciado es como sigue:

1. La adición o borrado de tareas en el cambio de modo, puede suponer modificar el techo de prioridad de algunos monitores. Al iniciarse el cambio de modo:
   - Para cada monitor no bloqueado, cuyo techo de prioridad se debe elevar, se eleva inmediata e indivisiblemente.
   - Para cada monitor bloqueado, cuyo techo de prioridad se debe elevar, se eleva inmediata e indivisiblemente después de ser desbloqueado.
   - Para cada monitor cuyo techo de prioridad se debe disminuir, se disminuye cuando todas las tareas que pueden bloquear M y que tienen prioridades mayor que el nuevo techo de prioridad, hayan sido borradas.
2. Una tarea τ que hay que borrar, se puede borrar inmediatamente después del inicio de cambio de modo, si τ no ha iniciado aún la ejecución en el período actual. Además, la capacidad de cómputo de τ, se puede reclamar inmediatamente. Si, por el contrario, τ hubiera iniciado la ejecución, se podría borrar después de finalizar su ejecución y antes del siguiente instante de activación. La capacidad de procesamiento asociado a τ no se podrá reclamar hasta la siguiente activación.
3. Una tarea, τ se puede añadir al sistema, si se cumplen las siguientes condiciones:
   - Si la prioridad de τ es mayor que el techo de prioridad de los monitores bloqueados $M_1, M_2, \ldots, M_n$, entonces el techo de prioridad de $M_1, M_2, \ldots, M_n$ se debe elevar antes de añadir τ.
   - Hay capacidad de procesador suficiente para añadir τ.

4.5.3 Determinación precisa de los tiempos de respuesta.

Un estudio posterior [Tindell&92a] presenta un contraejemplo que invalida parte de esta propuesta. En particular, se demuestra que no es suficiente que el análisis de planificabilidad garantice el conjunto de tareas activo en cada modo. En este trabajo se presentan en detalle los métodos analíticos para calcular los tiempos de respuesta más desfavorables de las tareas durante un cambio de modo. El análisis detallado del comportamiento de las tareas durante un cambio de modo es complicado y las fórmulas resultantes son poco intuitivas. Por esta razón se han incluido un resumen de este estudio en el apéndice A.

El resultado más interesante de este trabajo es que proporciona un método preciso para calcular el tiempo de respuesta más desfavorable de las tareas del modo previo, cuando durante un cambio de modo se activan otras nuevas. De esta forma se puede determinar el instante en que se deben activar las tareas que cambian sus características o se inician en

el modo nuevo, garantizando el cumplimiento del último plazo de respuestas de las tareas activadas antes del cambio.

4.5.4 Comunicación entre tareas.

La comprobación de las condiciones en que se debe cambiar el techo de prioridad durante un cambio de modo son costosas. Por esta razón, en [Tindell&92a] y en realizaciones prácticas de este protocolo [Alonso&92a] se supone que la comunicación entre tareas se realiza según el protocolo de herencia inmediata de prioridades. y el techo de prioridad de un monitor es la prioridad mayor entre las tareas cliente en cualquier modo. Este enfoque simplifica el protocolo de cambio de modo, sin suponer desventajas considerables.

La forma de incluir en los análisis presentados el bloqueo por inversión de prioridades, sería, al igual que en los análisis de planficabilidad en ejecución normal, añadir al tiempo de cómputo máximo de la tarea bajo análisis el factor B_i. Esta variable se definió como el tiempo máximo en que cualquier tarea menos prioritaria puede bloquear a una tarea con prioridad igual o mayor que τ_i.

4.5.5 Conclusión

El cambio de modo es un paradigma útil para adaptar la ejecución del sistema a las situaciones cambiantes del entorno y para tratar convenientemente los fallos de ejecución errores. El profundo estudio realizado por [Tindell&92a] permite analizar con precisión el comportamiento de un conjunto de tareas en presencia de un cambio de modo. Aunque la ecuaciones presentadas no son intuitivas, proporcionan la base para desarrollar una herramienta que permita determinar la planificabilidad del sistema en presencia de cambios de modo.

4.6 Otros estudios.

El interés de esta teoría de planificación ha generado una serie de trabajos que amplían su ámbito de aplicación. A continuación se reseñan algunos de los trabajos más interesantes, aunque no se presentan en detalles por referirse a situaciones muy específicas.

En [González&91a] se trata el problema de planificar un conjunto de tareas cuando la prioridad de cada tarea puede variar durante la ejecución del código correspondiente a una activación. Este caso ocurre cuando se tiene un sistema complejo en el que hay interrupciones, regiones críticas y otros mecanismos que producen un esquema complicado de prioridades. Las tareas se descomponen en subtareas que se ejecutan en serie y cada una de éstas se caracteriza por una prioridad y un tiempo de cómputo. En el artículo se propone un método para analizar la planificabilidad de las tareas.

Un método para planificar conjuntos de tareas con plazos de respuesta arbitrarios se presenta en [Lehoczky90]. El autor generaliza los límites del caso más desfavorable de ejecución de un conjunto de tareas cuando $D_i = T_i$ [Liu&73], en el sentido de permitir

plazos de respuesta del tipo $D_i = \Delta T_i$, para cualquier $\Delta > 0$. En particular, cuando se amplia el plazo de respuesta de las tareas en el valor del período ($\Delta = 2$), entonces el caso más desfavorable pasa de 0,693 a 0,811. El autor argumenta que este enfoque es especialmente útil en sistemas distribuidos en los que hay que computar el retardo por comunicación de los resultados.

En [Klein&90] se desarrolla un modelo matemático de la planificabilidad de un sistema de tiempo real considerando factores como el cambio de contexto, sincronización, interrupciones y otras cuestiones relacionadas con dispositivos de entrada/salida.

4.7 Conclusiones.

En este capítulo se han presentado los trabajos más importantes relacionados con los métodos de planificación basados en prioridades. El resultado neto de estos esfuerzos de investigación es la suficiente madurez de los métodos, que ha resultado en su aplicación en sistemas industriales.

Sin embargo, hay aún cuestiones abiertas en relación con los métodos de planificación basados en prioridades:

- Tolerancia a fallos: los requisitos de seguridad de muchos sistemas de tiempo real implican la necesidad de disponer de mecanismos para detectar y recuperarse de fallos compatibles con el método de planificación.

- Reconfiguración dinámica: aunque el cambio de modo permite adaptarse a cambios en el entorno exterior, son necesarios mecanismos adicionales que aumenten la flexibilidad con la que realizar esta operación.

- Detalles de bajo nivel: la mayoría de los trabajos presentados no considera detalles, como el cálculo de los tiempos de cómputo de las tareas, contabilidad de los tiempos de cómputo de elementos que interrumpen al procesador (como temporizadores), etc.

- Realizaciones concretas: la posibilidad de realizar práctica y eficazmente los métodos teóricos enunciados aumenta su valía. El estudio de los problemas concretos y la definición de directrices para implementar estos métodos con lenguajes y sistemas operativos comerciales sería de mucha utilidad.

- Herramientas de ayuda: el tamaño de muchos sistemas de tiempo real dificultan su desarrollo con técnicas artesanales. El empleo de herramientas que permitan desarrollar sistemas basados en estos métodos, simplificarían el proceso y ayudarían a aumentar su implantación industrial.

En el resto de este trabajo se tratan algunas de estas cuestiones.

Capítulo 5

Sistemas de tiempo real distribuidos.

5.1 Introducción.

5.1.1 Sistemas distribuidos de control.

Las primeras aplicaciones de control eran centralizadas. Un procesador realizaba todas las operaciones de control, supervisión e interacción con el operador de planta. La disponibilidad de redes de comunicación y de procesadores con mejores prestaciones y precio han ido transformando este panorama. En la actualidad se emplean múltiples procesadores para realizar estas funciones. La integración de los sistemas involucrados en la operación de un proceso industrial, es una de las características más relevantes de los sistemas de control de la siguiente generación [Rodd&89].

Un ejemplo de esta tendencia son los sistemas de fabricación integrados por computador (*Computer Integrated Manufacturing, CIM*). Este término describe los sistemas de fabricación en los que todos los componentes de una fábrica están conectados, de forma que se puede mostrar a cualquier usuario en cualquier punto del sistema, información sobre todos los componentes y todos los subsistemas de control. Esta estructura permite integrar y coordinar los diversos subsistemas involucrados en el proceso de fabricación. éstos son sistemas de control distribuido, ya que el control se realiza mediante la colaboración de un conjunto de computadores. Por ejemplo, el subsistema de control de un robot que coloca piezas en una cinta transportadora, deberá detenerse si el controlador de ésta detecta algún problema en el funcionamiento de la cinta.

Los sistemas distribuidos de control tienen requisitos de tiempo globales, que los sistemas de tiempo real centralizados no pueden satisfacer, ya que sólo consideran plazos de respuesta locales y no contemplan la comunicación con otros procesadores. Algunos de estos requisitos de tiempo son:

- En determinadas situaciones el sistema debe responder de acuerdo a los requisitos de funcionamiento de la planta. En una situación de alarma, los diversos componentes del sistema deben realizar las funciones de tratamiento en un intervalo de tiempo determinado.

- La sincronización entre diversos componentes de la planta debe realizarse en un plazo de tiempo acotado. Si se quiere coordinar el comportamiento de un robot y un carro móvil, cada uno deberá conocer con precisión la posición del otro.
- Los datos deben reflejar el estado real de la planta. Los datos en un sistema de control tienen un tiempo de validez limitado. Para dar al operador una imagen correcta del estado de la planta, los datos se deben actualizar en un plazo de tiempo determinado por la dinámica del sistema.

Otros requisitos funcionales también aconsejan la utilización de sistemas distribuidos [Stankovic88a]:

- *Requisitos funcionales de complejidad creciente.* Las operaciones necesarias para realizar la funcionalidad requerida son cada vez más complejas. Por tanto, en determinadas aplicaciones la capacidad de cómputo necesaria no se puede obtener con un único procesador. La potencia de cómputo requerida la proporcionará un conjunto de computadores en cooperación.
- *Fiabilidad.* Los sistemas de control se emplean en sistemas en los que un fallo puede tener consecuencias desastrosas, desde el punto de vista económico y humano. La tasa de fallos que se les exige a estas aplicaciones es rigurosa. Si se distribuye el cómputo entre diferentes computadores y falla uno de ellos, el resto puede realizar acciones para recuperarse de este error y continuar la ejecución de la funcionalidad original de la aplicación.
- *Dinamicidad y adaptación.* Dada la capacidad de cómputo disponible, se pueden reasignar las tareas a los procesadores, para adaptar el sistema a cambios en las condiciones del entorno.

En conclusión, es necesario desarrollar métodos de diseño de sistemas distribuidos de tiempo real que contemplen la conexión de múltiples nodos y permitan garantizar requisitos de tiempo globales a varios procesadores.

5.1.2 Sistemas distribuidos de tiempo real.

Los espectaculares avances tecnológicos en el desarrollo de hardware durante los últimos años, están cambiando radicalmente los sistemas informáticos. La arquitectura clásica centralizado está dejando paso a los sistemas distribuidos. Estos sistemas están formados por un conjunto de procesadores conectados por un medio de comunicación de alta velocidad y que colaboran, para conseguir los fines del sistema.

El software no ha tenido un auge paralelo. La programación de estos sistemas es radicalmente diferente a la de sistemas centralizados, lo que ha impedido su uso generalizado. Los sistemas operativos distribuidos han sido objeto de estudio desde hace varios años y su implantación comercial masiva aún no ha tenido lugar. La definición de un sistema operativo distribuido es [Tanenbaum&85] [Mullender89]:

> Un sistema operativo distribuido es aquel que, desde el punto de vista del usuario, se comporta como un sistema operativo centralizado, pero se ejecuta en múltiples procesadores independientes.

Parte de las dificultades del desarrollo de software para sistemas distribuidos provienen de la necesidad de gestionar un gran número de recursos. Otra razón, tanto o más importante, es la funcionalidad más compleja y ambiciosa requerida a estos sistemas. Algunas de estas características adicionales son [Tanenbaum92]:

- *Transparencia*: el usuario percibe el sistema como si fuera centralizado. No debe ser consciente de la complejidad de la arquitectura subyacente.
- *Flexibilidad*: debe ser posible modificar y reconfigurar el sistema, en función de la evolución de las necesidades y de la funcionalidad requerida.
- *Fiabilidad*: el sistema debe seguir funcionando, aún en presencia de fallos. La existencia de múltiples componentes hardware y la potencia de cómputo disponible, debe permitir que si un elemento falla, otros hagan su trabajo.
- *Altas prestaciones*: deben ser mayores que en un sistema centralizado y en consonancia a la potencia disponible.
- *Capacidad de crecimiento*: el sistema no debe colapsarse si se aumenta el número de procesadores conectados.

Los sistemas de tiempo real no son ajenos a este panorama. Sin embargo, al igual que ocurre en sistemas centralizados, el desarrollo de sistemas distribuidos de tiempo real es mucho más complejo que los de propósito general y su grado madurez es menor. Sin embargo, la distribución es una característica imprescindible para satisfacer las necesidades de la siguiente generación de sistemas de tiempo real.

El diseño de sistemas distribuidos de tiempo real, sin considerar requisitos como alto grado de tolerancia a fallos o de reconfiguración dinámica, plantea problemas esenciales nuevos, respecto a los sistemas centralizados:

- *Planificación de los canales de comunicación.* El medio de comunicación es un recurso que no puede estar dedicado. Con la tecnología actual, es utópico que cada tarea que se comunique con tareas remotas disponga un canal dedicado de comunicación.
- *Asignación de tareas a procesadores.* Hay que determinar qué tareas se deben ejecutar en cada procesador. Esta labor se debe realizar tratando de equilibrar la carga en los diversos procesadores, de minimizar la comunicación entre tareas remotas, tareas que se deben ejecutar en procesadores concretos, bien por disponibilidad de dispositivos bien por características específicas de éstos.

Durante los últimos años la investigación de sistemas operativos distribuidos de tiempo real ha sido intensa. Algunos de los trabajos desarrollados son: *ARTS* [Tokuda&89] [Mercer&90], *CHAOS* [Gopinath&89], *Alpha* [Northcutt87], *DRAGON SLAYER* [Wedde&89], *HARTOS* [Kandlur&89], *MARRUTI* [Levi&89], *Real-Time Mach* [Tokuda&90], ...

En el resto del capítulo se presentan tres enfoques diferentes y alternativos al desarrollo de sistemas distribuidos de tiempo real.

5.2 Sistemas síncronos.

5.2.1 Planteamiento.

Los sistemas síncronos se caracterizan porque el único evento que activa operaciones en el sistema es la interrupción de reloj. Los eventos asíncronos generados por líneas serie, buses externos, etc., son tratados mediante servidores de consulta. Este principio se extiende a la planificación del medio de comunicación. El acceso al medio físico se divide en rodajas y a cada tarea se le asignan estáticamente unas rodajas de tiempo específicas para enviar sus mensajes.

El mejor ejemplo de sistema distribuido de tiempo real síncrono es *MARS*. En el resto de este apartado se comentan los aspectos más relevantes de su diseño.

5.2.2 *MARS*.

Descripción general.

MARS (*Maintainable Real-Time System*) es una arquitectura de sistemas distribuidos, tolerantes a fallos y de tiempo real crítico, desarrollada en la Universidad Técnica de Viena. Su ámbito de uso son los sistemas industriales de control. Los objetivos principales del diseño del núcleo de ejecución de *MARS* [Damm&88] [Kopetz&89] [Reisinger89] [Kopetz&92] son garantizar el cumplimiento de los plazos de respuesta de las tareas y proporcionar un mecanismo de comunicación eficiente y adecuado para sistemas distribuidos de tiempo real. La característica distintiva de *MARS* respecto a otros sistemas de tiempo real es su comportamiento determinista en cualquier circunstancia, especialmente durante los picos de carga. Esto se debe al carácter síncrono de su concepción y realización.

El modelo de desarrollo se basa en el concepto de transacción para describir las actividades del sistema. En este contexto, una transacción es una secuencia de acciones interrelacionadas que cambian el estado del sistema. Un estímulo, el reloj, dispara una transacción y la correspondiente respuesta tiene que generarse en un intervalo de tiempo determinado.

Arquitectura del sistema.

Las configuraciones en las que ejecuta *MARS* están compuestas por un conjunto de nodos (*clusters* en su terminología) con alta conectividad interna. Cada nodo está compuesto por un conjunto de componentes interconectados mediante el *MARS-Bus* [Reisinger89]. Un componente es un computador autocontenido en el que se ejecutan las tareas críticas y acríticas que forman la aplicación y una copia del núcleo de *MARS*. Los componentes son homogéneos para facilitar el mantenimiento y la tolerancia a fallos.

Todos los componentes *MARS* tienen acceso a un reloj que tiene una precisión conocida [Kopetz&89b]. Una tarea de sincronización en cooperación con un circuito VLSI llamado CSU (*Clock Synchronization Unit*) mantienen el reloj del sistema. El tiempo se usa para razonar sobre la validez de la información de tiempo real, para detección de errores, para controlar el acceso al bus de tiempo real y para descartar la información redundante.

Gestión de interrupciones

La interrupción de reloj es la única que ocurre en el sistema. Los dispositivos externos no interrumpen, ya que interaccionan con el procesador mediante servidores de consulta. El manejador de la interrupción de reloj está compuesto por dos secciones que se ejecutan con frecuencia diferente:

- La primera sección se ejecuta cada milisegundo y suspende las llamadas al sistema. El manejador realiza operaciones como consultar el estado de los dispositivos, refrescar la memoria dinámica,...
- La segunda sección se ejecuta cada ocho milisegundos, después de la ejecución de la primera sección y no suspende las llamadas al sistema. Las operaciones principales que se realizan son la gestión del reloj del sistema, la planificación de tareas, el envío de los mensajes y la consulta de recepción de mensajes.

Comunicación en *MARS*.

La comunicación entre tareas se realiza mediante radiado (*broadcast*) de mensajes de estado. Estos mensajes se emplean para comunicar el estado de las tareas periódicamente, y tienen un tiempo de validez determinado. Los mensajes de estado se caracterizan por un identificador. Cada nodo sabe cuáles son los identificadores de los mensajes que tiene que leer. Dos mensajes con el mismo identificador o son redundantes o al segundo se le considera una nueva instancia del primero. Esta distinción dependerá de la marca de tiempo del mensaje.

Los nodos del sistema se comunican mediante una red de área local. El protocolo de comunicación es del tipo TDMA (*Time Division Multiple Access*). El acceso al medio físico se divide en rodajas de tiempo. Los mensajes de tiempo real se envían en rodajas de tiempo específicas. Esta planificación del medio de comunicación se realiza estáticamente, es decir, las tareas envían sus mensajes en instantes de tiempo predeterminados. Las

ventajas de este protocolo son su predecibilidad y determinismo. Las desventajas son la poca flexibilidad en el diseño y mantenimiento de aplicaciones y el desaprovechamiento de la red, cuando la carga es pequeña.

Planificación de recursos.

La planificación de tareas es estática y se realiza a partir sus tiempos de cómputo máximo, de los mensajes intercambiados y de la asignación de mensajes a las rodajas de tiempo del *MARS-Bus*. El resultado es un conjunto de tablas en las que se indica los instantes de comienzo y de finalización de cada una de las tareas. éstas se inician en el código de la segunda sección del manejador de interrupciones y deben ceder el procesador antes del instante de finalización. En caso contrario, no se habrá cumplido el plazo de respuesta y el sistema operativo tomará las acciones pertinentes para tratar el fallo.

El despachador de tareas se ejecuta durante la segunda sección del manejador de interrupciones. Este elemento consulta las tablas y cambia la tarea en ejecución en consecuencia. La llegada de mensajes está sincronizada con esta sección. Por tanto, es común que una tarea que espera un mensaje, se suspenda hasta este instante.

El sistema permite cambiar el modo de ejecución. La realización es muy sencilla. Basta con mantener una tabla por modo de ejecución y cambiar de tabla, cuando se solicita un cambio de modo. Este cambio se puede hacer inmediatamente después de la petición del cambio (asíncrono), lo que implica abortar las actividades en curso, o al final de un ciclo principal (síncrono).

Tolerancia a fallos

La detección de errores en *MARS* se comprueban en tres niveles:

- Detección de errores en el hardware: los componentes están diseñados para que se autocomprueben.
- Detección de errores en el sistema operativo: se emplean estructuras de datos robustas, aserciones y chequeos del tiempo de cómputo de rutinas del sistema.
- Detección de errores en el software de aplicación: las tareas críticas se ejecutan con redundancia.

Los componentes de *MARS* fallan silenciosamente, es decir, se desconectan cuando se detecta un fallo y no realizan actividad alguna. Los nodos del sistema detectan que hay nodos caidos, mediante un servicio de pertenencia [Kopetz&89a], que permite conocer el estado de otros componentes con un retardo menor que dos ciclos de TDMA.

La tolerancia a fallos en *MARS* [Kopetz&89a] [Kopetz&89c] se basa en la ejecución redundante de las tareas críticas y comparación de los resultados. Las ejecuciones redundantes están desplazadas en el tiempo, porque los autores consideran que así se detectan más errores que si estuvieran completamente sincronizadas.

Cuando se produce un fallo, pasa cierto tiempo hasta que se corrige. Si en este intervalo ocurre un segundo fallo, el sistema podría caerse. Para evitar esta posibilidad, se puede añadir un tercer componente, llamado *componente sombra*, que observa el comportamiento del sistema, pero no actúa. En caso de un fallo, tomaría el lugar del componente defectuoso.

5.2.3 Comentarios

El sistema operativo *MARS* es el más maduro de los sistemas operativos distribuidos de tiempo real. Su carácter síncrono simplifica algunos problemas de realización y asegura el determinismo de la ejecución y el cumplimiento de los plazos de respuesta, ya que todo los eventos ocurren en momentos predeterminados y fijos.

Sin embargo, está característica es, al mismo tiempo su mayor desventaja. La determinación de los tiempo de inicio de cada tarea, de los instantes en que cada mensaje se envía, etc, es una labor de gran complejidad. La única forma de desarrollar estos sistemas es con un conjunto de herramientas específicas, algunas de las cuales están en fase de desarrollo [Kopetz&91].

Consecuentemente, el mantenimiento de sistemas desarrollados sobre *MARS* es muy costoso. La modificación de alguna tarea en un procesador puede implicar el replanteamiento global de la aplicación, al tener que rediseñar las tablas de planificación y las tablas de transmisión de mensajes.

El sistema *MARS* es el único que se ha diseñado para ser tolerante a fallos. La consecuencia es que el porcentaje de fallos que se detectan y toleran es muy cercano al 100% [Kopetz&89c]. En este sentido, *MARS* es el más destacable entre los sistemas revisados.

5.3 Sistemas dinámicos.

5.3.1 Planteamiento.

La complejidad de los sistemas de tiempo real aumenta constantemente. Los sistemas de las siguiente generación se caracterizarán por [Stankovic88a]:

- serán sistemas distribuidos
- su comportamiento será muy dinámico y adaptativo.
- el tiempo de vida de los sistemas será muy grande.
- tendrán un comportamiento inteligente.
- su fallo tendrá consecuencias catastróficas.

Ejemplos de estos sistemas son los vehículos autónomos, sistemas inteligentes de fabricación, las estaciones espaciales, etc.

La tecnología de desarrollo de estos sistemas debe avanzar notablemente satisfacer estos requisitos. Uno de los aspectos más difíciles será demostrar que estos sistemas cumplen los requisitos de prestaciones, especialmente los relativos al cumplimiento de los plazos de respuesta. El carácter dinámico y adaptativo de estos sistemas requiere que esta comprobación se deba hacer en tiempo de ejecución, para determinar si tareas que se crean dinamicamente pueden ser garantizadas. En caso necesario, se debería migrar tareas con objeto de garantizar los plazos de aquéllas que en su procesador original no lo podían ser, o de equilibrar la carga.

5.3.2 *Spring*

El sistema operativo *Spring* pretende proporcionar parte de la funcionalidad básica requerida para cumplir estas expectativas [Stankovic&91]. El objetivo de este enfoque es desarrollar aplicaciones deterministas y, al mismo tiempo, flexibles. El sistema *Spring* está pensado para sistemas de tiempo real muy grandes y complejos.

El diseño resultante es una arquitectura distribuida en la que se separan las funciones críticas de la aplicación, de operaciones no deterministas que puedan interferir en el cumplimiento de los plazos de respuesta. El sistema operativo se basa en un planificador multinivel, muy dinámico y que, eventualmente, puede modificar determinados parámetros del sistema para adaptarse a cambios en el entorno.

Arquitectura del sistema

Springnet es un sistema físicamente distribuido, compuesto por una red de multiprocesadores, donde se ejecuta el núcleo *Spring* [Stankovic&91] [Stankovic90] [Stankovic&89] [Stankovic&87]. Cada multiprocesador está compuesto por:

- Uno o más procesadores de aplicación, donde se ejecutan tareas de alto nivel específicas de la aplicación y garantizadas a priori.

- Uno o más procesadores del sistema, encargados de ejecutar el algoritmo de planificación y otras operaciones específicas del sistema operativo. Estos componentes intentan eliminar parte de la sobrecarga del sistema operativo, para no interferir con las tareas de tiempo real.

- Subsistema de Entrada/Salida, gestiona las operaciones de entrada salida acríticas, dispositivos lentos y sensores rápidos. Nuevamente, se pretende que las interrupciones no interfieran con las tareas de la aplicación.

En relación a las comunicación entre nodos, no se detallan los mecanismos específicos que se emplean [Stankovic&91][Stankovic90]. Sin embargo, se propone el uso de múltiples buses para incrementar el paralelismo y las prestaciones y disponer de mecanismos de comunicación predecibles.

Planificación de recursos.

El planificador del núcleo *Spring* [Stankovic&91] [Zhao&87a] [Zhao&87b] [Ramamritham&89] [Ramamritham&84] considera los requisitos de procesador y otros recursos que requieren las tareas conjuntamente. Es un planificador dinámico y distribuido. El planificador local está basado en el algoritmo dinámico de prioridad al más urgente. Además, cuando el nodo recibe la petición de ejecutar una tarea, se analiza dinamicamente si se puede garantizar. Si no es así, se ejecuta la parte distribuida del planificador para tratar de encontrar un nodo donde se pueda garantizar.

Una tarea en *Spring* tiene un tamaño relativamente pequeño y su comportamiento es determinista. Surge como consecuencia de descomponer los procesos (entendidos como tareas más complejas) programados por el usuario. Las tareas adquieren los recursos necesarios al principio de su ejecución y los liberan al terminar. Este enfoque es válido por su reducido tamaño y permite evitar bloqueos al acceder a los recursos. Un grupo de tareas es un conjunto de tareas con requisitos de precedencia entre ellas y que comparten un plazo de respuesta global. El concepto de grupo de tareas está aún en investigación [Stankovic&91].

Los autores dividen las tareas en las siguientes categorías, según su criticidad:

- críticas: el incumplimiento de sus plazos de respuesta produce resultados catastró*ficos.*
- esenciales: son tareas con plazos de respuesta y su operación es importante para el funcionamiento del sistema. Sin embargo, el incumplimiento ocasional de los plazos de respuesta no provoca daños.
- no esenciales: ejecutan cuando no hay ninguna de las anteriores preparada.

Las tareas críticas se garantizan estáticamente y se les reserva los recursos necesarios *para su ejecución. Se ejecutan en los procesadores de aplicación para evitar interferen*cias del sistema operativo o de interrupciones externas. Las tareas esenciales se tratan dinamicamente, porque no se pueden reservar los recursos necesarios para prever todas las contingencias. En consecuencia, cada vez que se modifica la carga en un nodo, se comprueba la planificabilidad del conjunto y se intenta redistribuir la carga si no es posible garantizarlas.

El planificador resultante está compuesto por cuatro módulos:

- Despachador: se encarga simplemente de seleccionar la siguiente tarea a ejecutar de la tabla de tareas del sistema, en la que se incluyen todas las tareas garantizadas. Es el único módulo del planificador que se ejecuta en los procesadores de aplicación.
- Planificador local: se encarga de comprobar si localmente se puede garantizar una tarea nueva y, en su caso, incluirla en la tabla de tareas del sistema.
- Planificador global: intenta encontrar un nodo donde se pueda garantizar una tarea que no pueda serlo localmente.

- Controlador meta nivel: su misión es adaptar parámetros diversos o mecanismos de planificación a cambios importantes en el entorno.

El planificador local se debe emplear estáticamente para garantizar las tareas críticas y dinamicamente para comprobar si se pueden garantizar tareas nuevas. El planificador local considera el problema de planificar un conjunto de tareas en un sistema con un conjunto de recursos. Las tareas se caracterizan por su tiempo de cómputo, su plazo de respuesta, recursos requeridos para su ejecución y un tiempo de inicio, que es calculado por el planificador.

La planificación se plantea como un problema de búsqueda en un árbol. Cada nodo del árbol es un plan parcial, formado por un conjunto planificable de tareas con los recursos disponibles y los tiempos de inicio de éstas. Los descendientes de cada nodo se calculan añadiendo una tarea al conjunto. El nodo inicial es el plan vacío y los nodos terminales son nodos en los que se ha logrado un plan total o en los que no es posible garantizar más tareas. Se emplea una función heurística para decidir cuál es el recorrido con más garantías de éxito. Esta función sintetiza diversas características de las tareas que influyen en la planificabilidad del conjunto.

5.3.3 Comentarios

El sistema operativo *Spring* es el más innovador de los que se presentan en este capítulo. Sus diversos niveles de planificación permiten desarrollar aplicaciones dinámicas y adaptables al entorno donde se ejecutan. Los problemas de cumplimiento de requisitos temporales, gestión de recursos y asignación a procesadores se tratan de forma integrada. Esta aproximación es complicada y requiere el uso de herramientas específicas de análisis.

Sin embargo, es el más inmaduro. Aunque se proponen diversos niveles de planificación, no se concreta su comportamiento y los trabajos de investigación sobre los heurísticos adecuados no están finalizados.

Este sistema operativo no es tolerante a fallos. Los mecanismos para dotar a *Spring* de esta característica están en proceso de investigación. En cualquier caso, un alto nivel de tolerancia a fallos no es algo que se pueda añadir a posteriori. Lo que posiblemente implique un rediseño del núcleo y de la arquitectura.

Finalmente, es importante recordar que este sistema operativo está especialmente diseñado para sistemas de tiempo real grandes y complejos. La consecuencia es que la sobrecarga del sistema es importante, hasta el punto que se recomienda [Stankovic&91] emplear planificadores más eficientes, como el ejecutivo cíclico y métodos basados en prioridades, para tratar tareas con plazos de respuesta muy cortos.

5.4 Sistemas distribuidos basados en prioridades.

5.4.1 Planteamiento.

Un enfoque alternativo para el desarrollo de sistemas distribuidos de tiempo real consiste en extender el concepto de prioridad para la gestión de los recursos compartidos del sistema. En concreto, la competencia entre los nodos del sistema para acceder a un bus o una red de área local se resolvería a favor de la petición solicitada por la tarea más prioritaria.

En este apartado se van a describir dos enfoques de sistemas basados en prioridades, que se han desarrollado en dos centros de investigación. Uno de ellos proviene del *Software Engineering Institute* de la Universidad de Carnegie-Mellon y plantea una arquitectura muy general, que emplea componentes software y hardware comerciales. El otro tiene su origen en la Universidad de York. El sistema que define es menos general, pero plantea un método muy interesante para distribuir adecuadamente las tareas entre los procesadores.

5.4.2 Generalización de la planificación basada en prioridades.

Descripción general.

En el *Software Engineering Institute* de la Universidad de Carnegie-Mellon se han investigado intensamente los métodos de planificación basados en prioridades. En la actualidad, se están extendiendo estos métodos para desarrollar sistemas distribuidos [Sha&93]. El objetivo es generalizar las técnicas presentadas en el capítulo 4 para planificar los recursos compartidos del sistema. Una de las actividades realizada es la especificación de los requisitos de la arquitectura hardware y software para desarrollar sistemas distribuidos de tiempo real con este enfoque.

El método se basa en la asignación de prioridades a las tareas del sistema. Este concepto se debe extender a los protocolos de arbitraje de los buses, de paso de mensajes y de DMA. Si esta premisa no se cumple, entonces será inviable la aplicación del método, pues no se podrán acotar los tiempo de cómputo de las tareas y, consecuentemente, no se podrá garantizar el cumplimiento de los plazos de respuesta.

Arquitectura del sistema.

La arquitectura se basa en un conjunto de nodos conectados por una red de área local. Cada nodo es un multiprocesador conectado por un bus. Los procesadores de un nodo están compuestos por una CPU, la memoria y el sistema operativo.

Para aplicar esta teoría, los módulos deben solicitar el bus con la prioridad de la tarea que manda el mensaje. No todos los protocolos de arbitraje de buses permiten que un mismo módulo solicite simultáneamente el bus con diferentes prioridades. La norma IEEE Futurebus+ [Futurebus91] permite realizar esta funcionalidad. Una alternativa es el arbitraje basado en prioridades por software. El empleo de un árbitro centralizado o distribuido [Sha&91b] permite ocultar esquemas de arbitraje de buses existentes, con lo

que se consigue compatibilidad lógica con productos comerciales.

Una alternativa es la red de área local tipo *token ring* [IEEE 802.5]. Su funcionamiento se basa en un mensaje especial, llamado testigo, que circula por la red cuando las estaciones están ociosas. Cuando una estación quiere enviar un mensaje debe capturar antes el testigo. Este protocolo gestiona el envío de mensajes con prioridades. Básicamente este esquema consiste en que cuando una estación quiere enviar un mensaje, debe capturar un testigo cuya prioridad sea menor o igual que la del mensaje a enviar. Existe la posibilidad de reservar el testigo para la próxima emisión, cuando circula un mensaje con datos. En este caso, se escribe en éste la prioridad si es mayor o igual que la actual. El siguiente testigo se generará con la mayor prioridad reservada.

Se han propuesto diversos formas de emplear red de área local para garantizar los tiempos de envío de los mensajes. En [Strosnider&89] [Pleinevaux92] la red se planifica como el procesador. La competencia por el uso de la red se resuelve a favor de la estación que lo solicita con prioridad mayor. Las estaciones con igual prioridad se reparten el medio equitativamente. En esta situación, se pueden aplicar las técnicas presentadas en capítulos anteriores para asignar prioridad a los mensajes. Un problema de este enfoque es que, generalmente, el número de niveles de prioridad es insuficiente. Entonces aumenta la duración del caso más desfavorable respecto al caso en que el número de niveles es adecuado [Sha&86a][Sha&86].

Otra opción [Agrawal&92] consiste en que cada estación, cuando tiene en testigo, emite mensajes durante un período de tiempo asignado a priori. La capacidad de cada estación debe ser suficiente para enviar los mensajes críticos. La estructura de la red garantiza un retardo máximo entre dos recepciones consecutivas del testigo en una estación. Este es el método que se emplea en este trabajo. El medio de comunicación es una red tipo FDDI [Tanenbaum88]. En la sección siguiente se muestra cómo se puede realizar el análisis de planificabilidad en una red con estas características.

Planificación del sistema.

El objetivo principal de la arquitectura descrita es desacoplar la planificación de los diversos recursos. Se quiere analizar cada procesador de un nodo como si estuviera aislado y utilizar la misma técnica para planificar el paso de mensajes por el bus o por la red.

El análisis se realiza a varios niveles y por elementos, como si estuvieran aislados:

1. Análisis de plazos de respuesta de operaciones que implican tareas en diversos procesadores.

   En primer lugar se analiza la planificabilidad de operaciones que son realizadas por un conjunto de tareas con requisitos de precedencia. En este análisis se supone que hay capacidad suficiente en el medio y en cada uno de los procesadores. El tiempo de respuesta más desfavorable se calcula sumando:

   - los tiempos de respuesta más desfavorables de las ejecuciones de las tareas implicadas, que coincide con el período.

- los tiempos máximos de transmisión de los mensajes, que también es el período de la tarea emisora.

Si la suma de estos valores es menor que el plazo de respuesta del conjunto de la operación, entonces se podrá completar en el intervalo requerido. En caso contrario, habrá que modificar las características del conjunto de tareas o reducir los retardos en la comunicación, cambiando el procesador en que algunas de las tareas se ejecutan.

2. Planificación de los mensajes en la red.

   El análisis de la viabilidad del envío de los mensajes en sus plazos de respuesta, trata por separado los elementos software y hardware. En relación al software, se supone la existencia de una tarea por mensaje enviado, tanto en el nodo emisor como en el nodo receptor. Estas tareas se diseñan como servidores esporádicos con un período igual al de envío del mensaje, y el tiempo de cómputo máximo necesario para mandar o recibir los mensajes. En consecuencia, estas tareas se analizan como cualquier otra de la aplicación.

   Respecto al análisis de la capacidad del medio de comunicación, se emplea el método propuesto en [Agrawal&92]. Se debe comprobar que los nodos tienen el ancho de banda necesario, para enviar los mensajes en el tiempo asignado a cada nodo, en cada rotación del testigo. Consiguientemente, un parámetro importante del protocolo de comunicación es el tiempo de rotación del testigo, $TTRT$. Cada nodo puede transmitir durante un tiempo determinado, H_i, cada vez que recibe el testigo. Lógicamente se debe cumplir:

$$TTRT = \sum_{i=1}^{n} H_i + RR$$

   donde RR es la capacidad consumida en la rotación del testigo.

   Los mensajes críticos se mandan una vez en cada activación de la tarea correspondiente. Por consiguiente, necesitan ocupar el canal de comunicación una determinada cantidad de tiempo en cada activación. Este problema es completamente análogo al cálculo de la planificabilidad de un conjunto de tareas que consumen tiempo de cómputo en un procesador.

   En consecuencia, se tendrá un conjunto de mensajes, ME_i, que se deben enviar con una periodicidad determinada, T_i, y que cada uno ocupa el medio durante un tiempo máximo, C_i. Por lo tanto se tendrá un conjunto de mensajes, $(C_1, T_1), (C_2, T_2), ..., (C_n, T_n)$.

   La ocupación del medio por parte de otros nodos se puede modelar suponiendo que se tiene una tarea de alta prioridad, denominada de rotación del testigo, τ_{rt}. El tiempo de transmisión de esta tarea será $TTRT - H_i$ y el período $TTRT$. Entonces se puede analizar el conjunto empleado las técnicas propias de la teoría de planificación de prioridad a la tarea más urgente.

3. Planificación de tareas en un procesador.

   Se supone que se dispone de los recursos de comunicación necesarios y se estudia la planificabilidad de un conjunto de tareas en un procesador. Consecuentemente, se emplean los métodos presentados en el capítulo 4.

5.4.3 Sistema desarrollado en la Universidad de York.

Descripción general.

Los trabajos de investigación sobre sistemas distribuidos de tiempo real que se están desarrollando en la Universidad de York [Tindell&92a] [Audsley91b], siguen las mismas líneas maestras que el trabajo presentado en el apartado anterior. Es decir, separa el problema de planificación del procesador y planificación del medio de comunicación y se basa en métodos de planificación con prioridades. La novedad de este trabajo es el tratamiento del problema de reparto de las tareas en los procesadores del sistema. Parece lógico pensar que si se disocian los problemas de planificación de recursos y la asignación de tareas a los procesadores, será difícil conseguir soluciones óptimas, e incluso factibles en muchos casos.

El problema de reparto de carga no se trata convenientemente en la mayoría de los enfoques de desarrollo de sistemas distribuidos de tiempo real. Por ejemplo, en el sistema operativo *MARS* y en el sistema descrito en el apartado anterior, la planificación y el reparto de tareas son dos problemas independientes. En *Spring* los diversos niveles del planificador pueden emplearse para tratar estos problemas conjuntamente y conseguir repartos de cargas más racionales.

Planificación del sistema.

La arquitectura hardware donde se ejecuta este sistema, está constituida por un conjunto de procesadores conectados por una red que permite radiar mensajes. La red propuesta es de paso de testigo con un funcionamiento similar al del apartado anterior.

La planificación de las tareas en cada procesador está basada en prioridades. Las tareas siguen un esquema en el que envían sus mensajes al final de cada activación. Sus plazos de respuesta hacen referencia al instante en que estos mensajes llegan a su destino. Por tanto, se deben enviar antes del fin del plazo de respuesta, y el código de cada activación se debe completar d unidades de tiempo antes del plazo de respuesta global, donde d es el retardo máximo de la transmisión del mensaje.

En cuanto a la planificación de la red, el método que se emplea es análogo al presentado en el apartado anterior.

Configuración del sistema.

Los métodos presentados en el apartado anterior permiten analizar la planificabilidad de una configuración de tareas en procesadores determinada. Sin embargo, no proporciona

ningún método de determinar la configuración óptima. En el trabajo desarrollado en la Universidad de York, se propone un método para automatizar la configuración del sistema. Para este fin, hay que tener en cuenta otras consideraciones como:

- Algunas tareas deben residir en un subconjunto de los procesadores disponibles, por ejemplo, a causa de sensores que utiliza y que están fisicamente conectados a un procesador específico.
- Algunas tareas se replican por tolerancia a fallos y, consecuentemente, no pueden estar en el mismo procesador.
- La memoria que utilizan las tareas en un procesador está limitada

El método que se plantea se denomina recocido simulado (*simulated annealing*). Es una técnica de optimización global que trata de encontrar el punto mínimo en un paisaje de energía. Esta técnica ha sido aplicada con éxito a otros problemas conceptualmente similares [Laarhoven&87]. Una característica distintiva del algoritmo es que realiza saltos aleatorios a soluciones potenciales. Esta habilidad se controla y reduce según avanza la aplicación del algoritmo.

El conjunto de todas las posibles configuraciones de las tareas del sistema en los procesadores forman el espacio del problema. Un punto en el espacio del problema es una configuración concreta de tareas a procesadores. El espacio vecino está formado por los puntos del espacio que son alcanzables cambiando una tarea de un procesador a otro diferente. La energía de un punto es una medida de la adecuación de una configuración. La función de energía permite calcular la energía en un punto.

Se elige un punto de inicio, P_0, aleatorio
Se elige una temperatura inicial, Te_0, aleatoria
repetir
    repetir
        E_n := Energía en el punto P_n
        Se elige P_t, un vecino de P_n
        E_t := Energía en el punto P_t
        si $E_t < E_n$ **entonces**
            $P_{n+1} = P_t$
        si no
            $x := \frac{E_n - E_n}{Te_n}$
            si $e^x \geq$ random(0,1) **entonces**
                $P_{n+1} := P_t$
            si no
                $P_{n+1} := P_n$
            fin si
        fin si
    hasta equilibrio térmico
    $Te_{n+1} = F(Te_n)$

hasta algún criterio de parada

Tres funciones son necesarias para ejecutar este algoritmo:

- La función *vecino*, que se emplea para seleccionar un punto de energía vecino. Se pueden elegir funciones tan sencillas como seleccionar un procesador y una tarea aleatoriamente o que consideren posibilidades como intercambiar tareas entre procesadores.
- La función *energía*, que determina la energía de un punto. Esta función es más compleja y debe penalizar situaciones como:
  - Tareas asignadas a procesadores sin el hardware necesario.
  - Tareas replicadas asignadas al mismo procesador.
  - Procesadores en los que se utiliza más memoria que la disponible.
  - Tareas que no están garantizadas.
- La función de *enfriado* F, que disminuye progresivamente el factor de temperatura Te, para reducir los saltos aleatorios.

La evaluación de este método es difícil, porque no se puede comparar con el caso óptimo, ya que se desconoce especialmente en sistemas complejos. El método ha sido aplicado a diversos casos con éxito. Por ejemplo, se le suministró un caso complejo en el que las tareas se podían particionar en grupos altamente acoplados, con una energía óptima de valor cero. El algoritmo encontró la distribución de mínima energía, agrupó las tareas correctamente y las asignó al mismo procesador. Estas pruebas permiten suponer que el método de simulado recocido es adecuado para configurar sistemas distribuidos.

5.4.4 Comentarios

Los sistemas distribuidos de tiempo real basados en prioridades son un enfoque válido. El principio de tratar la planificación del procesador y de la red aisladamente es adecuado para tratar la complejidad intrínseca de este tipo de sistemas.

El mantenimiento de aplicaciones es menos costoso que en *MARS*. Es más rápido calcular una serie de inecuaciones a partir de los parámetros de las tareas, que tener que recalcular todas las tablas de planificación.

Una ventaja importante de esta aproximación es que no requiere núcleos de ejecución específicos y sofisticados. El método se puede aplicar con sistemas operativos y medios de comunicación disponibles comercialmente. Además, y como ya se ha indicado, las normas IEEE POSIX.1b, IEEE POSIX.1c, Futurebus+ y Ada9X incluyen características apropiadas para su aplicación, con las ventajas que ello comporta.

El método sistemático de asignación de tareas a procesadores permite resolver adecuadamente este problema. Desde el punto de vista del diseñador del sistema, su uso es muy

sencillo. Es suficiente describir cómo es una solución válida, sin tener que determinar cómo obtenerla.

El mayor inconveniente del método es la carencia de mecanismos para tolerancia a fallos, que es una característica importante en los sistemas de la siguiente generación. El empleo de servidores que se esboza en este trabajo podría ser útil para dotar a las aplicaciones de cierto grado de tolerancia a fallos. En particular, se podrían desarrollar servidores de tolerancia a fallos como base al empleo de técnicas de redundancia de tareas.

5.5 Consideraciones finales.

Los trabajos presentados demuestran que con la tecnología actual es posible desarrollar sistemas distribuidos de tiempo real. Sin embargo, no han alcanzado el nivel suficiente para que estas aplicaciones tengan todas las características deseables y necesarias de la siguiente generación.

El sistema operativo *MARS* y los métodos basados en prioridades proporcionan la funcionalidad suficiente para su aplicación industrial. Ambos métodos permiten garantizar el cumplimiento de los plazos de respuesta de las actividades del sistema. La tolerancia a fallos del primero y la flexibilidad del segundo son sus características más destacadas. El sistema operativo *MARS* es recomendable para aplicaciones críticas en cuanto a la seguridad y con un tamaño mediano. Los métodos basados en prioridades son adecuados para sistemas más complejos.

El sistema operativo *Spring* es el más inmaduro y aún falta tiempo para que constituya una base adecuada para el desarrollo de aplicaciones con las características esperadas. La dinamicidad de la planificación y configuración del sistema es la características más destacable de su diseño y debe ser una cualidad fundamental para sistemas muy complejos y con una vida útil larga.

Sin embargo, aún falta mucho trabajo para que los sistemas distribuidos de tiempo real adquieran el mismo nivel de desarrollo que los de propósito general. Alguna de las cuestiones abiertas son:

- *Tolerancia a fallos.* Se deben desarrollar mecanismos de tolerancia a fallos y las abstracciones adecuadas para su empleo en la programación de sistemas distribuidos.

- *Reconfiguración dinámica.* Esta funcionalidad es básica para crear aplicaciones que se adapten a las circunstancias cambiantes del entorno exterior. A este respecto se deben diseñar objetos que puedan migrar de procesador con transparencia y garantizando el cumplimiento de los requisitos temporales de las actividades del sistema. Las necesidades especiales de ciertos sistemas que no pueden detenerse su ejecución, también deben ser satisfechas. Un enfoque posible consiste en desarrollar el sistema como un conjunto de unidades reemplazables y definir un protocolo de reemplazamiento dinámico que considere las restricciones temporales del sistema.

- *Herramientas y metodologías de diseño.* La complejidad inherente al desarrollo de este tipo de sistemas sólo puede ser acometida con un conjunto de herramientas suficientemente potente y con una metodología que contemple las necesidades específicas de los sistemas de tiempo real.

- *Aplicaciones reales en los que se demuestre su viabilidad.* Aunque se tienen las bases técnicas necesarias, no se tiene mucha experiencia sobre su aplicación en sistemas reales y complejos. El desarrollo de este tipo de sistemas permitiría extraer importantes directrices sobre el uso de sistemas operativos, lenguajes de programación y arquitecturas de computadores.

Sin embargo, algunas de estas cuestiones no están resueltas satisfactoriamente ni siquiera para sistemas de tiempo real centralizados. Es necesario conocer profundamente los mecanismos básicos adecuados para resolver estas cuestiones. Este ha sido el planteamiento inicial que ha motivado este trabajo.

Capítulo 6

Desarrollo de sistemas de tiempo real basados en prioridades.

6.1 Planteamiento.

En el capítulo 4 se ha presentado el método de planificación basado en prioridades estáticas y las ampliaciones más interesantes. Este conjunto forma una base adecuada para desarrollar sistemas de tiempo real, ya que se dispone de los mecanismos necesarios para satisfacer los requisitos funcionales de los sistemas de tiempo real industriales, proporciona un nivel de abstracción suficientemente alto y el modelo de sistema es compatible con metodologías de desarrollo de software, como HOOD [Hood89a][Hood89b] y SA/RT [Ward&85] [Ward86], entre otras.

Sin embargo, hay cuestiones prácticas que pueden dificultar la inmediata aplicación del método al desarrollo de sistemas industriales. En la sección 4.7 se han enumerado algunos de estos problemas, y a continuación se citan las más relevantes en relación al trabajo desarrollado:

- El método no propone mecanismos de detección y tratamiento de fallos de ejecución.
- No se han desarrollado esquemas de realización práctica de algunas de las ampliaciones, como es el caso del cambio de modo.
- El desarrollo teórico del método se basa en un comportamiento ideal de las tareas y del núcleo de ejecución. No se consideran detalles como el tratamiento de interrupciones, los cambios de contexto, etc.
- Los mecanismos de adaptación dinámica a cambios en las condiciones de ejecución no son suficientes.

En relación al segundo punto, es importante señalar que aunque se han publicado esquemas de realización de algunos de estos protocolos su utilidad práctica es cuestionable [Rajkumar&89]. En estos trabajos se ha partido de núcleos de ejecución con

determinadas características orientadas al protocolo. Si bien este enfoque conlleva una ejecución eficiente y un uso sencillo, presenta inconvenientes, ya que si el núcleo de ejecución que se vaya a emplear no dispone de esta funcionalidad no será posible emplear estos esquemas. Por esta razón, el enfoque que se seguirá consiste en identificar los requisitos funcionales mínimos para la realización práctica de estos protocolos, teniendo en cuenta que esta funcionalidad esté disponible en los entornos de ejecución[1] comerciales.

Una de las características más destacables de la planificación basada en prioridades es la capacidad de comprobar analíticamente la planificabilidad de una aplicación. Esta propiedad es vital para la aceptación del método, ya que influye positivamente en diversas fases del ciclo de vida de una aplicación. En concreto:

- en las primeras fases de desarrollo proporciona una estimación de la planificabilidad del sistema, a partir de los tiempos previstos de cómputo de las tareas.
- cuando se ha desarrollado el código de las tareas, permite garantizar sus plazos de respuesta, sin entrar en detalles sobre la ejecución de las tareas de un sistema concreto.
- en la fase de mantenimiento, permite predecir la influencia de las modificaciones del sistema en su comportamiento temporal.

El propósito del trabajo que se va a presentar a continuación es dar respuesta a algunas de las necesidades anteriormente expuestas. En concreto:

- Análisis de la funcionalidad básica de los componentes de un sistema de tiempo real.
- Definición de los servicios necesarios que debe proporcionar un núcleo de ejecución para desarrollar sistemas basados en prioridades. Se supone que los núcleos de ejecución no proporcionan mecanismos orientados a los protocolos empleados, como ocurre en la realidad.
- Desarrollo de esquemas de realización práctica de los diversos componentes que forman un sistema de tiempo real para cumplir la funcionalidad especificada. Es fundamental que los esquemas permitan la realización del análisis de planificabilidad.

La primera cuestión se presenta a continuación y guía el desarrollo de las dos restantes. Los servicios del sistema operativo necesarios y los esquemas de tareas se desarrollarán en paralelo y de forma incremental, con el objetivo de satisfacer los requisitos funcionales identificados.

[1]En este documento, entorno de ejecución se refiere al conjunto lenguaje de programación y núcleo de sistema operativo que se emplea en el desarrollo de una aplicación.

6.1.1 Requisitos de los sistemas de tiempo real.

En esta sección se presentan los requisitos funcionales de los componentes que forman los sistemas de tiempo real basados en prioridades.

Tareas de tiempo real.

Los sistemas de tiempo real que se están considerando son concurrentes, por lo que estarán compuesto por un conjunto de tareas que se ejecutan en paralelo. Las características funcionales de estos sistemas requieren la coexistencia de al menos tres tipos de tareas: periódicas, esporádicas y de segundo plano, tal y como se describió en el apartado 3.1.2. En consecuencia, los primeros requisitos funcionales están relacionados con la activación y cumplimiento de los plazos de respuesta de estas tareas, en consonancia con sus características de ejecución. Al conjunto de parámetros que especifica las características de ejecución de las tareas se le denomina *perfil de ejecución.*

Por otro lado, las tareas críticas deben cumplir los requisitos establecidos por el método de planificación, para poder determinar analíticamente su planificabilidad (ver sección 3.3.1). Estos requisitos se concretan en:

- Las tareas deben tener una prioridad, cuyo valor depende de su perfil de ejecución, y ejecutarse con un planificador basado en prioridades y expulsivo.
- Cuando una tarea se activa no se debe bloquear hasta que finalice el cómputo de la activación.
- Las tareas esporádicas se realizarán según el protocolo del servidor esporádico.

La comunicación entre tareas es una necesidad básica cuando se desarrollan sistemas concurrentes. Por tanto se deberán determinar mecanismos de comunicación acordes con el método de planificación.

Fiabilidad.

Uno de los objetivos de cualquier proyecto de desarrollo de software es que el producto final cumpla los requisitos de su especificación y no falle su ejecución. Por ello las metodologías de desarrollo de software fomentan la programación sin errores y conceden gran importancia a la fase de pruebas del sistema.

A pesar de estos esfuerzos y con el estado tecnológico actual es utópico asegurar que el producto final carece de errores y no se producirán fallos de ejecución. Además el comportamiento de los dispositivos hardware también puede fallar. Por consiguiente, en cualquier sistema realista es necesario introducir mecanismos para detectar y tratar los posibles fallos que ocurran durante su funcionamiento. Ésto es especialmente importante en sistemas críticos en seguridad, en los que un fallo de ejecución puede producir importantes daños económicos, materiales, e incluso personales.

La primera condición para desarrollar un sistema robusto es detectar los fallos que ocurran en su ejecución. Atendiendo a su naturaleza diferente, se consideran varios tipos de fallos por separado:

- *Fallos lógicos.* En este apartado se incluyen los fallos producidos por el software. Estos fallos pueden ser operaciones ilegales, uso inadecuado de los recursos del sistemas o resultados contrarios a la especificación del sistema. Los dos primeros tipos de fallos los suele detectar el núcleo de ejecución y, de alguna manera, se los comunica a la aplicación. El tercer tipo lo detecta la propia aplicación, generalmente mediante comprobaciones internas.

- *Fallos temporales.* El comportamiento temporal de las tareas no es el adecuado. Estos fallos son característicos de los sistemas de tiempo real. En los sistemas que se están estudiando, pueden tener dos orígenes:

  - Exceso de tiempo de cómputo de una tarea. El análisis de planificabilidad supone que las tareas se activan adecuadamente y que en cada activación el tiempo de cómputo máximo es conocido y acotado. Si una tarea consume más tiempo de cómputo que el supuesto, entonces puede incumplir su plazo de respuesta o impedir que otras tareas lo cumplan. Por consiguiente, se debe comprobar que el tiempo de cómputo consumido por las tareas durante una activación no supera el estimado y utilizado en el análisis de planificabilidad.
  - Incumplimiento de plazos de respuesta. Cuando una tarea de tiempo real se activa, debe completarse antes de su plazo de respuesta. El control de los tiempos de cómputo de las tareas es fundamental para garantizar el cumplimiento de los plazos de respuesta. Sin embargo, no es suficiente. El análisis de planificabilidad puede haber sido incorrecto, bien por errores en su aplicación, bien por errores en el tratamiento de las interferencias del núcleo de ejecución cuando se producen interrupciones, cambios de contexto, etc... Además, hay núcleos de ejecución que no proporcionan mecanismos para medir el tiempo de cómputo de las tareas. En estos casos, la única comprobación factible del comportamiento temporal es la detección de incumplimientos de plazos de respuesta.

- Fallos del hardware. Es evidente que el hardware también puede fallar. Una práctica común en sistemas de tiempo real es incluir una serie de rutinas que periódicamente comprueban el funcionamiento de los dispositivos hardware. Este código se suele ejecutar como parte de tareas de segundo plano. El tratamiento de los fallos hardware depende de la criticidad del dispositivo. Algunos componentes son vitales para la ejecución de la aplicación, como es el caso de los buses, la memoria o el procesador. La única forma de poder recuperarse de estos fallos es mediante la replicación de estos dispositivos, aunque estas técnicas están fuera del ámbito de este trabajo. La detección de fallos en otros componentes como son los sensores o actuadores, permitiría al menos saber que el funcionamiento del sistema no es adecuado y utilizar mecanismos alternativos o, si no hay opción, detener la ejecución de la aplicación.

En este trabajo no se va a tratar el problema de la detección de fallos en toda su complejidad. Sólo se tratarán en profundidad aquellos aspectos específicos de los sistemas de

tiempo real. En concreto, se estudiará en detalle la detección de los fallos de temporización durante la ejecución del sistema. Por otro lado, y motivado por los objetivos primarios de este trabajo, los mecanismos de tolerancia a fallos que se estudiarán, están orientados a degradar el controladamente el sistema en presencia de fallos. Este tratamiento se centra en las siguientes cuestiones:

- La tarea que ha fallado termina su ejecución.
- El fallo se encapsula para que no se transmita incontroladamente a unidades correctas.
- La funcionalidad del sistema se degrada controladamente, para minimizar los daños externos.

Dinamicidad.

Una característica deseable en los sistema de tiempo real es su capacidad para adaptarse dinámicamente a los cambios que acontecen durante su ejecución. Estos cambios son de diversa naturaleza y se pueden agrupar como sigue:

- *Cambios de las condiciones del entorno previstos y lógicos en su evolución.* Una forma adecuada de tratar estos eventos es mediante el cambio de modo de ejecución del sistema. Los estados del entorno exterior, con el que indirecto la aplicación, se agrupan en modos según un criterio de semejanza de requisitos computacionales. Para cada modo se diseña un conjunto de tareas que satisface estos requisitos. Cuando las condiciones externas varían suficientemente, el modo de ejecución del sistema cambia y, consecuentemente, el conjunto de las tareas en ejecución.
- *Fallos de ejecución.* Cuando se detecta un fallo en la ejecución del sistema, es necesario tratarlo dinamicamente, para reducir en lo posible sus efectos negativos. En este trabajo, cuando se produce un fallo, la funcionalidad del sistema se degrada controladamente. En particular, se detiene la ejecución de la tarea errónea y se activa una tarea de repuesto, que al menos garantiza que no se producirán daños en el sistema. Las tareas a las que no perjudica este fallo, continuará su ejecución sin variación.
- *Cambios de las condiciones del entorno imprevisibles durante el diseño del sistema.* La fase de mantenimiento de software es la más larga en el ciclo de vida de una aplicación y durante ella suele surgir la necesidad de sustituir partes de la aplicación por diversas circunstancias:
  - Cambios en la especificación del sistema o en el entorno de ejecución, imposibles de prever durante el diseño del software.
  - Fallos graves en su ejecución.
  - Incumplimiento de requisitos funcionales.

La forma habitual de mantener los sistemas es detener su ejecución, sustituir el código y reanudar la ejecución de la aplicación. Este procedimiento no es siempre posible. Hay sistemas que no se pueden detener sin causar daños. Una solución a este problema es diseñar el sistema mediante un conjunto de unidades que se pueden reemplazar sin detener el resto del sistema. En el capítulo siguiente se presenta un protocolo de reemplazamiento dinámico de software en sistemas de tiempo real crítico y se integra con las estructuras de tareas que se van a presentar.

6.1.2 Entornos de ejecución.

La tendencia actual consiste en normalizar lenguajes y sistemas operativos con objeto de aumentar la transportabilidad de las aplicaciones que se realizan con ellos. En la actualidad hay dos normas principales para desarrollar sistemas de tiempo real:

- POSIX: El objetivo de esta norma es definir un interfaz y un entorno normalizado de un sistema operativo basado en UNIX. Esta norma agrupa a un conjunto de documentos. El primero de los cuales define un conjunto de servicios del sistema operativo para obtener transportabilidad del código fuente [POSIX.1 90]. Estos servicios no son adecuados para el desarrollo de sistemas de tiempo real, por lo que se han redefinido y ampliado en otros documentos:

  - Extensiones para sistemas de tiempo real [POSIX.1b 93]. Se redefinen los servicios del POSIX 1003.1 para adecuarlos al desarrollo de sistemas de tiempo real.
  - Extensiones de tareas ligeras [POSIX.1c 93]. El objetivo de este documento es completar el documento base de POSIX para disponer de múltiples flujos de control (tareas ligeras o *threads*) dentro de una tarea. El comportamiento de estos elementos se ha diseñado para adecuarse al desarrollo de sistemas de tiempo real.
  - Definición de perfiles de aplicación [POSIX.13 92] Este documento define varios entornos de ejecución de sistemas de tiempo real basados en POSIX.

  En la definición de las normas POSIX, la especificación de las llamadas al sistema se hace en lenguaje C. Esta aproximación debe cambiarse de acuerdo a una directiva de ISO, según la cuál la norma básica se desarrolla sin referenciar a un lenguaje concreto. Posteriormente se deberán desarrollar particularizaciones para cada uno de los lenguajes. En cualquier caso el panorama actual es que la práctica totalidad de las normas aprobadas definen los servicios del sistema operativo en lenguaje C.

- Ada: Es un lenguaje de programación cuyo desarrollo fue patrocinado por el Departamento de Defensa de EEUU para desarrollar sistemas empotrados de tiempo real [AdaLRM83]. Tres objetivos principales marcaron la definición del lenguaje:

  - *Fiabilidad y facilidad mantenimiento.* El lenguaje promueve la legibilidad antes que la facilidad de escritura y ofrece la posibilidad de realizar compi-

lación separada de unidades de programa, garantizando que se realizan las mismas comprobaciones entre unidades que dentro de una unidad.

- *Programación como una actividad humana.* A este respecto se intentó de mantener el lenguaje lo más reducido posible, a pesar de la complejidad del entorno de aplicación. Se ha tratado de cubrir este dominio con un pequeño número de conceptos básicos integrados de forma consistente y sistemática. El desarrollo de programas es cada vez más una labor descentralizada y distribuida. En consecuencia, la posibilidad de ensamblar un programa a partir de piezas software producidas independientemente, ha sido una idea básica en su diseño. Los conceptos de paquetes, unidades genéricas y tipos privados están intimamente relacionados con esta idea.
- *Eficiencia.* Cada construcción del lenguaje fue examinada a la luz de las técnicas de implementación. Aquellas construcciones del lenguaje cuya implementación no era clara o requería excesivos recursos fue eliminada.

Las características más destacables del lenguaje son [Barnes89] [Burns&89]:

- Legibilidad.
- Tipado fuerte.
- Programación a gran escala.
- Tratamiento de excepciones.
- Abstracción de datos.
- Concurrencia.
- Unidades genéricas.
- Programación de bajo nivel.

El lenguaje de programación Ada presenta algunos inconvenientes para el desarrollos de sistemas de tiempo real [Burns&87b] [Cornhill&87a] [Alonso&90]. Esta es una de las razones por las que esta norma se ha revisado y está siendo modificada. El resultado es el lenguaje denominado Ada 9X [Ada9XRM93], que presenta algunas características muy interesantes para los objetivos de este trabajo y que como se justificará posteriormente, será utilizado para codificar los esquemas de tareas. Este lenguaje no está muy difundido porque está aún en proceso de normalización, por lo que continuación se comentan las características más novedosas en relación a este documento.

El lenguaje de programación Ada 9X

El objetivo principal del proyecto Ada 9X es revisar el lenguaje de programación Ada [AdaLRM83] para reflejar los requisitos esenciales de sus usuarios. El enfoque seguido divide el lenguaje en un núcleo y una serie de anexos [Ada9XRM93]. Los compildores conformes a la norma, deberán proporcionar en su totalidad la funcionalidad del núcleo del lenguaje. Los anexos proporcionan características especiales para áreas de aplicación

específicas y no es obligatorio su realización. Especialmente relevantes en el ámbito de este trabajo son el anexo de programación de sistemas y el de tiempo real.

A continuación se comentan brevemente las características nuevas del lenguaje que se emplean en la programación de los esquemas que se presentarán en las secciones siguientes:

- *Objetos protegidos.*

  Los objetos protegidos [Ada9XIntro93] [Ada9XRM93] se utilizan para sincronizar el acceso a datos compartidos. Han sido incluidos al lenguaje para disponer de un mecanismo pasivo de comunicación entre tareas. Su diseño está basado en los monitores y las variables condición.

  Un objeto protegido está formado por unos datos comunes y un conjunto de subprogramas para acceder a ellos y que pueden ser de tres clases: procedimientos, puntos de entrada y funciones. Los dos primeros realizan operaciones de lectura/escritura de los datos, mientras que las funciones sólo pueden leerlos. Se pueden ejecutar simultaneamente varias operaciones de lectura, mientras que las operaciones de escritura se ejecutan con exclusión mutua. La ejecución de los puntos de entrada está controlada por una expresión lógica denominada guarda. Cuando una tarea llama a un punto de entrada se puede ejecutar si la guarda es cierta. En caso contrario, la llamada se encola en una lista asociada al punto de entrada.

- *Transferencia asíncrona de control.*

  La transferencia asíncrona de control en Ada 9X se realiza mediante una sentencia de selección. La transferencia de control se realiza después de completarse una llamada a un punto de entrada o cuando expira un temporizador. La estructura de esta sentencia es:

```
select
    -- Alternativa asíncrona
    Sentencia_Bloqueante;
    Sentencias_1;
then abort
    -- Alternativa abortable
    Sentencias_2;
end select;
```

  Esta sentencia tiene dos alternativas: la alternativa asíncrona y la abortable. La primera, debe comenzar con una operación bloqueante, que será una llamada a un punto de entrada de otra tarea o una sentencia de retardo. Si esta operación se completa inmediatamente, se ejecutarán `Sentencias_1` y se termina la sentencia de selección. Si no se completa inmediatamente, la alternativa asíncrona queda a la espera y se comienza a ejecutar la alternativa abortable. Si se completa la ejecución

de `Sentencias_2` antes de que se desbloquee la alternativa asíncrona, la sentencia de selección termina y se cancela esta alternativa. Si por el contrario, se desbloquea la llamada antes de completar `Sentencias_2` se aborta su ejecución y continúa ejecutándose la alternativa asíncrona.

La realización práctica de esta sentencia es compleja. Para simplificarla parcialmente se han restringido las sentencias que pueden incluirse en la alternativa abortable. El objetivo es que su comportamiento sea lo más parecido al cuerpo de un procedimiento. Por esta razón se han excluido las sentencias que transfieran el flujo de control fuera de la sentencia de selección y las sentencias **accept**.

- *Contabilidad del tiempo de cómputo.*

  El anexo de tiempo real [Ada9XAnS92] incluyó durante algún tiempo la especificación de un paquete que permite medir el tiempo de cómputo consumido por una tarea. Finalmente se ha decidido no incluir esta funcionalidad para reducir la complejidad del anexo. En cualquier caso, dado su interés y la práctica certeza de que este paquete esté disponible aunque no sea parte de la norma, se empleará en el desarrollo de los esquemas de este capítulo.

  La contabilidad del tiempo de cómputo se basa en cargar unidades discretas de tiempo de procesador a las tareas. La precisión de la unidad de contabilidad depende de las capacidades del sistema.

  La especificación del paquete es como sigue:

```
with SYSTEM.REAL_TIME;
package CPU_TIME_ACCOUNTING is

    ACCOUNTING_UNIT: constant
            REAL_TIME.FINE_DURATION:= .  .  .  --implementation defined;
    subtype ACCOUNTING_RANGE is REAL_TIME.FINE_DURATION
    range 0.0..REAL_TIME.FINE_DURATION'LAST;

    function VALUE(T: SYSTEM.ROOT_TASK_CLASS) return ACCOUNTING_RANGE;
    procedure SET_VALUE
            (T: SYSTEM.ROOT_TASK_CLASS; VALUE: ACCOUNTING_RANGE);

    protected type OVERRUN_SIGNAL is
        procedure CLEAR;
        entry WAIT;
        pragma INTERRUPT_PRIORITY;
    private
        ...  -- implementation defined
    end OVERRUN_SIGNAL;

    type SIGNAL_PTR is access all OVERRUN_SIGNAL;

    procedure ATTACH_OVERRUN_SIGNAL
            (T: SYSTEM.ROOT_TASK_CLASS; SIGNAL: SIGNAL_PTR);
```

```
end CPU_TIME_ACCOUNTING;
```

El tipo OVERRUN_SIGNAL se usa para indicar la sobreutilización de procesador. La llamada a WAIT bloquea al llamador hasta que se realiza una llamada a CLEAR. El comportamiento de este paquete se puede explicar en términos de un contador virtual asociado a cada tarea. El contador representa el tiempo de procesador de que dispone la tarea. La función VALUE devuelve el valor de este contador. El procedimiento ATTACH_OVERRUN_SIGNAL asigna un objeto del tipo OVERRUN_SIGNAL a una tarea T.

El procedimiento SET_VALUE asigna un valor al contador asociado a la tarea T. Después de asignar un valor positivo, se decrementa el contador en la cantidad ACCOUNTING_UNIT cada vez que la tarea consume una unidad de procesador, hasta que el contador llega a cero. La implementación llama al procedimiento CLEAR del objeto asignado a una tarea cada vez que su contador llega a cero. Esta operación no debe ser invocada por el usuario.

Mediante este paquete es posible programar temporizadores relativos al tiempo de cómputo de una tarea, como se mostrará posteriormente.

- *Funcionalidad adicional en el anexo de tiempo real.*

  El anexo de tiempo real define una serie de paquetes que proporciona al programador con funcionalidad adicional para desarrollo de sistemas de tiempo real. Las características más relevantes para este trabajo son:

  - Prioridades dinámicas. El paquete ADA.DYNAMIC_PRIORITIES proporciona dos procedimientos para obtener y cambiar la prioridad de una tarea.
  - Techo de prioridades de objetos protegidos. Los pragmas PRIORITY e INTERRUPT_PRIORITY permite definir el techo de prioridad de un objeto protegido. Cuando se ejecuta un subprograma del objeto protegido, se ejecuta con esta prioridad. Si una tarea con prioridad mayor que el techo intenta acceder a un objeto protegido, se eleva la excepción PROGRAM_ERROR.
  - Colas ordenadas por prioridades. El *pragma* QUEUING_POLICY(....) permite definir la política de ordenación de las colas asociadas a puntos de entrada de tareas y de objetos protegidos, entre otros elementos. El anexo de tiempo real obliga a que se incluya la política PRIORITY_QUEUING que ordena las tareas encoladas según su prioridad.

6.1.3 Directrices de codificación.

En las siguientes secciones de este capítulo se van a presentar esquemas de realización práctica de tareas de tiempo real, con las características reseñadas anteriormente. En su realización se ha dado mucha importancia a la claridad de la presentación, con objeto de facilitar la comprensión de las características representadas. La claridad de los programas

está influida por dos elementos principales: la estructura de los esquemas y el lenguaje de representación.

En relación al primer factor, se ha empleado un esquema fijo en el que se integran los sucesivos cambios necesarios para realizar la complejidad creciente de la funcionalidad de las tareas. Este enfoque incremental permite identificar más fácilmente las modificaciones introducidas, respecto a la realización anterior.

La legibilidad de los programas en Ada 9X [Ada9XRM93] y su adecuación para el desarrollo de sistemas de tiempo real ha motivado su selección para el desarrollo de los esquemas[2]. La alternativa de utilizar POSIX y un lenguaje de programación dificultaría la comprensión del código. El código que se va a presentar está estructurado en paquetes, tal y como es habitual en Ada. Una alternativa razonable es definirlos como paquetes genéricos. Sin embargo, para centrar la atención en los aspectos relativos al comportamiento de las tareas, esta opción se ha descartado.

Por otro lado y de forma consciente se va a conculcar la regla sintáctica de Ada que indica que una sentencia `accept` no se puede incluir en el cuerpo de un procedimiento. Los esquemas de tareas que se van a presentar pretenden ser un modelo para la programación en cualquier entorno de ejecución de sistemas de tiempo real con las características mostradas. La práctica anterior permite agrupar lógicamente determinadas operaciones, de forma que la presentación es más clara y fácil de entender. En cualquier caso, la modificación de los esquemas para compilar correctamente en Ada 9X es inmediata.

En los diversos esquemas que se van a presentar se han utilizado dos paquetes. Uno es global a todas las tareas del sistema y se denomina `Global`. Incluye las declaraciones de elementos como los tipos que representa los perfiles de ejecución de las tareas, los modos del sistema, el instante de inicio de la ejecución, etc..

El otro se emplea para declarar una tarea de tiempo real. En este enfoque, la especificación del paquete exporta los mecanismos necesarios para que otras tareas se comuniquen con ella. Como se justificará posteriormente, el interfaz de las tareas periódicas será nulo y el de las esporádicas será el imprescindible para la notificación del evento correspondiente. La tarea se declara en el cuerpo del paquete, en el que se también se incluyen declaraciones de otras variables necesarias para su codificación. El cuerpo de la tarea se define en una subunidad, con objeto de separar la declaración de variables y otros elementos auxiliares de la estructura interna de la tarea. Desde un punto de vista práctico, este enfoque permite modificar el cuerpo de la tarea sin necesidad de recompilar el paquete. Un esquema de este componente se presenta a continuación:

```
package Tarea_Periódica is
end Tarea_Periódica;

package body Tarea_Periódica is
    -- Declaraciones de parámetros de la tarea
```

[2]En el momento de escribir esta tesis, el lenguaje Ada 9X estaba en fase de normalización. Por ello, puede haber algunas inconsistencias entre el código presentado y la versión definitiva del lenguaje.

Igualmente es posible que existan errores de sintaxis. En el momento de la codificación no había un compilador completo del lenguaje.

```
    . . . .
    task Tarea_Periódica is
    . . .
    end Tarea_Periódica;

    task body Tarea_Periódica is separate;

end Tarea_Periódica;
```

La funcionalidad de las tareas de tiempo real conlleva un determinado código que se ejecuta cada vez que la tarea se activa. Esta activación se asocia a la expiración de un temporizador, si la tarea es periódica, o a la ocurrencia de un evento concreto, si es aperiódica. Por consiguiente, la estructura más razonable es un bucle que se ejecuta una vez en cada activación y como consecuencia de las acciones anteriores. La ejecución de la tarea sólo se detendrá si falla su ejecución o se para el sistema. Cuando se activa por primera vez la tarea, es necesario realizar unas acciones de inicio, que configuren adecuadamente su entorno de ejecución. Estas operaciones se incluyen en el procedimiento `Operaciones_Iniciales`. El código del bucle está formado por tres procedimientos:

- `Operaciones_Previas`. Operaciones que se deben de ejecutar antes de ejecutar el código propio de la activación.
- `Actividad_Tarea`. Código que incluye la funcionalidad específica de la tarea en la aplicación.
- `Operaciones_Posteriores`. Operaciones que se realizan después de completar el código de la actividad. En general serán operaciones de comprobación del entorno de ejecución y de preparación para la próxima activación.

El esquema resultante se presenta a continuación:

```
separate (Tarea_Periódica)

task body Tarea_Periódica is

    -- ****************************************************

    procedure Operaciones_Iniciales is

    begin
        . . .
    end Operaciones_Iniciales;

    -- ****************************************************
    procedure Operaciones_Previas is

    begin
        . . .
    end Operaciones_Previas;
```

```
    -- ****************************************************
    procedure Actividad_Periódica is

    begin
        . . .
    end Actividad_Periódica;

    -- ****************************************************
    procedure Operaciones_Posteriores is

    begin
        . . .
    end Operaciones_Posteriores;

-- ****************************************************
begin -- Tarea_Periódica
    Operaciones_Iniciales;
    loop
        Operaciones_Previas;
        Actividad_Periódica;
        Operaciones_Posteriores;
    end loop;
end Tarea_Periódica;
```

6.1.4 Características generales de los sistemas.

En conclusión, los sistemas de tiempo real que se pretenden realizar deben presentar las siguientes características funcionales:

- Planificación basada en prioridades.
- Coexistencia de tareas periódicas, esporádicas y de segundo plano.
- Comunicación entre tareas
- Comprobación analítica de la planificabilidad del sistema.
- Cambio del modo de ejecución del sistema.
- Detección de fallos de temporización.
- Tratamiento de fallos de ejecución.
- Reemplazamiento dinámico de software.

6.2 Tareas de tiempo real crítico.

Los sistemas objeto de estudio son sistemas concurrentes. Por consiguiente la unidad de concurrencia, en este caso la tarea, es el componente fundamental. En esta sección se analizan los requisitos básicos de las tareas, los mecanimos de comunicación y esquemas de detección de fallos de ejecución.

6.2.1 Funcionalidad básica de las tareas.

La funcionalidad requerida de las tareas se analiza y se presentan esquemas de programación en Ada 9X que satisfacen estos requisitos.

Núcleo de ejecución.

Este capítulo estudia las cuestiones prácticas relacionadas con el desarrollo de sistemas de tiempo real concurrentes y basados en prioridades. La prioridad es una indicación de la importancia relativa de una tarea y se emplea para gestionar los recursos del sistema. Es decir, cuando varías tareas compitan por un recurso compartido, éste se concederá a la tarea más prioritaria entre las que lo solicitan. Por tanto, los núcleos de ejecución validos para producir estos sistemas deberán seguir este principio.

La mayoría de los núcleos de ejecución siguen este enfoque para gestionar el uso del procesador. Sin embargo, es importante constatar que también se aplica para gestionar otros recursos del sistema, ya que en caso contrario pueden darse casos adicionales de inversión de prioridades. Entonces, habrá que desarrollar métodos específicos para resolver estas situaciones o descartar el uso del núcleo para desarrollar sistemas de tiempo real con este método. Un ejemplo de esta situación ocurre en Ada [AdaLRM83], donde las colas de los puntos de entrada se ordenan según el instante de llegada y no según la prioridad de las tareas bloqueadas. Como solución se ha desarrollado un esquema de comunicación entre tareas, tal que nunca hay tareas en estas colas [Sha&90][Sha&89a] (en el apartado 8.3.5 se presenta esta solución).

El comportamiento del núcleo de ejecución debe ser determinista, para poder garantizar plazos de respuesta, i.e. realizar el análisis de planificabilidad. La norma [POSIX.1b 93] define el conceto de tiempo real en sistemas operativos como:

> La capacidad del sistema operativo para proporcionar los servicios requeridos con un tiempo de respuesta acotado.

Por consiguiente, la duración máxima de ejecución de los servicios que proporcionan los sistemas operativos para tiempo real debe conocerse a priori. Se puede añadir que éste valor no debe ser muy grande. Habitualmente, mientras se ejecutan estas operaciones no se puede interrumpir al sistema operativo. Si este tiempo es elevado, se vería limitada la capacidad de las aplicaciones de responder a los eventos del sistema con suficiente rapidez.

Activación de las tareas.

Tareas periódicas. Las tareas periódicas se ejecutan contínuamente. Se activan al comienzo del período y se deben completar antes del plazo de respuesta. La programación del instante de inicio de la activación se realiza mediante temporizadores relativos al reloj del sistema. Su formato suele ser una palabra reservada o una llamada a procedimiento con, al menos, un parámetro que caracteriza el retardo. Según sea este parámetro, se tienen dos tipos de temporizadores:

- Temporizadores relativos. El parámetro indica el tiempo que debe transcurrir hasta el fin de la temporización, medido a partir del instante en que se ejecuta el comando.
- Temporizadores absolutos. El parámetro indica el instante de tiempo en que el temporizador debe expirar.

Los temporizadores absolutos son los más adecuados para la programación del instante de la activación de una tarea periódica, pues es precisamente este valor el parámetro del comando de temporización. Si sólo se dispone de la posibilidad de establecer retardos relativos, el argumento del comando será la diferencia entre el instante de la siguiente activación y la hora actual del sistema. Si esta operación no es atómica se pueden producir errores de deriva, ya que se puede expulsar a la tarea antes de haberse programado el temporizador, pero después de haber realizado la diferencia entre estos valores de tiempo [Burns&87b] [Cornhill&87a].

La precisión de la temporización es otro factor importante para determinar la utilidad de los servicios de un núcleo de ejecución. Intérnamente, el núcleo calcula, a partir de los parámetros del comando, el número de interrupciones de reloj que deben producirse para que expire el temporizador. La frecuencia con que el reloj interrumpe determina el error máximo de una temporización, entendido como la diferencia entre el instante en que se solicita el fin de una temporización y el instante físico en que se produce.

Tareas esporádicas. Una tarea esporádica se implementa como un servidor esporádico, de acuerdo al protocolo presentado en la sección 4.4. La tarea esporádica permanece bloqueada, a la espera del evento que debe tratar. La naturaleza de este evento es diversa y determina cómo se notificará a la tarea:

- *Una interrupción.* La tarea esporádica realiza las operaciones de tratamiento de la interrupción. La tarea puede estar diréctamente acoplada a una interrupción o la rutina de tratamiento de la interrupción indica a la tarea esporádica su ocurrencia. La selección de uno de estos enfoques, depende de los detalles de tratamiento de interrupciones del entorno de ejecución en cuestión.
- *Un mensaje de otra tarea.* En esta situación, otra tarea del sistema le comunica a la tarea esporádica la ocurrencia del evento que debe tratar. La tarea esporádica estará bloqueada a la espera de la recepción del mensaje.

- *Una señal software*. Una señal se puede definir como una interrupción software y es un mecanismo disponible en sistemas operativos basados en UNIX [Kernighan&84] [POSIX.1b 93]. Su transmisión y tratamiento es muy similar a las interrupciones y se producen mediante la ejecución de un servicio del sistema operativo. Las señales son especialmente adecuadas cuando se debe activar la tarea esporádica al terminar una operación asíncrona.

Las tareas esporádicas también requieren temporizadores para su programación. Se debe asegurar que no se trata otro evento antes de que transcurra la separación mínima entre eventos, a partir del instante en que se produjo el último evento tratado. Los requisitos de los temporizadores son exáctamente iguales a los expresados para tareas periódicas.

Tareas de segundo plano. Las tareas de segundo plano son aquellas tareas que no tienen requisitos de tiempo para ejecutar. Estas tareas tienen que tener una prioridad menor que cualquier tarea de tiempo real. Además, es aconsejable que no bloqueen a ninguna de ellas. Si es necesario que se comuniquen con una tarea de tiempo real, el tiempo de bloqueo que introducen debería ser lo menor posible.

En estas condiciones las tareas de segundo plano no influyen, o lo hacen de forma controlada, en la ejecución de las tareas de tiempo real. En cuanto a sus requisitos funcionales, se puede afirmar que no presentan ninguno específico y por lo tanto no serán consideradas en el resto del estudio.

Análisis de planificabilidad.

El método de planificación presupone un comportamiento específico de las tareas de tiempo real cuando ejecutan el código relativo a una activación. Este comportamiento se concreta en las siguientes características:

- La tarea no debe cambiar su prioridad. Algunas excepciones a esta regla son el protocolo de cambio de modo en el sistema y en determinados enfoques de realización de los protocolos del techo de prioridad. En estos casos se hará de acuerdo con el protocolo correspondiente, ya que se han desarrollado métodos de análisis de planificabilidad que contemplan estas alteraciones de las reglas.

- El tiempo de cómputo máximo debe estar acotado y conocerse. Esto quiere decir que hay que evitar el uso de estructuras de programa que resulten en un tiempo de ejecución no acotado. Esta norma es la más fácil de cumplir, pues en el código de los esquemas nunca se usaran estas estructuras de forma anómala.

- Cuando la tarea se activa, sólo se bloqueará cuando haya otras tareas preparadas más prioritarias, al comunicar con otras tareas o porque ha completado la activación. Alternativamente, esta condición implica que en ausencia de otras tareas, una tarea de tiempo real comienza a ejecutarse cuando se producen las condiciones de

activación, se ejecuta sin interrupciones hasta completar el código de la activación y se queda bloqueada hasta que se den las condiciones de la siguiente activación.

En relación a los esquemas que se presenten se comprobará que su comportamiento cumple las anteriores normas. En aquellos casos que no sea así, se indicará como contemplar estas anomalías en el análisis de planificabilidad.

Una parte de las tareas es el código específico de la aplicación, que debe aportar el usuario. Este código también debe cumplir las reglas anteriores, lo que habría que comprobar para garantizar el comportamiento de las tareas en su conjunto. Lo más adecuado sería desarrollar una herramienta que detectara estos errores mediante un análisis sintáctico del código fuente.

Esquema de realización.

En primer lugar se describe el paquete `Global`, en el que se incluyen las declaraciones globales a todas las tareas. En este caso sólo es necesario declarar los tipos que representan el perfil de ejecución de las tareas y el instante real de inicio de la ejecución del sistema. Este valor contempla el tiempo que se consume en la elaboración de los componentes del sistema. Por tanto, cuando se produzca la primera activación de las tareas, se puede asegurar que todos los componentes están preparados.

```
with System;
with Calendar;

   package Global is

      type Tipo_Perfil_Periódico is record
         Período        :  Duration;
         Fase           :  Duration;
         Prioridad      :  System.Any_Priority;
         Plazo_Respuesta:  Duration;
      end record;

      type Tipo_Perfil_Esporádico is record
         Separación     :  Duration;
         Prioridad      :  System.Any_Priority;
         Plazo_Respuesta:  Duration;
      end record;

      Instante_Inicio :  Calendar.Time := . . . ;

   end Global;
```

Tarea periódica. La tarea se encapsula en un paquete, lo que es una práctica habitual en aplicaciones de tiempo real en Ada. El interfaz que exporta el paquete es nulo porque la tarea no proporciona ningún mecanismo para que otras tarea se comuniquen con ella.

La tarea se incluye en la aplicación mediante una cláusula de uso `with`, que suele estar en el procedimiento principal. De esta forma, al elaborar este procedimiento se elabora la tarea periódica. El cuerpo de este paquete es muy sencillo y sólo incluye la declaración de la variable que contiene el perfil de la tarea, al que se asignan unos valores arbitrarios.[3]

```
package Tarea_Periódica is
end Tarea_Periódica;


with Global;

package body Tarea_Periódica is

    Perfil_Tarea    :  Global.Tipo_Perfil_Periódico :=
        (Período=> 0.1, Fase=> 1.0, Prioridad=> 3,
        T_Cómputo=> 0.01, Plazo_Respuesta=> 0.1);

    task Tarea_Periódica is
    end Tarea_Periódica;
    task body Tarea_Periódica is separate;

end Tarea_Periódica;
```

El código de la tarea se reduce a la ejecución de la actividad periódica y a la programación de la siguiente activación. El contenido de los procedimientos generales que forman el cuerpo de la tarea se comentan a continuación:

- `Operaciones_Iniciales`: Las operaciones iniciales son las relativas al establecimiento del perfil de ejecución de la tarea y la programación de la primera activación. En relación al perfil de ejecución, se asigna a la tarea la prioridad de ejecución correspondiente. El instante de la primera activación de la tarea se determina a partir de un instante de tiempo, que en el esquema de ejecución se representa por `Instante_Inicio`, y la fase del perfil de la tarea. El primero es un parámetro global al sistema y representa el intervalo necesario para que todas las tareas del sistema ejecuten las operaciones de inicio. El segundo es un parámetro específico del perfil de ejecución de las tareas periódicas.

- `Operaciones_Previas`. No es necesario ejecutar ninguna operación.

- `Operaciones_Posteriores`. Este procedimiento incluye las operaciones necesarias para programar el siguiente instante de activación. Este valor se calcula a partir instante anterior y el período de la tarea. A continuación se ejecuta una sentencia de retardo absoluto hasta el instante calculado.

[3]Está previsto que en el anexo de tiempo real de Ada 9X se definan tipos que permitan especificar temporizaciones con más precisión que la proporcionada por los mecanismos en el núcleo del lenguaje. Sin embargo, en este momento no está claro la forma que tomarán los paquetes correspondientes. Por ello, en los esquemas que se van a presentar se emplearán los mecanismos del núcleo del lenguaje, siendo inmediata su sustitución por los del anexo.

```
with Calendar;
with Ada.Dynamic_Priorities;

separate (Tarea_Periódica)
task body Tarea_Periódica is
    Siguiente_Activación :  Calendar.Time;

-- ***************************************************

    procedure Operaciones_Iniciales is

    begin
        Ada.Dynamic_Priorities.Set_Priority(Perfil_Tarea.Prioridad, Tarea_Periódica);
        Siguiente_Activación := Global.Instante_Inicio + Perfil_Tarea.Fase;
        delay until Siguiente_Activación;
    end Operaciones_Iniciales;
-- ***************************************************
    procedure Operaciones_Previas is

    begin
        null;
    end Operaciones_Previas;

-- ***************************************************
    procedure Actividad_Periódica is

    begin
        . . .
    end Actividad_Periódica;

-- ***************************************************
    procedure Operaciones_Posteriores is
    begin
        Siguiente_Activación := Siguiente_Activación + Perfil_Tarea.Período;
        delay until Siguiente_Activación;
    end Operaciones_Posteriores;

-- ***************************************************
begin -- Tarea_Periódica
    Operaciones_Iniciales;
    loop
        Operaciones_Previas;
        Actividad_Periódica;
        Operaciones_Posteriores;
    end loop;

end Tarea_Periódica;
```

El funcionamiento de este esquema es muy sencillo. La expiración del temporizador se programa para el siguiente instante de activación de la tarea. Cuando éste ocurre la tarea pasa de bloqueada a lista. Entonces se ejecuta el código de la activación sin que el

esquema bloquee la tarea hasta que se complete. Evidentemente, el planificador puede expulsar a la tarea, según el comportamiento normal del método de planificación.

La realización práctica de los temporizadores está asociado a un reloj externo que interrunpe al procesador. Cuando expira, el núcleo debe realizar una serie de operaciones como pasar la tarea de la cola de tareas bloqueadas a la cola de tareas listas, gestionar posibles temporizaciones pendientes y llamar al planificador para que decida qué tarea se debe ejecutar a continuación. Esta operación puede suponer una inversión de prioridades, que aunque sea pequeña hay que tenerla en cuenta al realizar el análisis de planificación. La inversión de prioridades se produce si una tarea es interrumpida por un temporizador programado por una tarea menos prioritaria. En el caso más desfavorable, durante la ejecución de una activación de una tarea pueden expirar un temporizador por cada tarea menos prioritaria del sistema. Si no se considera, el efecto acumulado por el tratamiento de estos temporizadores podría impedir que la tarea cumpliera su plazo de respuesta.

Es inmediato comprobar el comportamiento adecuado de este esquema en relación al análisis de planificabilidad. La prioridad de ejecución se establece inicialmente y no se modifica nunca más. La tarea se activa cuando expira el temporizador y permanece preparada o lista para ejecutar hasta que se completa la activación.

Tareas esporádicas. El paquete que encapsula a las tareas esporádicas es análogo al correspondiente a las tareas periódicas. La diferencia estriba en que en este paquete se debe especificar el mecanismo para notificar los eventos asociados, que puede ser una interrupción, una señal o un mensaje de otra tarea. En este caso, el aviso del acontecimiento del evento se le comunica mediante una llamada al punto de entrada `Notificar_Evento` de la tarea esporádica. Como este punto de entrada será llamado por una tarea externa, es necesario incluir su declaración en la parte visible del paquete.

```
with Calendar;

package Tarea_Esporádica is
    task Tarea_Esporádica is
        entry Notificar_Evento(Instante_Detección :  in Calendar.Time);
    end Tarea_Esporádica;
end Tarea_Esporádica;


with Global;

package body Tarea_Esporádica is
    Perfil_Tarea    :  Global.Tipo_Perfil_Esporádico :=
        (Separación => 0.1, Prioridad => 3,
        T_Cómputo => 0.01, Plazo_Respuesta => 0.1);

    task body Tarea_Esporádica is separate;

end Tarea_Esporádica;
```

El esquema de la tarea esporádica es similar la tarea periódica. La tarea esporádica, después de ejecutar el código de inicio, queda bloqueada a la espera del evento que debe tratar. Esta espera se realiza en este caso mediante una sentencia de aceptación de una llamada al punto de entrada `Notificar_Evento`. A continuación se determina el instante de tiempo en que se ha producido la activación, para no tratar otro evento antes de que transcurra la separación mínima. El contenido de los procedimientos del cuerpo de la tarea es:

- `Operaciones_Iniciales`. Este procedimiento determina la prioridad de la tarea y programa el instante en que debe comenzar a ejecutar. Las tareas esporádicas no tienen fase inicial y pueden tratar eventos a partir instante inicial del sistema. Se inicializa la variable `Activación_Previa`, para asegurar que el primer evento no sea considerado error. El propósito de esta variable se justifica a continuación.
- `Operaciones_Previas`. Este procedimiento realiza las operaciones de validación de la ocurrencia del evento. La separación mínima entre eventos sucesivos es un factor importante de las tareas esporádicos y vital para garantizar la planificabilidad del sistema. Por tanto hay que comprobar que no ocurren eventos antes de que transcurra este tiempo.

  El control de este parámetro se hace mediante la variable `Activación_Previa` que almacena el instante en que se produjo el evento anterior. Si el siguiente evento ocurre antes de que transcurra la separación entre eventos entonces se ha producido un error y, en este caso, se aborta la tarea. En secciones posteriores se proponen métodos alternativos para tratar los fallos de ejecución.
- `Operaciones_Posteriores`. No es necesario realizar operaciones después de ejecutar el código de la actividad.

```
with Ada.Dynamic_Priorities;

separate (Tarea_Esporádica)
task body Tarea_Esporádica is
   Activación_Previa, Instante_Activación :  Calendar.Time;

-- ****************************************************

   procedure Esperar_Evento(Instante_Activación :  out Calendar.Time) is

   begin
      accept Notificar_Evento(Instante_Detección) do
         Instante_Activación := Instante_Detección;
      end Notificar_Evento;
   end Esperar_Evento;
-- ****************************************************

   procedure Operaciones_Iniciales is

   begin
```

```
        Ada.Dynamic_Priorities.Set_Priority(Perfil_Tarea.Prioridad, Tarea_Funcional);
        Activación_Previa := Global.Instante_Inicio - Perfil_Tarea.Separación;
        delay until Global.Instante_Inicio;
    end Operaciones_Iniciales;
-- *************************************************
    procedure Operaciones_Previas is

    begin
        Esperar_Evento(Instante_Activación);
        if Instante_Activación < Activación_Previa + Perfil_Tarea.Separación
        then -- ERROR
            abort Tarea_Esporádica;
        end if;
        Activación_Previa := Instante_Activación;
    end Operaciones_Previas;

-- *************************************************
    procedure Actividad_Esporádica is

    begin
        . . .
    end Actividad_Esporádica;

-- *************************************************
    procedure Operaciones_Posteriores is
    begin
        null;
    end Operaciones_Posteriores;

-- *************************************************
begin -- Tarea_Esporádica
    Operaciones_Iniciales;
    loop
        Operaciones_Previas;
        Actividad_Esporádica;
        Operaciones_Posteriores;
    end loop;
end Tarea_Esporádica;
```

En relación al análisis de planificabilidad, cabe hacer las mismas consideraciones que en el apartado anterior. A este respecto, es importante asegurar que no se traten dos eventos con una separación menor que la indicada en el perfil de la tarea esporádica.

6.2.2 Comunicación entre tareas.

La comunicación entre tareas es una funcionalidad necesaria para desarrollar sistemas de tiempo real concurrentes. En la sección 4.3 se presentó el problema de la inversión de prioridades, que acontece en estas circunstancias y que puede impedir que las tareas involucradas cumplan sus plazos de respuesta. El protocolo del techo de prioridad soluciona este problema, ya que acota la duración de la inversión de prioridades, limita el

número de inversiones de prioridades que sufre una tarea durante su ejecución e impide los interbloqueos. El protocolo de herencia inmediata del techo es una variante de este método, según la cuál cuando una tarea inicia el acceso a la región crítica, hereda el techo de prioridad. Por sus ventajas, esta es la variante más adecuada para aplicaciones prácticas.

Este protocolo se puede realizar según dos enfoques, que se diferencian en la naturaleza del mecanismo de interacción entre las tareas:

- *Pasivo*: el objeto que gestiona el acceso a la región crítica no tiene un flujo de control propio. Las tareas interaccionan con él mediante llamadas a subprogramas que se ejecutan como parte del flujo de control de la tarea llamante.
- *Activo*: el objeto gestor tiene un flujo de control propio. En estos casos la comunicación suele estar basada en el modelo cliente-servidor y se realiza mediante en intercambio de mensajes.

A continuación se analizan estos dos enfoques por separado.

Mecanismos de comunicación pasivos.

Los mecanismos pasivos de comunicación han sido tradicionalmente los más utilizados y están especialmente indicados para sistemas monoprocesador. La realización práctica habitual de semáforos, monitores, etc., sigue este principio. Se basan en un mecanismo de exclusión mutua que permite bloquear el acceso a la región crítica cuando está siendo accedida por una tarea. El interfaz que proporcionan suele ser un conjunto de procedimientos o funciones. El código de acceso puede estar incluido en estos objetos, como ocurre en un monitor, o no, como en el caso de comunicación mediante semáforos.

En este contexto, la realización del protocolo del techo de prioridad implica que una tarea, durante el acceso al recurso compartido, se ejecuta con el techo de prioridad correspondiente. La gran utilidad de este enfoque ha motivado que las normas más recientes de lenguajes o núcleos de ejecución para sistemas de tiempo real crítico incluyan mecanismos pasivos de comunicación entre tareas basados en este protocolo. El enfoque que se sigue consiste en declarar el techo de prioridad y cuando una tarea accede al recurso, las operaciones de bloqueo del acceso al recurso y modificación de la prioridad de la tarea se realizan atómicamente. Además se comprueba que la tarea no tiene una prioridad mayor que el techo, lo que constituiría una violación del protocolo y por tanto un fallo de ejecución.

Dos ejemplo destacados de este enfoque son Ada 9X y POSIX:

- En Ada 9X [Ada9XRM93] [Ada9XIntro93] se introduce el concepto de objeto protegido que encapsula y sincroniza el acceso a los datos privados de los objetos del tipo. Su diseño está basado en el monitor y en las variables condición. El acceso a los datos se realiza mediante un conjunto de subprogramas y la implementación garantiza que las llamadas a éstos se ejecutan con exclusión mutua. A un objeto

protegido se les puede definir un techo de prioridad. Cuando una tarea llama a uno de estos subprogramas, se ejecutará con esta prioridad. Si la prioridad de la tarea es mayor que el techo de prioridad del objeto protegido, se eleva una excepción.

- En POSIX [POSIX.1b 93] [POSIX.1c 93] existen los conceptos de tarea pesada (*process*) y de tarea ligera (*thread*). Una tarea pesada está compuesta por un espacio de direcciones, una o más tareas ligeras que se ejecutan en este espacio y un conjunto de recursos del sistema. Una tarea ligera es un flujo de control autónomo dentro de una tarea pesada. POSIX proporciona diversos mecanismos de comunicación entre tareas. Dos de estos mecanismos son específicos para comunicar tareas ligeras:
  - *Mutex*: Es un objeto que permite serializar el acceso de múltiples tareas ligeras a datos compartidos. Es posible declarar objetos de este tipo para que se comporten según el protocolo del techo de prioridad inmediato. Entonces, cuando una tarea ligera ejecuta dentro de la región crítica protegida por un objeto de este tipo, se ejecuta incondicionalmente con su techo de prioridad.
  - Variables condición. Es un objeto que permite bloquear a una tarea ligera hasta que se cumple una codición determinada.

En aquellos sistemas en los que no se dispone de mecanismos de comunicación pasivos con esta funcionalidad, el programador puede emular este protocolo elevando la prioridad de la tarea al valor del techo de prioridad antes de intentar acceder a los datos compartidos. Es importante asegurar que estas dos operaciones se ejecuten en este orden. Si no fuera así, podría expulsarse a la tarea antes de cambiar la prioridad, pero después de bloquear el recurso. En este caso la realización del protocolo sería incorrecta y podrían producirse interbloqueos.

Mecanismos de comunicación activos.

La utilización de una tarea para serializar el acceso a datos compartidos es recomendable en determinadas ocasiones, como cuando:

- no se disponen de mecanismos pasivos de comunicación entre tareas. Esto ocurre en Ada, donde el mecanismo de comunicación entre tareas es la cita, que es un mecanismo activo. La realización práctica de mecanismos como los semáforos se realiza mediante tareas.

- no se quiere comunicar tareas por memoria compartida.

En estas circunstancias, la comunicación se realiza según el modelo cliente-servidor. La tarea servidora exporta una serie de servicios, que cuando son solicitados por los clientes se ejecutan con exclusión mutua, para garantizar la consistencia de los datos. La prioridad de la tarea servidora debe ser el techo de prioridad de las tareas cliente. A efectos del análisis de planificabilidad, el enfoque más adecuado consiste en no incluir a la tarea servidora como una más del sistema. El tiempo de cómputo consumido en la ejecución de los servicio se contabiliza a las tareas clientes que los soliciten.

6.2.3 Detección de fallos de ejecución.

El objetivo de este apartado es estudiar los mecanismos de detección de fallos en el comportamiento temporal de las tareas y desarrollar una estructura de programa que encapsule los fallos para que se traten adecuadamente y no interfieran en la ejecución de otras tareas.

Tiempo de cómputo.

El análisis de planificabilidad comprueba si un conjunto de tareas, con unos requisitos de utilización de procesador conocidos, cumplirán sus plazos de respuesta. El tiempo de cómputo máximo de las tareas es un parámetro básico en este cálculo. En este contexto, ocurre un fallo de ejecución cuando una tarea emplea durante una activación más tiempo de cómputo que el considerado en el análisis. Es importante detectar este evento, pues puede provocar que tareas menos prioritarias que la errónea incumplan sus plazos de respuesta, aún cuando su comportamiento sea correcto.

La contabilidad del tiempo de cómputo de las tareas y el establecimiento de temporizadores basados en este concepto de tiempo, son dos requisitos necesarios para asegurar que una tarea no se excede en el uso del procesador. Esta funcionalidad no está habitualmente disponible en los entornos de ejecución de sistemas tiempo real, lo cúal es especialmente grave si se tiene en cuenta que es complicado añadir esta funcionalidad sin modificar el código del núcleo de ejecución del entorno.

El esquema de programación de temporizadores relacionados con esta noción de tiempo difiere del que se emplea con temporizadores relativos al reloj del sistema. En este caso, cuando una tarea programa un temporizador, queda bloqueada esperando su expiración. Cuando este evento ocurre, la tarea pasa de bloqueada a preparada y el planificador decide posteriormente cuando se debe ejecutar. Sin embargo, cuando una tarea programa un temporizador respecto al tiempo de cómputo, no tiene sentido que se bloquee esperando su expiración. La tarea continuará ejecutándose y cuando se produce este evento habrá dos flujos de ejecución activos dentro de una tarea, correspondientes al código específico de la activación y a la rutina de tratamiento del temporizador. En este caso, el comportamiento conveniente es ejecutar cuanto antes la rutina de tratamiento, dado que la expiración del temporizador es síntoma de un fallo de ejecución. Este cambio súbito del flujo de control de una tarea se denomina genéricamente transferencia asíncrona de control. En sistemas basados en UNIX, como es la norma POSIX, este comportamiento se puede programar mediante señales. En Ada no se dispone de este mecanismo. Esta carencia se ha corregido en Ada 9X y el código para programar esta funcionalidad se ha presentado anteriormente.

El interés de disponer de mecanismos para contabilizar el tiempo de cómputo de las tareas ha motivado su inclusión en el estándar POSIX [POSIX.1b 93], que proporciona primitivas para definir un temporizador asociado a una tarea según esta noción de tiempo. Como se comentó anteriormente, en diversas versiones del anexo de tiempo real de Ada 9X también se había definido un paquete con una funcionalidad muy similar, aunque al final se ha eliminado del lenguaje. En los esquemas de tareas que se presentan a continuación se emplea este mecanismo.

A partir de estos requisitos funcionales se puede diseñar una tarea que detecte por sí sola el exceso de uso del procesador. Sin embargo, en determinadas ocasiones esta no es la mejor solución. En el siguiente apartado se analiza esta cuestión en relación a los plazos de respuesta. El mismo razonamiento puede aplicarse a este caso.

Plazo de respuesta

Las tareas de tiempo real deben completar su activación durante el intervalo definido por sus plazos de respuesta. Es fundamental comprobar en tiempo de ejecución el cumplimiento de este requisito. Un adecuado control del tiempo de cómputo empleado por las tareas sería teóricamente suficiente, si la medida de este parámetro ha sido correcta, si las tareas se activan cuando deben y si los detalles de bajo nivel, como interrupciones, cambios de contexto, etc., se han tratado adecuadamente en el análisis de planificabilidad. Como siempre es posible que alguno de estos condicionantes no se cumpla, es recomendable comprobar el cumplimiento del plazo de respuesta de las tareas.

Por otro lado, si el núcleo de ejecución no contabiliza el tiempo de cómputo de las tareas, la comprobación del plazo de respuesta será el único medio de tener una cierta confianza de que el comportamiento temporal de las tareas es el adecuado. Sin embargo, esta prueba por sí sóla ni permite asegurar que el comportamiento temporal de las tareas es el deseado, ni que la tarea que incumple un plazo sea la causante del fallo. En efecto, puede ocurrir que algunas tareas empleen más tiempo de cómputo que el supuesto y que esta sobreutilización se compense, porque otras tareas empleen menos tiempo de cómputo que el máximo. La consecuencia es que todas las tareas del sistema cumplirían sus plazos. Por otro lado, en este escenario y dado que el método es estable, podría ocurrir que las tareas menos prioritarias incumplan sus plazos de respuesta, siendo su comportamiento correcto. Este fallo estaría inducido por el exceso de tiempo de cómputo de tareas más prioritarias. Un ejemplo de esta situación se presenta en la figura 6.1. En ella se presenta la ejecución de tres tareas con los siguientes perfiles de ejecución:

Tarea	Período	Tiempo de cómputo	Plazo de respuesta	Prioridad
τ_1	10	3	10	3
τ_2	15	5	15	2
τ_2	30	11	30	1

Este conjunto es planificable. Esto se puede comprobar fácilmente en la figura, ya que todas las tareas cumplen su primer plazo de ejecución y, en estas activaciones, consumen el tiempo de cómputo máximo. En su última activación la tarea τ_1 consume más tiempo de cómputo que el considerado y cumple su plazo de respuesta. Sin embargo, la tarea τ_3 no lo hará aunque haya funcionado corréctamente.

En cualquier caso, en las circunstancias reseñadas la comprobación del cumplimiento del plazo de respuesta de las tareas es la única razonable que se puede realizar.

La acción que se debe tomar cuando se detecta que una tarea ha incumplido un plazo de respuesta es detenerla inmediatamente y ejecutar las acciones de tratamiento del fallo. Sin

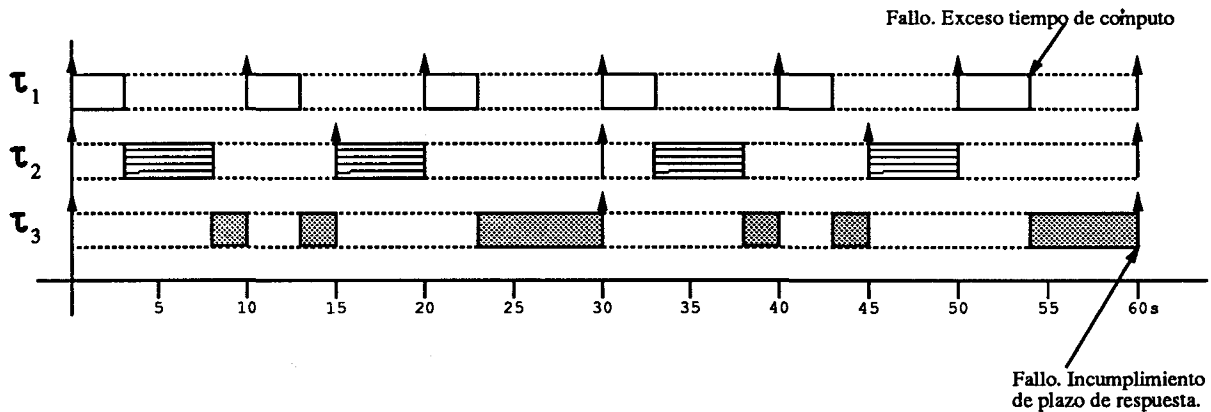

Figura 6.1: Fallo de temporización.

embargo, la realización adecuada de esta operación no es evidente. Existen dos opciones básicas:

- La propia tarea controla el cumplimiento de sus plazos de respuesta. Para tal fin, se necesita un mecanismo de transferencia asíncrona de control ligado a un temporizador relativo al reloj del sistema. Cuando el temporizador expira la tarea abandona la ejecución del código de la activación y ejecuta la rutina de tratamiento del fallo.

  Este enfoque es eficiente y permite realizar las tareas de tiempo real con una sóla tarea del núcleo. Sin embargo, en determinadas situaciones puede haber problemas si la rutina de tratamiento de la temporización se ejecuta con la misma prioridad que el código original de la tarea. En efecto, como el sistema es estable, cuando está sobrecargado la tarea menos prioritaria incumplirá su plazo de respuesta y en estas circunstancias puede transcurrir bastante tiempo hasta que se trate el fallo. En sistemas críticos en seguridad, puede ser vital tratar el fallo inmediatamente después de su detección.

- Las tareas de tiempo real se realizan mediante dos tareas físicas: la tarea funcional, que ejecuta la actividad propia de la aplicación, y la tarea supervisora, que controla el cumplimiento de los plazos de respuesta y se ejecuta con prioridad alta, generalmente mayor que la de cualquier otra tarea funcional. La tarea funcional debe comunicar a la tarea supervisora el final de una activación antes del plazo de respuesta. En caso contrario, la tarea supervisora deberá abortarla inmediatamente y ejecutar la rutina de tratamiento.

  Este enfoque es más complejo de realizar prácticamente, introduce más sobrecarga al sistema e incumple el requisito de que el código de la activación de una tarea se debe ejecutar sin interrupción y con la prioridad adecuada. Este problema es importante y se analiza en detalle posteriormente. Por otro lado, es la opción más útil en sistemas criticos en seguridad y en entornos de desarrollo en los que no se dispone de mecanismos de transferencia asíncrona de control.

Estos enfoques no son los únicos posibles. Hay alternativas intermedias como sería disponer de una tarea supervisora que controlara el cumplimiento de los plazos de respuesta de todas las tareas del sistema. En el apartado en el que se detalla la realización práctica sólo se presentarán las dos opciones anteriormente analizadas. Otras alternativas pueden ser fácilmente realizadas basándose en estas dos estructuras.

Encapsulamiento de los fallos.

Una cuestión de gran importancia es asegurar que los fallos de ejecución de una tarea no se transmitan fuera de su ámbito. La forma de conseguir este objetivo es asegurando que una tarea detectará todos los fallos de ejecución que ella provoque. En el siguiente apartado se muestra como se puede conseguir fácilmente este objetivo con el lenguaje de programación Ada.

Esquema de realización.

Las soluciones que se van a proporcionar a continuación son aplicables tanto a tareas periódicas como esporádicas. Por consiguiente, en lo sucesivo sólo se presenta el código correspondiente a las tareas periódica.

Esquema con una tarea. En este apartado se presenta un esquema de tarea periódica que comprueba el cumplimiento de sus restricciones temporales. Esta funcionalidad se consigue mediante el empleo de la transferencia asíncrona de control y temporizadores respecto al tiempo de cómputo de las tareas.

El paquete `Global` incorpora un campo adicional a la definición del perfil de la tarea, que se denomina `T_Cómputo` y representa el tiempo de cómputo asignado a la tarea en cada activación. El tipo del parámetro es `Accounting_Unit`, de acuerdo con las definiciones y la estructura del paquete que proporciona los temporizadores sobre el tiempo de cómputo.

```
with System;
with Calendar;
with CPU_Time_Accounting;

package Global is

    type Tipo_Perfil_Periódico is record
        Período         :  Duration;
        Fase            :  Duration;
        Prioridad       :  System.Any_Priority;
        T_Cómputo       :  CPU_Time_Accounting.Accounting_Range;
        Plazo_Respuesta :  Duration;
    end record;

    Instante_Inicio :  Calendar.Time := . . .;
end Global;
```

En el paquete que incluye la tarea periódica, se añade la declaración de `Contador_T_Cómputo` para establecer temporizadores respecto al tiempo de cómputo de la tarea.

```
package Tarea_Periódica is
end Tarea_Periódica;


with CPU_Time_Accounting;
with Global;

package body Tarea_Periódica is
   Perfil_Tarea   : Global.Tipo_Perfil_Ejecución :=
      (Período=> 0.1, Fase=> 1.0, Prioridad=> 3,
      T_Cómputo=> 0.01, Plazo_Respuesta=> 0.1);

   Contador_T_Cómputo :  CPU_Time_Accounting.Overrun_Signal;

   task Tarea_Periódica is
   end Tarea_Periódica;
   task body Tarea_Periódica is separate;

end Tarea_Periódica;
```

La estructura del cuerpo de la tarea se ha modificado ligeramente para programar los controles de detección de fallos de temporización. Además, para encapsular los fallos que se detecten, se incluye un manejador de excepciones que permite tratar cualquier excepción que se produzca durante la ejecución de esta tarea y no se trate. El tratamiento del fallo consiste en abortar la tarea, para evitar que cause más daños. En secciones posteriores se presenta un modelos de tratamiento de fallos más elaborado. La estructura de la tarea es como sigue:

- `Operaciones_Iniciales`. Además de las operaciones de inicio anteriores, se asocia la tarea periódica al objeto `T_Cómputo`. De esta forma los temporizadores programados sobre este objeto cuentan el tiempo de código ejecutado por esta tarea.

- `Operaciones_Previas`. La única operación dentro de este procedimiento es la programación del contador de tiempo de cómputo. El valor inicial del contador es el tiempo de cómputo de la tarea.

- *Activación de la tarea.* La detección de los errores de temporización ha motivado que la llamada al procedimiento `Actividad_Periódica` no se haga directamente. Los errores de temporización se detectan mediante dos sentencias de transferencia de control anidadas. La primera asegura que no se consume más tiempo de cómputo que el asignado y la segunda comprueba el cumplimiento de los plazos de respuesta. La parte asíncrona de la primera es una llamada al punto de entrada `Wait` del objeto `Contador_T_Cómputo`. Esta llamada se acepta cuando la tarea ha consumido más tiempo de cómputo que el especificado en el procedimiento `Operaciones_Previas`, que coincide con el tiempo de cómputo de la tarea.

Un detalle que se debe tener en cuenta, es que el tiempo de cómputo de Operaciones_Posteriores y Operaciones_Previas, no se considera al realizar el control de este parámetro. Por consiguiente, si se quiere afinar al programar el temporizador anerior, el tiempo de cómputo del perfil de ejecución debería ser menor que el real de la tarea, al descontar el valor cosumido por estos procedimientos.

La parte asíncrona de la segunda sentencia es un retardo absoluto con el valor del plazo de respuesta. Si este retardo expira antes de completarse la sentencia de transferencia asíncrona será porque la tarea no ha cumplido el plazo de respuesta. La tarea se ejecuta corréctamente si termina la activación antes que se acepte la llamada al punto de entrada o antes del tiempo de respuesta.

- Operaciones_Posteriores. Estas operaciones se limitan a programar el instante de la siguiente activación.

```
with Calendar;
with Ada.Dynamic_Priorities;

separate (Tarea_Periódica)
task body Tarea_Periódica is
    Siguiente_Activación :  Calendar.Time;

    -- ****************************************************
    procedure Tratar_Fallo(Fallo :  Tipo_Fallo) is

    begin
        abort Tarea_Periódica;
    end Tratar_Fallo;
-- ****************************************************

    procedure Operaciones_Iniciales is

    begin
        CPU_Time_Accounting.Attach_Overrun_Signal
                (Tarea_Periódica, Contador_T_Cómputo);
        Dynamic_Priorities.Set_Priority(Perfil_Tarea.Prioridad, Tarea_Periódica);
        Siguiente_Activación := Global.Instante_Inicio + Perfil_Tarea.Fase;
        delay until Siguiente_Activación;
    end Operaciones_Iniciales;
-- ****************************************************
    procedure Operaciones_Previas is

    begin
        CPU_Time_Accounting.Set_Value(Tarea_Periódica, Perfil_Tarea.Contador_T_Cómputo);
    end Operaciones_Previas;

-- ****************************************************
    procedure Actividad_Periódica is

    begin
        . . .
    end Actividad_Periódica;
```

```
-- ***************************************************
   procedure Operaciones_Posteriores is
   begin
      Siguiente_Activación := Siguiente_Activación + Perfil_Tarea.Período;
      delay until Siguiente_Activación;
   end Operaciones_Posteriores;

-- ***************************************************
begin -- Tarea_Periódica
   Operaciones_Iniciales;
   loop
      Operaciones_Previas;

      select
         Contador_T_Cómputo.Wait;
         Tratar_Fallo;
      then abort
         select
            delay until Siguiente_Activación + Perfil_Tarea.Plazo_Respuesta;
            Tratar_Fallo;
         then abort
            Actividad_Periódica;
         end select;
      end select;

      Operaciones_Posteriores;
   end loop;

exception
   when others =>
      Tratar_Fallo;
end Tarea_Periódica;
```

Este esquema mantiene la propiedad de ejecutar de forma continuada el código correspondiente a una activación y no modificar la prioridad de ejecución de la tarea. Desde el punto de vista del análisis de planificabilidad, la única diferencia respecto al caso anterior es que el esquema introduce sobrecarga mayor. Al calcular valor del tiempo de cómputo más desfavorable hay que considerar el caso en que se detecta un fallo de ejecución, por el tiempo que se consume al tratarlo.

Esquema basado en dos tareas. El enfoque basado en una tarea presenta el inconveniente de que en caso de fallo de ejecución puede transcurrir bastante tiempo hasta que se ejecute la rutina de tratamiento. Esto ocurre claramente en el caso que se produzca una sobrecarga transitoria y sea la tarea más prioritaria la que falle.

Un enfoque alternativo es utilizar dos tareas: una tarea funcional, que ejecuta el código específico de la aplicación y una tarea supervisora que controla el cumplimiento de los requisitos temporales de la primera. A continuación se presenta una relización basada en este concepto.

En el paquete `Global` se añade la declaración del tipo `Tipo_Ejecución` que representa el estado de la ejecución de una activación de una tarea de tiempo real. Este tipo se emplea para que la tarea funcional informe a la supervisora del resultado de la activación.

```
with System;
with Calendar;
with CPU_Time_Accounting;

    package Global is

        type Tipo_Ejecución is (Correcta, Fallo_Interno, Fallo_Externo);

        type Tipo_Perfil_Periódico is record
            Período         :  Duration;
            Fase            :  Duration;
            Prioridad       :  System.Priority;
            T_Cómputo       :  CPU_Time_Accounting.Accounting_Range;
            Plazo_Respuesta :  Duration;
        end record;

        Instante_Inicio :  Calendar.Time := .  .  .  ;

    end Global;
```

El paquete que encapsula el código de una tarea periódica incluye ahora la declaración de dos tareas: la tarea funcional y la tarea supervisora. En la especificación de la tarea funcional se incluye un punto de entrada para la comunicación con la tarea supervisora.

```
    package Tarea_Periódica is
    end Tarea_Periódica;


    with Global;
    with CPU_Time_Accounting;

    package body Tarea_Periódica is
        Perfil_Tarea    :  Global.Tipo_Perfil_Ejecución :=
            (Período=> 0.1, Fase=> 1.0, Prioridad=> 3,
            T_Cómputo=> 0.01, Plazo_Respuesta=> 0.1);

        Contador_T_Cómputo :  CPU_Time_Accounting.Overrun_Signal;

        task Tarea_Supervisora is
        end Tarea_Supervisora;
        task body Tarea_Supervisora is separate;

        task Tarea_Funcional is
            entry Fin_Activación(Ejecución :  out Global.Tipo_Ejecución);
        end Tarea_Funcional;
        task body Tarea_Funcional is separate;
```

```
end Tarea_Periódica;
```

Tarea funcional. La tarea funcional incluye el código de la tarea relacionado con su funcionalidad específica y se ejecuta de acuerdo a su período y con la prioridad asignada por el método. En el código es la tarea denominada `Tarea_Funcional`. En este modelo se han eliminado las sentencias de transferencia de control asíncronas, ya que la tarea supervisora es la que debe realizar esta funcionalidad. La tarea funcional incluye citas con la supervisora para notificarla el estado de la ejecución y la detección de errores funcionales. En cada activación, la tarea funcional debe aceptar la cita con la tarea supervisora antes de que se cumpla el plazo de respuesta. Si no, ésta detectara el fallo y tomará las medidas oportunas. Por otro lado, la acción de tratamiento de excepciones también consiste en citarse con la tarea supervisora, pero indicando ahora que se ha producido un fallo de ejecución.

La funcionalidad de los procedimientos del código de la tarea es como sigue:

- `Operaciones_Iniciales`. El código inicial de la tarea periódica es igual que en el esquema anterior.
- `Operaciones_Previas`. No hay operaciones previas. La programación del temporizador respecto al tiempo de cómputo lo realiza la tarea supervisora.
- `Operaciones_Posteriores`. La tarea funcional debe indicar a la tarea supervisora el final de cada activación, que ocurre cuando termina de ejecutarse `Actividad_Periódica`. Este funcionalidad se incluye en este procedimiento. Esta llamada tiene un parámetro que indica si la activación se ha ejecutado satisfactoriamenmte.

```
with Calendar;
with Ada.Dynamic_Priorities;

separate (Tarea_Periódica)
task body Tarea_Funcional is
   Siguiente_Activación :  Calendar.Time;

-- ***************************************************

   procedure Operaciones_Iniciales is

   begin
      CPU_Time_Accounting.Attach_Overrun_Signal
            (Tarea_Funcional, Contador_T_Cómputo);
      Ada.Dynamic_Priorities.Set_Priority(Perfil_Tarea.Prioridad, Tarea_Funcional);
      Siguiente_Activación := Global.Instante_Inicio + Perfil_Tarea.Fase;
      delay until Siguiente_Activación;
   end Operaciones_Iniciales;
-- ***************************************************
   procedure Operaciones_Previas is
```

```
    begin
        null;
    end Operaciones_Previas;

-- ***************************************************
    procedure Actividad_Periódica is

    begin
        . . .
    end Actividad_Periódica;

-- ***************************************************
    procedure Operaciones_Posteriores is
    begin
        accept Fin_Activación(Ejecución) do
            Ejecución := Global.Correcta;
        end Fin_Activación;
        Siguiente_Activación := Siguiente_Activación + Perfil_Tarea.Período;
        delay until Siguiente_Activación;
    end Operaciones_Posteriores;

-- ***************************************************
begin -- Tarea_Funcional
    Operaciones_Iniciales;
    loop
        Operaciones_Previas;
        Actividad_Periódica;
        Operaciones_Posteriores;
    end loop;

exception
    when others =>
        accept Fin_Activación(Ejecución) do
            Ejecución := Global.Fallo_Interno;
        end Fin_Activación ;
end Tarea_Funcional;
```

Tarea Supervisora. La tarea supervisora es la encargada de controlar la ejecución de la tarea funcional. Su estructura es la misma que ésta. La detección de los fallos de temporización se basa en dos sentencias de transferencia asíncrona de control anidadas que siguen la misma estructura y funcionalidad que en el caso anterior. La parte abortable de la segunda sentencia es una llamada al punto de entrada de la tarea funcional. Si esta llamada no se acepta antes de que transcurra el plazo de respuesta o antes de que la tarea funcional consuma más tiempo de cómputo que el asignado, entonces se habrá producido un fallo de temporización. Como consecuencia, se ejecutará la rutina de tratamiento de fallos y se completará la ejecución de la tarea. La llamada al punto de entrada `Fin_Activación` de la tarea funcional incorpora un parámetro para indicar si la ejecución de la activación ha sido correcta o si se han producido fallos de ejecución.

El tratamiento de los fallos de ejecución es independiente de cúal sea éste, por lo que

se ha definido el procedimiento Tratar_Fallo. En este esquema, este procedimiento aborta a la tarea funcional, si el fallo ha sido detectado por la tarea supervisora. Si el fallo lo ha comunicado la tarea funcional, no será necesario realizar acción alguna, ya que ésta termina por sí sóla.

La programación de las activaciones es diferente al caso de la tarea funcional. En la tarea funcional se programa el instante de la siguiente activación. Cuando llega, la tarea pasa a lista y se ejecuta cuando el planificador la seleccione. Un esquema similar supondría que la tarea supervisora comenzaría a ejecutar en el instante de activación, ya que su prioridad es prácticamente la mayor del sistema. Este esquema se ha simplificado. La tarea supervisora sólo ejecuta cuando la funcional acepta su llamada o cuando detecta un fallo de temporización. De esta forma se reduce la sobrecarga que introduce la tarea supervisora.

En relación al tiempo de cómputo del conjunto, nótese que el temporizador que controla este parámetro sólo referencia al consumido por la tarea funcional. La sencillez del código de la tarea supervisora no requiere su control. En cualquier caso hay que tener en cuenta que el tiempo de cómputo que se emplea en el análisis de planificabilidad difiere del valor en el perfil de ejecución. El primero es la suma de los tiempo de cómputo de la tarea supervisora y de la funcional, mientras que el segundo sólo contempla este segundo valor.

```
with Calendar;
with Ada.Dynamic_Priorities;

separate (Tarea_Periódica)
task body Tarea_Supervisora is

    Fallo_T_Funcional :  Boolean := False;
    Result_Ejecución :  Global.Tipo_Ejecución := Global.Correcta;
    Siguiente_Activación :  Calendar.Time;

    -- **************************************************

    procedure Tratar_Fallo(Fallo_T_Funcional :  in Boolean;
                Result_Ejecución :  in Global.Tipo_Ejecución) is

    begin
        if Fallo_T_Funcional then
            abort Tarea_Funcional;
        end if;
    end Tratar_Fallo;

    -- **************************************************

    procedure Operaciones_Iniciales is

    begin
        Ada.Dynamic_Priorities.Set_Priority(System.Priority'Last - 1, Tarea_Supervisora);
        Siguiente_Activación := Global.Instante_Inicio + Perfil_Tarea.Fase;
    end Operaciones_Iniciales;
```

```
-- ***************************************************

begin -- Tarea_Supervisora

    Operaciones_Iniciales;

    loop

        CPU_Time_Accounting.Set_Value(Tarea_Funcional, Perfil_Tarea.T_Cómputo);

        select
            Contador_T_Cómputo.Wait;
            Fallo_T_Funcional := True;
            Tratar_Fallo(Fallo_T_Funcional, Result_Ejecución);
        then abort
            select
                delay until Siguiente_Activación + Plazo_Respuesta;
                Fallo_T_Funcional := True;
                Tratar_Fallo(Fallo_T_Funcional, Result_Ejecución);
            then abort
                Tarea_Funcional.Fin_Activación(Result_Ejecución);
                if Result_Ejecución /= Global.Correcta then
                    Tratar_Fallo(Fallo_T_Funcional, Result_Ejecución);
                end if;
            end select;
        end select;

        exit when Fallo_T_Funcional or Result_Ejecución /= Global.Correcta ;

        Siguiente_Activación := Siguiente_Activación + Perfil_Tarea.Período;

    end loop;

exception
    when others =>
        Fallo_T_Funcional := True;
        Tratar_Fallo(Fallo_T_Funcional, Result_Ejecución);
end Tarea_Supervisora;
```

Análisis de planificabilidad. El comportamiento del conjunto de las tareas funcional y supervisora respecto al análisis de planificabilidad es algo más complejo. El código de la activación se ejecuta sin bloqueos. La tarea funcional ejecuta la activación y cuando la completa avisa a la tarea supervisora y, como consecuencia, la desbloquea. Ésta ejecutaría inmediatamente, por ser más prioritaria, y cuando se vuelve a bloquear, la tarea funcional completa la activación. Por consiguiente, el conjunto ejecuta sin bloquearse.

La tarea supervisora se ejecuta con una prioridad mayor que la indicada en el perfil. En esta situación se produce inversión de prioridades, que afecta a las tareas con prioridad intermedia entre la tarea funcional y la tarea supervisora. La duración de la inversión está acotada y coincide con el tiempo de cómputo de la tarea supervisora en cada activación.

En ausencia de fallos, ésta ejecuta como respuesta a una acción de la tarea funcional. Esto mismo ocurre cuando se accede a un objeto de comunicación basado en el protocolo de la prioridad techo. Por tanto, como se mostará en la sección 6.5 esta inversión de prioridades se puede asimilar al caso de comunicación y tratarse con los métodos de análisis de planificabilidad conocidos.

Sin embargo, este panorama cambia si la tarea supervisora se ejecuta como consecuencia de un fallo de temporización, dado que ahora la tarea ejecuta autonomamente. La duración de la inversión de prioridades está acotado, pero no se puede garantizar que las tereas más prioritarias se verán bloqueadas una sóla vez en cada activación. En conclusión, el caso peor de inversión de prioridades para la tarea periódica más prioritaria será cuando todas las tareas menos prioritarias fallen su ejecución a la vez. La consideración de este caso puede ser problemática ya que puede suponer una considerable pérdida de planificabilidad, especialmente en el caso en que la carga del procesador sea elevada. Este problema se trata con más detalle posteriormente.

6.3 Cambio de modo.

6.3.1 Planteamiento.

El objetivo de esta sección es desarrollar un algoritmo de cambio de modo basado en el protocolo presentado en la sección 4.5 y en el apéndice A. Este protocolo es síncrono, en el sentido que cuando cambia el modo del sistema, las tareas cambian su perfil de ejecución por sí solas. Este enfoque permite que las tareas cambien escalonadamente y con sus datos en estado consistente. La desventaja que presenta es el retardo que se produce desde que las tareas comienzan a ejecutar el protocolo hasta que todas terminan. Consiguientemente, este protocolo es utilizable en sistemas con requisitos no contradictorios con este proceder.

Una alternativa a este enfoque es el cambio de modo asíncrono. En este caso, cuando cambia el modo del sistema, todas las tares se abortan inmediatamente y comienza la ejecución con el perfil correspondiente al modo nuevo. Esta funcionalidad se suele realizar mediante transferencia asíncrona de control. El retardo en la aplicación de este protocolo es prácticamente cero. Sin embargo, las tareas deben aplicar procedimientos especiales para mantener la consistencia de los datos, lo que en ocasiones puede ser complicado.

Cuando se produce un cambio de modo, se comunica a las tareas del sistema, las cuáles abandonan inmediatamente la ejecución en curso y cambian su perfil de ejecución de acuerdo al nuevo. Para programar esta actividad se utilizan mecanismos de transferencia asíncrona de control.

En el resto de esta sección se analizarán las particularidades del diseño y realización del protocolo síncrono y posteriormente se presenta un algoritmo integrado con los esquemas de tareas anteriores.

6.3.2 Cambio de modo síncrono. Consideraciones de diseño.

Objetivos del protocolo de cambio de modo.

El protocolo de cambio de modo establece los mecanismos precisos para que las tareas siempre ejecuten según el modo del sistema. En concreto, enuncia las condiciones que se deben cumplir para que las tareas cambien su perfil y comienzen la ejecución en el modo nuevo. Los objetivos principales de este protocolo son:

- Mantenimiento de la consistencia de los datos.
- Ejecución de las tareas de acuerdo al modo del sistema.
- Cumplimiento de los plazos de respuesta durante y después de un cambio de modo.
- No interferir con la ejecución de las tareas cuyo perfil no varía durante el cambio de modo.

Por otro lado, no hay que olvidar que un objetivo básico de los esquemas de tareas que se están proponiendo en esta sección es que se basen en entornos de ejecución comerciales y normalizados, los cuales no proporcionan mecanismos orientados al cambio de modo síncrono.

En las secciones siguientes se analizarán los aspectos más relevantes del protocolo del cambio de modo a la luz de estos objetivos. En concreto, se deben tratar las siguientes cuestiones:

- Determinación de los retardos de activación.
- Detección del cambio de modo.
- Estructura de las tareas.
- Separación entre cambios de modo.
- Comunicación entre tareas.
- Requisitos al núcleo de ejecución.

Consideraciones de diseño.

Determinación de los retardos de activación. El inicio de la ejecución de las tareas en el modo nuevo, es un factor clave para garantizar el cumplimiento de los plazos de respuesta durante el cambio de modo. Este instante de tiempo depende del estado de ejecución de la tarea:

- Si la tarea estaba inactiva en el modo original, la primera activación es la suma del instante de cambio de modo y un retardo a determinar.

- Si la tarea estaba activa y no había comenzado a ejecutar en la activación, el instante de inicio se determina como en el caso anterior.
- Si la tarea estaba activa y había comenzado a ejecutar en la activación, el instante de inicio es la suma del instante en que debería comenzar su siguiente activación en el modo original y un retardo a determinar.

El valor de estos retardos no es fijo y depende de las circunstancias en que se produce el cambio de modo. El análisis de planificabilidad durante un cambio de modo (apéndice A), se determina considerando la situación más desfavorable, que para las tareas activas consiste en suponer siempre el tercer caso de los anteriores.

En la realización práctica del algoritmo, se pueden utilizar los retardos más desfavorables o determinar los exactos en tiempo de ejecución, considerando el estado preciso en que se encuentran las tareas. Aunque esta segunda opción es la óptima, su realización eficiente implica la necesidad de que el núcleo de tiempo real colabore en la ejecución de este protocolo. Dado que uno de los requisitos básicos de este trabajo es considerar que el sistema de ejecución no proporciona mecanismos especiales orientados a esta operación, la alternativa es suponer que siempre ocurre el caso más desfavorable y trabajar con los retardos de activación en el modo nuevo calculados estáticamente.

Detección del cambio de modo. El primer aspecto a tratar es cómo y cuando detectan las tareas la ocurrencia de un cambio de modo. Parece lógico mantener un objeto visible a todas las tareas, que centralize la información relativa al modo del sistema. Las tareas pueden detectar la existencia de un cambio de modo consultando a este objeto. Si el modo actual del sistema es diferente del modo de ejecución de la tarea, es evidente deducir que se ha producido un cambio de modo.

El momento en que las tareas deben detectar la existencia de un cambio de modo está implícito en el protocolo. A continuación se analizan los casos que se presentan:

- *Tareas que no se han ejecutado en la activación actual.* El protocolo establece que estas tareas (τ_1 en la figura 6.2) pueden comenzar las operaciones de cambio de modo inmediatamente. Esto quiere decir que pueden cambiar su perfil de ejecución y la capacidad de procesador asociada quedará disponible. Sin embargo, estas tareas no pueden detectar inmediatamente la ocurrencia del cambio de modo, ya que estarán preparadas para ejecutar y no se puede saber cuando lo harán.

  El análisis de planificabilidad de las tareas durante un cambio de modo, considera que el caso peor ocurre cuando todas tienen que ejecutar el código completo de la activación, aún en el caso que no se hubiera comenzado. A partir de esta suposición, se puede calcular el instante de inicio en el caso más pesimista. Esta es la opción que se va a seguir en la realización práctica del protocolo, porque es la única que permite garantizar los plazos de respuesta de las tareas. Por consiguiente, no se consideraran el caso de estas tareas y se engloban en la clase que sigue.

- *Tareas que se han ejecutado en la activación actual.* El protocolo establece que estas tareas deben completar su activación actual. El cambio del perfil de ejecu-

ción y la disponibilidad de su tiempo de cómputo, tienen lugar a partir del instante de tiempo en que debería ocurrir su siguiente activación en el modo de ejecución original. Por tanto, la detección del cambio de modo en estas tareas se debe hacer al final de la activación.

Aquellas tareas que no han completado su activación (τ_2 en la figura 6.2), al terminarla detectarán el cambio de modo y adaptarán su perfil adecuadamente. Teniendo en cuenta los esquemas de tareas periódicas presentados anteriormente, el caso realmente conflictivo lo constituyen aquellas tareas que ya han completado su activación (τ_3 en la figura 6.2). Permanecerán sin ejecutar hasta la siguiente activación, por lo que es imposible que realicen el cambio de modo antes de iniciar ésta.

La primera alternativa para solucionar esta cuestión es dejar que detecten por sí solas el cambio de modo cuando se ejecuten y actúen en consecuencia. Sin embargo, esta opción plantea algunos problemas. Si el instante de inicio de la tarea en el nuevo modo es anterior al instante en que se detecta el cambio, la tarea no cumpliría el protocolo. Si la situación es la inversa, la tarea en cuestión consumiría una cantidad de tiempo de cómputo que no le corresponde. Esto se debería tener en cuenta en el análisis de planificabilidad o podría provocar el incumplimiento de plazos de respuesta.

La segunda alternativa consiste en emplear algún medio de activar a estas tareas cuando se produce el cambio de modo. Para tal fin se emplearía una tarea supervisora encargada de despertarlas para que ejecuten el protocolo de cambio de modo. Esta opción también presenta problemas. Si se quiere que la supervisora avise a las tareas inmediatamente después de producirse un cambio de modo, su prioridad debería ser alta y por consiguiente el plazo de respuesta debería ser muy corto. La viabilidad de esta alternativa pasa por analizar la interferencia de esta tarea sobre el resto de las tareas del sistema. Aunque esta opción no es descartable completamente, la interferencia no es desdeñable dada su alta prioridad. En consecuencia, generalmente supondrá perder capacidad de procesador por una tarea que se ejecuta una vez en cada modo.

La tercera alternativa es retrasar el instante efectivo del cambio de modo, de forma que se asegure que todas las tareas le detecten autónomamente al final de una activación. De esta forma, se evitan los problemas comentados anteriormente asociados con la detección tardía del cambio. Evidentemente esta opción no es la ideal, pues se incrementa el retardo en la ejecución del cambio de modo. Sin embargo, en la mayoría de los sistemas con requisitos que permitan emplear el cambio de modo síncrono, es muy probable que este retardo no plantee problemas [CASA&91a].

La siguiente cuestión es determinar cuánto se debe retardar el cambio de modo para que todas las tareas lo detecten a tiempo. El protocolo requiere que todas las tareas hayan ejecutado las operaciones de cambio de modo antes de la activación siguiente correspondiente al modo original.

El caso más desfavorable corresponde a la tarea que más puede tardar en detectar un cambio de modo, que es la tarea con el mayor período. En efecto, si esta tarea ha

terminado la ejecución en la activación en la que se produce el cambio de modo, no volverá ejecutarse hasta la siguiente. Por tanto, para asegurar que detecta el cambio de modo sería suficiente con introducir un retraso igual a su período.

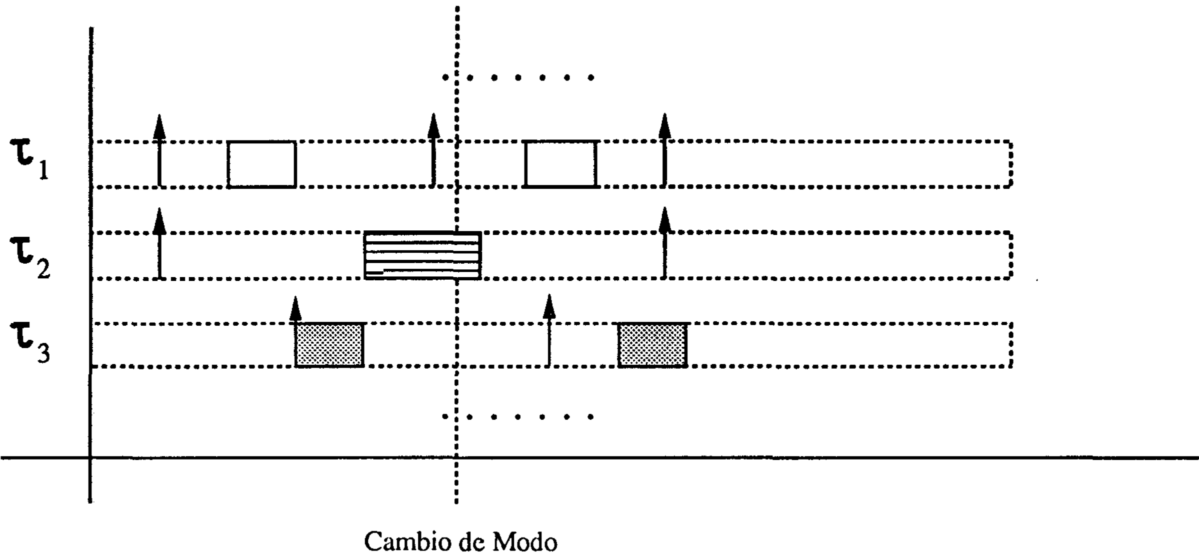

Figura 6.2: Caso más desfavorable de detección de cambio de modo.

La mayoría de las tareas detectarán el cambio de modo con bastante antelación al cambio efectivo, que es cuando deben ejecutar las operaciones del protocolo. En este intervalo, las tareas continuarán ejecutando según el perfil del modo vigente.

Es importante recalcar que la alternativa seleccionada no es la única posible ni, en determinadas ocasiones, la más adecuada. Sin embargo, es la solución más sencilla para la mayoría de los sistemas.

- *Tareas inactivas en el modo.* En un sistema con varios modos puede haber tareas que están inactivas en algunos de ellos. El comportamiento deseado es que estas tareas no se ejecuten cuando no están activas, pero que cambien su perfil adecuadamente cuando se cambie de modo. Por consiguiente es necesario disponer de mecanismos que bloqueen a una tarea en un modo y que cuando se produzca un cambio, se la desbloquee para que compruebe si en el modo nuevo se debe ejecutar o continuar inactiva.

  Un aspecto básico en el tratamiento de esta cuestión es integrar el tiempo de cómputo de estas tareas inactivas con el conjunto. En caso contrario su ejecución podría producir incumplimiento de plazos de respuesta. La solución más adecuada es disponer de una tarea gestora de las inactivas que las despierte. Esta tarea se diseñará como un servidor esporádico que se activa cuando se produce un cambio de modo. El tiempo de cómputo mínimo debe ser tal que todas las tareas inactivas puedan ejecutar las operaciones necesarias para actualizar su perfil de ejecución. El intervalo en el que se deben ejecutar estas operaciones está delimitado por la ocurrencia de un cambio

de modo y el instante efectivo. Por consiguiente, este será el plazo de respuesta de la tarea, y su prioridad será de las menores en el sistema. Su período debe ser menor o igual que la separación mínima entre cambios de modo. Teniendo en cuenta estas consideraciones, el tiempo de cómputo que se requiere, y por tanto la ocupación de procesador es pequeña.

Otra cuestión a considerar es cuál será la prioridad de las tareas inactivas cuando ejecutan las operaciones de comprobación del perfil de cambio de modo. La alternativa más adecuada es que cuando una tarea está inactiva, su prioridad se hace igual a la prioridad de la tarea supervisora de inactivas. Sin embargo, siempre se producirá una cierta inversión de prioridades que ocurrirá cuando la tarea inactiva cambia su perfil de acuerdo al modo nuevo de ejecución y por tanto eleva su prioridad. En el intervalo que transcurre desde que la tarea efectúa esta operación, hasta que se bloquea en espera de su próximo instante de activación, puede ocurrir que una tarea con prioridad intermedia se active y tenga que esperar a que esta operación se complete. En cualquier caso, el tiempo de inversión de prioridades es pequeño, acotado y asimilable a los bloqueos por comunicación entre tareas, por lo que es inmediato incluir su influencia en el análisis de planificabilidad.

Estructura de las tareas. La cuestión a tratar es cúal debe ser el esquema de las tareas para ejecutar según el perfil del modo del sistema. La alternativa más adecuada es emplear una tarea funcional y seleccionar el código a ejecutar dependiendo del modo de ejecución. Otra opción sería disponer de una tarea funcional por modo. Sin embargo, esta opción no presenta grandes ventajas y, por el contrario, aumenta el tamaño de la aplicación sin proporcionar más funcionalidad y aumenta la complejidad de la programación de los mecanismos de dormir tareas y de cambio de perfil de ejecución.

Separación entre cambios de modo. Los cambios de modo del sistema no se pueden producir tan rápidos como sean solicitados por las tareas. La separación mínima entre modos debe ser suficiente para que todas las tareas de tiempo real completen la ejecución del protocolo. En caso contrario, sería difícil determinar el comportamiento del sistema.

El cálculo del valor de las separación entre cambios no es inmediato. Estará definido por el instante más desfavorable de activación en el modo nuevo entre todas las tareas. Su determinación implica un análisis minucioso de estos valores, aunque es factible su automatización.

El valor de la separación depende del modo original y el modo al que se transita. Por tanto, si se quiere que la realización práctica del protocolo trabaje con valores precisos, sería necesario calcular la separación cuando se cambia entre cualquier pareja de modos. En este trabajo se supone que se toma el máximo entre estos valores, lo cuál será viable en la mayoría de las ocasiones.

Comunicación entre tareas. En un cambio de modo los perfiles de las tareas, y por tanto su prioridad, cambia. Consecuentemente, el techo de prioridad de los objetos de comuni-

cación también variará. En el enunciado del protocolo se proporciona las condiciones en las que el techo de prioridad de estos objetos se puede elevar. Sin embargo, esta operación aumenta la complejidad del algoritmo de cambio de modo. Como se ha indicado, una alternativa es calcular el techo de prioridad como la máxima prioridad entre las tareas que acceden al objeto de comunicación en cualquier modo del sistema. Esta medida puede aumentar ligeramente el tiempo de bloqueo de algunas tareas, pero la simplificación que provoca en el algoritmo la convierten en la opción más adecuada.

Requisitos al núcleo de ejecución. La única funcionalidad adicional requerida al núcleo, respecto a los casos anteriores, es la posibilidad de cambiar dinámicamente la prioridad del sistema.

6.3.3 Algoritmo de cambio de modo síncrono.

Teniendo en cuenta la consideraciones de diseño anteriormente planteadas, a continuación se presenta el esquema de un algoritmo de cambio de modo síncrono. Las operaciones del algoritmo se han dividido, según los objetos funcionales involucrados que deben realizarlas, como sigue:

- Gestión del modo del sistema.
    1. Una tarea ejecuta la rutina de cambio de modo. Esta operación tiene efecto si el modo al que se quiere transitar es distinto del actual o si ha transcurrido el tiempo mínimo establecido desde el anterior cambio de modo.
    2. Una vez que se ha aceptado la operación, se actualiza la variable que mantiene el valor del modo del sistema, se calcula el instante efectivo de cambio y se desbloquea a la tarea gestora de inactivas.
- Tareas de tiempo real.
    1. Las tareas del sistema consultan al final de cada activación cúal es el modo actual del sistema. Si es diferente al modo actual, es que se ha producido un cambio de modo.
    2. Después de detectar el cambio, la tarea espera hasta que el siguiente instante de activación sea mayor que el instante efectivo de cambio. Entonces procede a ejecutar el protocolo de cambio de modo.
    3. Si la tarea está inactiva en el modo nuevo se duerme. En caso contrario cambia el perfil de ejecución.
    4. Finalmente calcula el primer instante de activación en el modo nuevo, mediante la suma del siguiente instante de activación y el retardo de inicio en el modo nuevo.
- Desbloqueo de las tareas inactivas.

1. Una vez que la tarea gestora de inactivas se desbloquea, hace lo propio con las tareas inactivas y se bloquea hasta el siguiente cambio de modo.
2. Las tareas ejecutan el protocolo de cambio de modo de forma similar a las activas. La única diferencia estriba en que el instante de la primera activación se debe calcular como la suma entre el instante efectivo de cambio de modo y el retardo de inicio en el modo nuevo.

6.3.4 Esquema de realización del algoritmo.

Los objetos funcionales que realizan el algoritmo se han agrupado en tres elementos. Inicialmente se presentan las declaraciones globales al sistema. A continuación se muestran los elementos directamente relacionados con operaciones globales de gestión del modo de ejecución del sistema. Finalmente, se amplia el esquema de tarea periódica para que ejecute de acuerdo al modo del sistema.

Los parámetros relacionados con el cambio de modo indican cuando se debe activar la tarea después del cambio, cúal es la separación entre cambios o el retardo del instante de cambio. Estos parámetros se deberían definir para cada posible cambio entre cualquier par de modos del sistema. Esta opción se ha simplificado y en los esquemas de tareas se emplea el valor de estos parámetros en el caso más desfavorable. Si la situación lo requiriera habría que optar por la primera alternativa.

Declaración de tipos globales.

El paquete `Global`, que incluye las declaraciones de los tipos globales al sistema, varía pues ahora es necesario representar los modos del sistema y un perfil de ejecución por modo. El tipo enumerado `Tipo_Modo_Sistema` enumera los identificadores de los modos de ejecución del sistema y `Modo_Inicial_Sistema` el modo inicial de ejecución.

`Perfil_Ejecución` es ahora un tipo variable. El campo `Estado_En_Modo` permite discernir entre perfiles inactivos y activos. En el primer caso, no es necesaria más información para definir el perfil. En el segundo caso hay que proporcionar además la misma información que en el esquema de tareas de la sección anterior.

Finalmente, es necesario definir un perfil para cada modo de ejecución del sistema. El tipo `Tipo_Perfiles_Tarea` es un tipo formación. El índice es del tipo enumerado `Tipo_Modo_Sistema`.

```
with System;
with Calendar;
with CPU_Time_Accounting;

   package Global is
      type Tipo_Ejecución is (Correcta, Fallo_Interno, Fallo_Externo);

      type Tipo_Estado_En_Modo is (Activo, Inactivo);
```

```
   type Tipo_Perfil_Ejecución ( Estado_En_Modo :  Tipo_Estado_En_Modo := Activo)
   is record
      case Estado_En_Modo is
         when Activo =>
            Período         :  Duration;
            Fase            :  Duration;
            Prioridad       :  System.Priority;
            T_Cómputo       :  CPU_Time_Accounting.Accounting_Range;
            Plazo_Respuesta :  Duration;
         when Inactivo =>
            null;
      end case;
   end record;

   Instante_Inicio :  Calendar.Time := . . . ;

   type Tipo_Modo_Sistema is (Modo_Inicial, Modo_Estable, Modo_Error);

   Modo_Inicial_Sistema :  Tipo_Modo_Sistema := Modo_Inicial;

   type Tipo_Perfiles_Tarea is array (Tipo_Modo_Sistema) of Tipo_Perfil_Ejecución;

end Global;
```

Gestión del modo del sistema.

El modo de ejecución es un parámetro global del sistema. Por consiguiente hay que proporcionar operaciones a las tareas del sistema para interaccionar con él. Teniendo en cuenta el enfoque del diseño del algoritmo de cambio de modo, los servicios mínimos que se deben proporcionar son:

- *Cambiar el modo del sistema.* Este servicio permite a la tarea llamante el cambio de modo del sistema.
- *Consultar el modo actual.* Devuelve el modo actual de ejecución del sistema.
- *Consultar el instante efectivo de cambio de modo.* Devuelve el instante efectivo del último cambio de modo.
- *Bloqueo de una tarea inactiva.* Se bloquea a la tarea llamante hasta que se cambia de modo.

Esta funcionalidad la proporciona el paquete `Gestor_Modo_Sistema` cuya especificación se muestra a continuación:

```
with Calendar;
with Global;

package Gestor_Modo_Sistema is
```

```
      procedure Cambiar_Modo (Modo :  in out Global.Tipo_Modo_Sistema);

      function Consultar_Modo return Global.Tipo_Modo_Sistema;

      function Consultar_Instante_Efectivo_Cambio return Calendar.Time;

      procedure Dormir;

   end Gestor_Modo_Sistema;
```

Los servicios que proporciona este paquete son procedimientos o funciones. De esta forma se ocultan al usuario los detalles de realización de la funcionalidad requerida. Su realización se concreta mediante un objeto protegido, que encapsula y protege los datos relativos al modo del sistema, y una tarea, que gestiona a las tareas inactivas. Por tanto, las subrutinas anteriores ocultan llamadas a servicios del objetos protegido o a puntos de entrada de la tarea.

En el cuerpo del paquete `Gestor_Modo_Sistema` se incluyen los cuerpos de los subprogramas declarados en su especificación y las especificaciones del objeto protegido `Gestor_Modo` y la tarea `Tarea_Gestora_Inactivas`. Los cuerpos de estos elementos se incluyen en paquetes separados. En los siguientes apartados se justifican las decisiones que han motivado su definición. El código del cuerpo del paquete es:

```
package body Gestor_Modo_Sistema is

   task Tarea_Gestora_Inactivas is
      entry Dormir_Par;
      entry Dormir_Impar;
   end Tarea_Gestora_Inactivas;

   task body Tarea_Gestora_Inactivas is separate;


   protected Gestor_Modo is
      procedure Cambiar_Modo (Modo :  in out Global.Tipo_Modo_Sistema);
      function Consultar_Modo return Global.Tipo_Modo_Sistema;
      function Consultar_Instante_Efectivo_Cambio return Calendar.Time;
      function Consultar_Cambio_Par return Boolean;
      entry Esperar_Cambio;
      pragma priority (System.Priority'Last - 2);
   record
      Modo_Sistema                : Global.Tipo_Modo_Sistema
                                    := Global.Modo_Inicial_Sistema;
      Instante_Efectivo_Cambio    : Calendar.Time;
      Retardo_Efectivo            : Duration := ....;
      Separación_Entre_Cambios    : Duration := ....;
      Cambio_Modo                 : Boolean := False;
      Cambio_Par                  : Boolean := False;
   end Gestor_Modo;
```

```
protected body Gestor_Modo is separate;

procedure Cambiar_Modo (Modo:  in out Global.Tipo_Modo_Sistema) is
begin
   Gestor_Modo.Cambiar_Modo(Modo);
end Cambiar_Modo;

function Consultar_Modo return Global.Tipo_Modo_Sistema is
begin
   return Gestor_Modo.Consultar_Modo;
end Consultar_Modo;

function Consultar_Instante_Efectivo_Cambio return Calendar.Time is
begin
   return Gestor_Modo.Consultar_Instante_Efectivo_Cambio;
end Consultar_Instante_Efectivo_Cambio;

procedure Dormir is
begin
   case Gestor_Modo.Consultar_Cambio_Par is
      True  :  Tarea_Gestora_Inactivas.Dormir_Par;
      False :  Tarea_Gestora_Inactivas.Dormir_Impar;
   end case;
end Dormir;

end Gestor_Modo_Sistema;
```

Información relativa al modo del sistema. La estructura de datos que va a representar la información relativa al modo de ejecución del sistema será accedida concurrentemente por las tareas. Por tanto, es obvia la necesidad de disponer de mecanismos que garantizen la consistencia de los datos. El enfoque que se ha elegido es encapsular esta información en un objeto protegido. Se ha preferido esta opción respecto al uso de una tarea, porque es más eficiente y es inmediata la asignación del tiempo de cómputo que se consume a las tareas llamantes.

Las operaciones protegidas que se proporcionan coinciden con las que proporciona el paquete `Gestor_Modo_Sistema`. Además, se incluye la llamada `Esperar_Cambio`, que es un punto de entrada en el que se bloquea a la tarea gestora de inactivas hasta que se produce un cambio de modo.

La información que mantiene el objeto protegido y las operaciones asociadas se pueden dividir en dos apartados:

- *Información sobre el modo del sistema.* En este apartado se incluye la información necesaria para gestionar el modo del sistema y está compuesta por los siguientes elementos:
  - `Modo_Sistema`: Esta variable almacena el modo de ejecución del sistema en

cada momento.

- Instante_Efectivo_Cambio: Cuando se produce un cambio de modo, esta variable representa el instante efectivo de cambio de modo. Se calcula añadiendo a la hora del sistema el valor del retardo efectivo.
- Retardo_Efectivo: Este valor es una constante y representa el retardo máximo desde que se produce un cambio de modo, hasta que se puede garantizar que todas las tareas comenzarán a ejecutar en el modo nuevo a partir de su siguiente activación. Se considera el mayor retardo entre todos los modos.
- Separación_Entre_Cambios: los cambios de modo no se pueden producir tan frecuentemente como sea solicitado. Entre dos operaciones consecutivas, es necesario esperar a que todas las tareas involucradas hayan completado el cambio de modo. Esta constante representa este valor y es el máximo retardo entre todos los modos.

En cuanto a las operaciones asociadas, Consultar_Modo y Consultar_Instante_Efectivo_Cambio devuelve, respectivamente, el modo actual de ejecución del sistema y el instante efectivo de cambio. Este valor sólo será significativo si se acaba de producir un cambio de modo. El procedimiento Cambiar_Modo permite a las tareas cambiar el modo del sistema. Para que esta operación tenga efecto, se debe cumplir que el modo al que se quiere transitar es diferente del actual y que desde el anterior cambio de modo, ha transcurrido la separación mínima. Si esta condiciones son ciertas, se actualiza el valor que determina el modo del sistema, se calcula el instante efectivo de cambio y se modifica el valor de las variables Cambio_Modo y Cambio_Par, según se comenta a continuación.

- *Activación de la tarea gestora de inactivas.* Este objeto protegido mantiene información relativa a la gestión de tareas inactivas. La justificación es que algunas operaciones asociadas a esta funcionalidad deben sincronizarse con el instante del cambio de modo. En esta situación están las siguientes variables:

  - Cambio_Modo: Esta variable es la guarda del punto de entrada Esperar_Cambio. Cuando se produce un cambio de modo, se le asigna el valor cierto, para desbloquear a la tarea gestora de inactivas. Está asignará el valor falso a la variable inmediatamente después, para quedar nuevamente a la espera cuando desbloquee a las tareas inactivas.
  - Cambio_Par: Esta variable se emplea para seleccionar el punto de entrada donde deben bloquearse las tareas inactivas, según se justifica posteriormente. Cada vez que se produce un cambio de modo, se cambia el valor de esta variable.

Las operaciones asociadas son el punto de entrada Esperar_Cambio, con la funcionalidad comentada y Consultar_Cambio_Par, que devuelve el valor de la variable que corresponde.

```
separate (Gestor_Modo)
protected body Gestor_Modo is

   procedure Cambiar_Modo (Modo :  in out Global.Tipo_Modo_Sistema) is
   begin
      if Modo <> Modo_Sistema
        and Instante_Efectivo_Cambio + Separación_Entre_Cambios < Calendar.Clock
      then
         Instante_Efectivo_Cambio := Calendar.Clock + Retardo_Efectivo;
         Modo_Sistema := Modo;
         Cambio_Modo := True;
         Cambio_Par := not Cambio_Par;
      else
         Modo := Modo_Sistema;
      end if;
   end Cambiar_Modo;

   function Consultar_Modo return Tipo_Modo_Sistema is
   begin
      return Modo_Sistema;
   end Consultar_Modo;

   function Consultar_Instante_Efectivo_Cambio return Calendar.Time is
   begin
      return Instante_Efectivo_Cambio;
   end Consultar_Instante_Efectivo_Cambio;

   function Consultar_Cambio_Par return Boolean is
   begin
      return Cambio_Par;
   end Consultar_Cambio_Par;

   entry Esperar_Cambio when Cambio_Modo is
   begin
      Cambio_Modo := False;
   end Esperar_Cambio;

end Gestor_Modo;
```

Tarea gestora de inactivas. La función que proporciona esta tarea podría haber sido realizada con un registro protegido. El interés de utilizar una tarea radica en que la tarea se puede integrar en el análisis de planificabilidad como tal. Si se emplea un registro protegido la contabilidad del tiempo de cómputo se realiza asignando el tiempo de cómputo consumido a la tarea que efectuó la llamada. En este caso es más interesante la primera opción, pues el tiempo de cómputo que se consume para despertar a las tareas inactivas y que comprueben su perfil en el nuevo modo no se puede asignar a las tareas de tiempo real.

La tarea gestora de tareas inactivas realiza dos funciones:

- Proporciona un mecanismo para que las tareas inactivas en un modo permanezcan

bloqueadas y no consuman tiempo de cómputo durante su vigencia.

- Desbloquea a las tareas inactivas cuando se produce un cambio de modo. Entonces las tareas inactivas pueden adecuar su perfil de ejecución al nuevo modo.

Las tareas inactivas se bloquean mediante una llamada a un punto de entrada de esta tarea, que sólo se acepta cuando se produce un cambio de modo. La realización práctica de esta funcionalidad requiere el uso de dos puntos de entrada. El punto de entrada al que llaman las tareas se cambia cada vez que se produce un cambio de modo. Esta alternancia es necesaria porque cuando una tarea se desbloquea, comprueba si en el modo nuevo de ejecución está activa. Si no es así, volverá a llamar al punto de entrada para bloquearse. Entonces podría ocurrir que las tareas fueran despertadas varias veces en el mismo cambio de modo.

Esta realización se oculta a las tareas. Éstas siempre llaman al procedimiento `Dormir`, el cuál dependiendo del valor de la variable `Cambio_Par`, realizará la llamada a uno de los dos punto de entrada.

Habitualmente, la tarea gestora permanecerá bloqueada en el punto de entrada `Esperar_Cambio`. Cuando se produce un cambio de modo, es desbloqueada. A continuación, entra en un bucle en el que despierta a las tareas inactivas bloqueadas. Finalmente llamará al citado punto de entrada, con lo que volverá a bloquearse hasta que se produzca otro cambio de modo.

```
separate (Tarea_Gestora_Inactivas);

task body Tarea_Gestora_Inactivas is
   Num_Tareas :  Integer;
begin
   loop
      Gestor_Modo.Esperar_Cambio;

      case not Gestor_Modo.Consultar_Cambio_Par of
         True :  Num_Tareas := Dormir_Par'Count;
                 for i:= 1 to Num_Tareas loop
                    accept Dormir_Par;
                 end loop;
         False:  Num_Tareas := Dormir_Impar'Count;
                 for i:= 1 to Num_Tareas loop
                    accept Dormir_Impar;
                 end loop;
      end case;
   end loop;
end Tarea_Gestora_Inactivas;
```

Tareas de tiempo real.

El esquema de realización de las tareas coincide con el presentado anteriormente. Las diferencias están relacionadas con la definición de los perfiles de ejecución en cada modo y

con la detección y, en su caso, ejecución del protocolo de cambio de modo. Esta operación se debe realizar al final de cada activación.

Para facilitar la lectura de este apartado, se va dividir en dos partes:

- Realización de las tareas de tiempo real.
- Gestión del modo de las tareas.

Realización de las tareas de tiempo real. El paquete que representa una tarea periódica sufre ligeras modificaciones relacionadas con la representación del modo. Así, se declaran las variables `Info_Modo_Tarea`, que engloba toda la información relativa a la ejecución del protocolo de cambio de modo. También se declaran variables para almacenar los perfiles de ejecución de las tarea y el perfil específico del modo actual del sistema.

El tipo `Tipo_Info_Modo_Tarea` está declarado en el paquete `Gestor_Modo_Tarea`, al igual que los procedimientos necesarios para realizar el algoritmo de cambio de modo.

```
package Tarea_Periódica is
end Tarea_Periódica;


with System;
with Global;
with CPU_Time_Accounting;
with Gestor_Modo_Tarea;
package body Tarea_Periódica is

   Perfiles_Tarea :  Global.Tipo_Perfiles_Tarea :=
      ((Estado_En_Modo=> Activo, Período=> 0.1, Fase=> 1.0, Prioridad=> 3,
        T_Cómputo=> 0.01, Plazo_Respuesta=> 0.1)
      (Activo, 0.5, 0.3, 4, 0.1, 0.4)
      (Activo, 0.25, 0.2, 2, 0.05, 0.25));

   Perfil_Ejecución :  Global.Tipo_Perfil_Ejecución;

   Info_Modo_Tarea :  Gestor_Modo_Tarea.Tipo_Info_Modo_Tarea;

   Contador_T_Cómputo :  CPU_Time_Accounting.Overrun_Signal;

   task Tarea_Supervisora is
   end Tarea_Supervisora;

   task body Tarea_Supervisora is separate;

   task Tarea_Funcional is
      entry Fin_Activación(Ejecución :  out Tipo_Ejecución);
   end Tarea_Funcional;

   task body Tarea_Funcional is separate;

end Tarea_Periódica;
```

Las tareas de tiempo real deben comprobar al final de cada activación si se ha producido cambio de modo, y en este caso se debe ejecutar el protocolo. En el modo al que se transita la tarea puede ser inactiva, por lo que habrá que bloquear a la tarea supervisora y a la funcional. Por otro lado se quiere que la tarea funcional ejecute todo el código posible para minimizar la inversión de prioridades que se produce cuando la tarea supervisora se ejecuta. Por esta razón la tarea funcional detectará si ha habido cambio de modo y se bloqueará mediante los servicios proporcionados por `Gestor_Modo_Sistema`.

La primera conclusión es que la tarea funcional debe ejecutar el código en dos partes. Primero detecta el cambio de modo y luego ejecuta el procedimiento de cambio. Entre estas dos operaciones debe desbloquear a la tarea supervisora para que ejecute la rutina de cambio de modo. Si sólo lo ejecutara en una llamada, se bloquearía antes de despertar a la supervisora o ésta debería detectar el cambio de modo, que no es el comportamiento deseado.

La determinación de la siguiente activación de la tarea puede verse afectada por el protocolo de cambio de modo. Por esta razón el instante de la siguiente activación es un argumento en las llamadas para ejecutar el código del protocolo. Si no se ha producido un cambio de modo, este argumento no se modifica. En caso contrario, se devuelve el instante de la primera activación en el modo nuevo.

La identificación del perfil inicial de ejecución de la tarea se debe realizar de forma análoga a un cambio de modo. De esta forma se asegura un funcionamiento correcto en el caso que la tarea esté inactiva en el modo inicial de ejecución.

El comentario del código de la tarea funcional se realiza como es habitual, presentando cada una de los procedimientos básicos que le forman:

- `Operaciones_Iniciales`. Es necesario inicializar la variable `Info_Modo_Tarea` con los perfiles de ejecución de la tarea en cada uno de los modos. Para tal fin se llama al procedimiento `Asignar_Perfiles`. A continuación se ejecuta el protocolo de cambio de modo para obtener el perfil de ejecución correspondiente al modo inicial o bloquear a la tarea si es inactivo. Esta funcionalidad se realiza mediante una llamada a `Ejecutar_Cambio_Modo_Func`, el cuál también retorna el siguiente instante de activación.

- `Operaciones_Previas`. No son necesarias operaciones previas.

- `Operaciones_Posteriores`. En las operaciones posteriores es necesario realizar dos acciones relacionadas con el protocolo del cambio de modo. Al finalizar la activación y antes de despertar la tarea supervisora, se llama a `Detectar_Cambio_Modo`. En caso de que se haya producido uno, se actualiza la información en `Info_Modo_Tarea`. Posteriormente se le comunica a la tarea supervisora el fin de la activación. Al acabar esta operación se calcula el siguiente instante de activación y se llama a `Ejecutar_Cambio_Modo_Func`, que en caso de que llegue el instante de cambiar de modo realiza las operaciones relativas a la tarea funcional y que

devuelve el siguiente instante de activación y el perfil de ejecución en el nuevo modo.

Nótese que puede pasar bastante tiempo hasta que se retorne de esta función, pues la tarea puede permanecer inactiva.

```
with Calendar;

separate (Tarea_Periódica)
task body Tarea_Funcional is

   Siguiente_Activación :  Calendar.Time := Global.Instante_Inicio;

-- ****************************************************

   procedure Operaciones_Iniciales is

   begin
      Gestor_Modo_Tarea.Asignar_Perfiles
            (Info_Modo_Tarea, Perfiles_Tarea, Perfil_Ejecución);
      Gestor_Modo_Tarea.Ejecutar_Cambio_Modo_Func
         (Info_Modo_Tarea, Siguiente_Activación, Perfil_Ejecución);
      CPU_Time_Accounting.Attach_Overrun_Signal
            (Tarea_Funcional, Contador_T_Cómputo);
      delay until Siguiente_Activación;
   end Operaciones_Iniciales;
-- ****************************************************
   procedure Operaciones_Previas is

   begin
      null;
   end Operaciones_Previas;

-- ****************************************************
   procedure Actividad_Periódica is

   begin
      . . .
   end Actividad_Periódica;

-- ****************************************************
   procedure Operaciones_Posteriores is
   begin
      Gestor_Modo_Tarea.Detectar_Cambio_Modo(Info_Modo_Tarea);
      accept Fin_Activación(Ejecución) do
         Ejecución := Global.Correcta;
      end Fin_Activación;
      Siguiente_Activación := Siguiente_Activación + Perfil_Ejecución.Período;
      Gestor_Modo_Tarea.Ejecutar_Cambio_Modo_Func
         (Info_Modo_Tarea, Siguiente_Activación, Perfil_Ejecución);
      delay until Siguiente_Activación;
   end Operaciones_Posteriores;
```

```
-- ***************************************************
begin -- Tarea_Funcional
   Operaciones_Iniciales;
   loop
      Operaciones_Previas;
      Actividad_Periódica;
      Operaciones_Posteriores;
   end loop;

exception
   when others =>
      accept Fin_Activación(Ejecución) do
         Ejecución := Global.Fallo_Interno;
      end Fin_Activación;
end Tarea_Funcional;
```

La tarea supervisora también debe ejecutar la rutina de cambio de modo al inicio de la ejecución y al final de la activación. Esta operación sólo la realiza si no ha habido fallos de ejecución y se produce mediante una llamada a `Ejecutar_Cambio_Modo_Sup`. Si ha habido un cambio de modo, este procedimiento bloquea a la tarea, si el perfil para el modo nuevo es inactivo, y devuelve el siguiente instante de activación si es activo.

```
with Calendar;
with Ada.Dynamic_Priorities;

separate (Tarea_Periódica)
task body Tarea_Supervisora is

    Fallo_T_Funcional :  Boolean := False;
    Result_Ejecución :  Tipo_Ejecución := Correcta;
    Siguiente_Activación :  Calendar.Time := Global.Instante_Inicio;

    -- ***************************************************

    procedure Tratar_Fallo(Fallo_T_Funcional :  in Boolean;
                Result_Ejecución :  in Global.Tipo_Ejecución) is


    begin
        if Fallo_T_Funcional then
            abort Tarea_Funcional;
        end if;
    end Tratar_Fallo;
    -- ***************************************************

    procedure Operaciones_Iniciales is

    begin
        Ada.Dynamic_Priorities.Set_Priority
            (System.Priority'Last - 1, Tarea_Supervisora);
        Gestor_Modo_Tarea.Ejecutar_Cambio_Modo_Sup
```

```
            (Info_Modo_Tarea, Siguiente_Activación);
    end Operaciones_Iniciales;

-- ***************************************************

begin -- Tarea_Supervisora

    Operaciones_Iniciales;

    loop

        CPU_Time_Accounting.Set_Value(Tarea_Funcional, Perfil_Ejecución.T_Cómputo);

        select
            Contador_T_Cómputo.Wait;
            Fallo_T_Funcional := True;
            Tratar_Fallo(Fallo_T_Funcional, Result_Ejecución);
        then abort
            select
                delay until Siguiente_Activación + Plazo_Respuesta;
                Contador_T_Cómputo.Wait;
                Fallo_T_Funcional := True;
            then abort
                Tarea_Funcional.Fin_Activación(Result_Ejecución);
                if Result_Ejecución /= Global.Correcta then
                    Tratar_Fallo(Fallo_T_Funcional, Result_Ejecución);
                end if;
            end select;
        end select;

        exit when Fallo_T_Funcional or Result_Ejecución /= Global.Correcta ;

        Siguiente_Activación := Siguiente_Activación + Perfil_Ejecución.Período;

        Gestor_Modo_Tarea.Ejecutar_Cambio_Modo_Sup
            (Info_Modo_Tarea, Siguiente_Activación);

    end loop;

exception
    when others =>
        Fallo_T_Funcional := True;
        Tratar_Fallo(Fallo_T_Funcional, Result_Ejecución);
end Tarea_Supervisora;
```

Gestión del modo de las tareas. Todas las tareas de tiempo real deben ejecutar una serie de operaciones en un cambio de modo. Con objeto de aislar su código de los detalles de realización del protocolo y para centralizar el código, se ha desarrollado el paquete `Gestor_Modo_Tarea`. Éste exporta un tipo con la información necesaria y un conjunto de operaciones asociadas al protocolo de cambio de modo. Este conjunto se podría haber

diseñado como un objeto protegido. La opción elegida es más eficiente y válida, pues no va a haber problemas de consistencia de datos por accesos concurrentes.

El diseño de los elementos del paquete se ha realizado teniendo en cuenta la estructura de la tarea periódica. Se ha intentado que la tarea funcional ejecute la mayoría de las operaciones posibles, con objeto de evitar la inversión de prioridades que provoca la ejecución de la tarea supervisora. Por esta razón ha sido necesario dividir las actividades de la tarea funcional en dos procedimientos, según se ha comentado anteriormente.

La especificación de `Gestor_Modo_Tarea` incluye el tipo `Tipo_Info_Modo_Tarea` y los procedimientos para iniciar las variables del tipo, detectar el cambio de modo y para que las tareas funcionales y supervisora ejecuten el protocolo de cambio de modo. El paquete incluye en su parte privada la declaración del tipo protegido `Tipo_Bloquear_Tarea`. Este tipo exporta dos operaciones: un punto de entrada, para que una tarea se bloquee, y un procedimiento, para desbloquearla. Se emplea para bloquear a la tarea supervisora cuando el modo de ejecución es inactivo. La tarea funcional la desbloquea cuando se cambia a un modo activo.

Los campos más relevantes del tipo `Tipo_Info_Modo_Tarea` son:

- `Ejecución_Inicial`: Variable lógica que permite diferenciar la primera ejecución del resto. Permite forzar la ejecución del cambio de modo.
- `Cambio_Modo`: Variable lógica que indica la detección de un cambio de modo.
- `Tarea_Inactiva`: Variable lógica que indica que en el modo anterior la tarea ha permanecido inactiva. Permite decidir el método de cálculo del primer instante de activación en el modo nuevo.
- `Instante_Cambio`: Esta variable contiene el instante efectivo de cambio de modo.
- `Bloqueo_Tarea`: Variable del tipo protegido `Tipo_Bloquear_Tarea`.
- `Perfiles_Tarea`: Esta variable incluye los perfiles de ejecución de la tarea para cada modo del sistema. La primera operación con el tipo debe ser asignarla los perfiles.
- `Modo_Sistema` y `Modo_Tarea`: Estas variables representan el modo de ejecución del sistema y de la tarea, respectivamente. Su valor sólo diferira cuando se produzca un cambio de modo.

```
with Calendar;
with Global;
package Gestor_Modo_Tarea is

   type Tipo_Info_Modo_Tarea is limited private;

   procedure Asignar_Perfiles
              (Info_Modo_Tarea  :  in out Tipo_Info_Modo_Tarea;
               Perfiles_Tarea  :  in Global.Tipo_Perfiles_Tarea);
```

```
    procedure Detectar_Cambio_Modo
                (Info_Modo_Tarea :  in out Tipo_Info_Modo_Tarea);

    procedure Ejecutar_Cambio_Modo_Func
                (Info_Modo_Tarea :  in out Tipo_Info_Modo_Tarea;
                Siguiente_Activación :  in out Calendar.Time;
                Perfil_Ejecución:  in out Global.Tipo_Perfil_Ejecución);

    procedure Ejecutar_Cambio_Modo_Sup
                (Info_Modo_Tarea :  in out Tipo_Info_Modo_Tarea;
                Siguiente_Activación :  in out Calendar.Time);

private
    protected type Tipo_Bloquear_Tarea is
        procedure Desbloquear_Tarea;
        entry Bloquear_Tarea;
    private
        Desbloqueo_Tarea :  Boolean := False;
    end Tipo_Bloquear_Tarea;

    type Tipo_Info_Modo_Tarea is
    record
        Ejecución_Inicial  :  Boolean := True;
        Cambio_Modo,
        Tarea_Inactiva     :  Boolean := False;
        Instante_Cambio    :  Calendar.Time;
        Bloqueo_Tarea      :  Tipo_Bloquear_Tarea;
        Perfiles_Tarea     :  Global.Tipo_Perfiles_Tarea;
        Modo_Tarea,
        Modo_Sistema       :  Global.Tipo_Modo_Sistema := Global.Modo_Inicial_Sistema;
    end record;

end Gestor_Modo_Tarea;
```

Las operaciones sobre el tipo `Tipo_Info_Modo_Tarea` permiten a las tareas cambiar el modo del sistema. Se llaman en cada activación, aunque sólo se ejecutará el algoritmo cuando se haya producido un cambio de modo. A continuación se presentan:

- `Asignar_Perfiles`: El propósito de este procedimiento es asignar los perfiles de ejecución de la tarea en cada modo, al campo `Perfiles_Tarea`. Esta operación tiene que ser la primera que se realiza sobre el tipo y la ejecuta la tarea funcional.

- `Detectar_Cambio_Modo`: Las operaciones que debe realizar la tarea funcional para cambiar de modo se han dividido en dos partes. Este procedimiento constituye la primera. Su objetivo es detectar si se ha producido un cambio de modo, para asegurar que la tarea supervisora realiza las operaciones de cambio de modo adecuadamente.

  Este procedimiento consulta el modo del sistema y lo compara con el modo de ejecución de la tarea. Cuando son diferentes se ha producido un cambio de modo.

Sin embargo la tarea no ejecutará siempre el algoritmo de cambio de modo. Si el perfil de ejecución en el modo viejo y en el nuevo son iguales, la única operación que se debe realizar es actualizar el modo de la tarea. La segunda parte del algoritmo nunca se ejecutará.

Si los perfiles son diferentes, entonces se consulta el instante efectivo de cambio y se asigna a la variable `Cambio_Modo` el valor cierto. De esta forma se asegura que se ejecutará el algoritmo completo de cambio de modo.

Desde que se detecta el cambio de modo hasta el instante efectivo de cambio, se puede llamar varias veces a `Detectar_Cambio_Modo`. En este caso no es necesario realizar operación alguna, porque ya se ha detectado el cambio de modo.

- `Ejecutar_Cambio_Modo_Func`: Este procedimiento constituye la segunda parte de la ejecución del algoritmo de cambio de modo por parte de la tarea funcional. Estas operaciones sólo se ejecutan en la activación inicial de la tarea y cuando se detecta el cambio de modo y el instante de las siguiente activación es mayor que el efectivo de cambio de modo.

  Inicialmente se comprueba si la tarea es inactiva en el modo nuevo de ejecución. En tal caso se entra en un bucle en el que se permanece hasta que se transite a un modo en que la tarea es activa. A continuación la tarea funcional se bloquea mediante una llamada a `Gestor_Modo_Sistema.Dormir` y sólo se despertará cuando se produzca un cambio de modo. Entonces consulta el modo actual de ejecución del sistema y el instante efectivo de nuevo cambio. La tarea permanecerá en el bucle si el modo sigue siendo inactivo.

  Cuando el modo del sistema sea activo, se ejecutan las operaciones de cambio de perfil y de cálculo del siguiente instante de activación. La primera consiste en asignar al parámetro de llamada `Perfil_Ejecución` el perfil correspondiente al nuevo modo y cambiar la prioridad de la tarea. La segunda se realiza a partir del instante de la siguiente activación proporcionado por la tarea o el instante efectivo de cambio y la fase de la tarea en el modo. La elección del método depende de si en el modo anterior la tarea estuvo activa o no.

  En el caso de que la tarea haya estado inactiva en el modo anterior es necesario despertar a la tarea supervisora. Esto se lleva a cabo llamando al procedimiento `Desbloquear_Tarea` del objeto protegido `Bloqueo_Tarea`.

  Finalmente, se cambia el valor de una serie de variables lógicas para preparar el tipo para el siguiente cambio de modo. Esta operación la debe realizar la tarea funcional, ya que siempre será la última en completar el código del cambio de modo.

- `Ejecutar_Cambio_Modo_Sup`: Este procedimiento engloba las operaciones de cambio de modo que debe ejecutar la tarea supervisora. Las circunstancias en que se ejecutan coincide con el caso anterior.

  Las operaciones de cambio comienzan con la comprobación del estado de ejecución de la tarea en el modo nuevo. Si es inactivo, la tarea supervisora se bloquea

llamando al punto de entrada Bloquear_Tarea del objeto protegido Bloqueo_Tarea. En este estado permanece hasta que se cambia a un modo activo. Finalmente, continuación, calcula el siguiente instante de activación de la tarea.

```
with Ada.Dynamic_Priorities;
with Gestor_Modo_Sistema;
package body Gestor_Modo_Tarea is

   protected type Tipo_Bloquear_Tarea is
      procedure Desbloquear_Tarea is
      begin
         Desbloqueo_Tarea := True;
      end Desbloquear_Tarea;
      entry Bloquear_Tarea when Desbloqueo_Tarea is
      pragma priority (System.Priority'Last - 1);
      begin
         Desbloqueo_Tarea := False;
      end Bloquear_Tarea;
   end Tipo_Bloquear_Tarea;

-- ************************************************************

   procedure Asignar_Perfiles   (Info_Modo_Tarea :  in out Tipo_Info_Modo_Tarea;
                                 Perfiles_Tarea :  in Global.Tipo_Perfiles_Tarea) is
   begin
      Info_Modo_Tarea.Perfiles_Tarea := Perfiles_Tarea;
   end;

-- ************************************************************

   procedure Detectar_Cambio_Modo(Info_Modo_Tarea :  in out Tipo_Info_Modo_Tarea) is
   begin
      if not Info_Modo_Tarea.Cambio_Modo
      then
         Info_Modo_Tarea.Modo_Sistema := Gestor_Modo_Sistema.Consultar_Modo;
         if Info_Modo_Tarea.Modo_Sistema /= Info_Modo_Tarea.Modo_Tarea
         then
            if Info_Modo_Tarea.Perfiles_Tarea(Info_Modo_Tarea.Modo_Tarea)
               = Info_Modo_Tarea.Perfiles_Tarea(Info_Modo_Tarea.Modo_Sistema)
            then
               Info_Modo_Tarea.Modo_Tarea := Info_Modo_Tarea.Modo_Sistema;
            else
               Info_Modo_Tarea.Cambio_Modo := True;
               Info_Modo_Tarea.Instante_Cambio
                  := Gestor_Modo_Sistema.Consultar_Instante_Efectivo_Cambio;
            end if;
         end if;
      end if;
   end Detectar_Cambio_Modo;

-- ************************************************************
```

```
procedure Ejecutar_Cambio_Modo_Func
            (Info_Modo_Tarea : in out Tipo_Info_Modo_Tarea
            Siguiente_Activación : in out Calendar.Time,
            Perfil_Ejecución : in out Global.Tipo_Perfil_Ejecución) is
begin
   if Info_Modo_Tarea.Ejecución_Inicial
      or else (Info_Modo_Tarea.Cambio_Modo
      and then Info_Modo_Tarea.Instante_Cambio < Siguiente_Activación)
   then
      while
        (Info_Modo_Tarea.Perfil_Ejecución(Info_Modo_Tarea.Modo_Sistema).Estado_En_Modo
        = Global.Inactivo
      loop
         Info_Modo_Tarea.Tarea_Inactiva := True;
         Gestor_Modo_Sistema.Dormir;
         Info_Modo_Tarea.Modo_Sistema := Gestor_Modo_Sistema.Consultar_Modo;
         Info_Modo_Tarea.Instante_Cambio :=
            Gestor_Modo_Sistema.Consultar_Instante_Efectivo_Cambio;
      end loop;
      Info_Modo_Tarea.Modo_Tarea := Info_Modo_Tarea.Modo_Sistema;
      Perfil_Ejecución := Info_Modo_Tarea.Perfiles_Tarea(Info_Modo_Tarea.Modo_Tarea);
      if Info_Modo_Tarea.Tarea_Inactiva
      then
         Siguiente_Activación :=
            Info_Modo_Tarea.Instante_Cambio
            + Info_Modo_Tarea.Perfiles_Tarea(Info_Modo_Tarea.Modo_Tarea).Fase;
         Info_Modo_Tarea.Bloqueo_Tarea.Desbloquear_Tarea;
      else
         Siguiente_Activación :=
            Siguiente_Activación
            + Info_Modo_Tarea.Perfiles_Tarea(Info_Modo_Tarea.Modo_Tarea).Fase;
      end if;

      Info_Modo_Tarea.Cambio_Modo := False;
      Info_Modo_Tarea.Tarea_Inactiva := False;
      Info_Modo_Tarea.Ejecución_Inicial := False;

      Ada.Dynamic_Priorities.Set_Priority
         (Perfil_Ejecución(Info_Modo_Tarea.Modo_Tarea).Prioridad);

   end if;
end Ejecutar_Cambio_Modo_Func;

-- ***********************************************************

procedure Ejecutar_Cambio_Modo_Sup
            (Info_Modo_Tarea : in out Tipo_Info_Modo_Tarea;
            Siguiente_Activación: in out Calendar.Time) is

begin
   if Info_Modo_Tarea.Ejecución_Inicial
      or else ( Info_Modo_Tarea.Cambio_Modo
      and then Info_Modo_Tarea.Instante_Cambio < Siguiente_Activación )
```

```
      then
         if
            (Info_Modo_Tarea.Perfil_Ejecución(Info_Modo_Tarea.Modo_Sistema).Estado_En_Modo)
            = Global.Inactivo
         then
            Info_Modo_Tarea.Bloqueo_Tarea.Bloquear_Tarea;
         end if;

         if Info_Modo_Tarea.Tarea_Inactiva
         then
            Siguiente_Activación :=
               Info_Modo_Tarea.Instante_Cambio
               + Info_Modo_Tarea.Perfiles_Tarea(Info_Modo_Tarea.Modo_Tarea).Fase;
         else
            Siguiente_Activación :=
               Siguiente_Activación
               + Info_Modo_Tarea.Perfiles_Tarea(Info_Modo_Tarea.Modo_Tarea).Fase;
         end if;

   end Ejecutar_Cambio_Modo_Sup;

end Gestor_Modo_Tarea;
```

Análisis de planificabilidad. Las consideraciones respecto al análisis de planificabilidad son las mismas que en el esquema anterior. El algoritmo de cambio de modo sólo implica tiempo de cómputo adicional en cada activación. El cambio de perfil de ejecución, que implica cambio de prioridad, se realiza de acuerdo al protocolo de cambio de modo. Por consiguiente, se pueden garantizar la planificabilidad, si esta se ha comprobado adecuadamente y si los instantes de inicio en cada modo son correctos.

6.4 Tratamiento de fallos.

6.4.1 Planteamiento.

Un requisito importante en los sistemas de tiempo real crítico de la siguiente generación es la tolerancia a fallos [Stankovic88a], entendida como la capacidad de un sistema para proporcionar la funcionalidad requerida, aún en presencia de fallos software o hardware. La única forma de conseguir este objetivo es mediante la replicación de los componentes hardware y software del sistema y el empleo de técnicas que aseguren la consistencia de las operaciones y la detección y tratamiento de los fallos del sistema. El estudio de este problema en toda su extensión y complejidad está fuera del ámbito de este trabajo.

Sin embargo, es necesario proporcionar algún método de tratamiento de fallos que garantize un comportamiento seguro del sistema. El mecanismo de tratamiento de fallos empleado en los esquemas de tareas propuestos anteriormente es muy burdo. Como se recordará, consiste en detener sin más a las tareas cuando se detecta un fallo de ejecu-

ción. Si es la tarea supervisora la que ha detectado el fallo, aborta inmediatamente a la funcional y termina. Este enfoque no es adecuado. Si el sistema de tiempo real controla procesos físicos hay tener en cuenta que no se puede terminar sin más la ejecución, ya que no se sabe con certeza el estado en que queda el proceso. En concreto, podría ser un estado estado inestable que derivara en situaciones peligrosas. Por consiguiente, el comportamiento mínimo requerido a una tarea cuando falla es asegurar que los recursos dependientes quedan en un estado seguro.

Por otro lado, el detener una tarea o el sistema al primer fallo de ejecución no es deseable. Un enfoque alternativo consiste en degradar controladamente su funcionalidad en presencia de fallos. Si la aplicación está compuesta por un conjunto de subsistemas poco acoplados, es muy probable que si uno falla, el resto pueda continuar ejecutando correctamente. Así, aunque el sistema no se comporte optimamente, puede proporcionar funciones útilies hasta que se puedan resolver los fallos.

Así pues, los objetivos de esta sección se resumen en dos cuestiones:

- Plantear un mecanismo de tratamiento de fallos que degrade consistente y progresivamente la funcionalidad del sistema. En último caso, deberá dejar el sistema en un estado seguro.
- Integrar consistentemente estos mecanismos en los esquema de las tareas.

6.4.2 Grupo de recuperación.

Los objetivos propuestos respecto al tratamiento de fallos requieren el cumplimiento de las siguientes propiedades:

- *Encapsulamiento de fallos.* Los fallos de una tarea no se debe propagar incontroladamente al resto del sistema. Por tanto, se debe asegurar que son detectados y tratados en su ámbito de ejecución. Este objetivo se consideró anteriormente y se ha tenido en cuenta en los esquemas de tareas desarrollados. Las tareas se han diseñado de forma que encapsulen sus fallos de ejecución, para impedir que se propaguen a otras tareas.

  La transparencia total del fallo de una tarea no es una propiedad deseable, ya que en general las tareas están relacionadas entre sí. El fallo de una de ellas probablemente impida a otras el cumplimiento satisfactorio de su funcionalidad. Por consiguiente, las tareas deben conocer si alguna tarea de la que dependen ha fallado.

- *Tratamiento de fallos.* La detención inmediata de una tarea cuando falla implica la desaparación de las funciones que realiza. Este enfoque de todo o nada es muy extremista en muchos casos. Una alternativa más interesante, que aporta un nivel adicional de degradación de la tarea, consiste en ejecutar un código alternativo que proporcione cierta funcionalidad reducida de la tarea, cuando falla.

  Este requisito se puede realizar asociando a cada tarea una que se denominará de recuperación. Si la primera falla, la segunda se activa y proporciona al sistema una

funcionalidad reducida respecto a la original, pero suficiente para el funcionamiento correcto del sistema. En caso de fallo de la tarea de recuperción, se ejecutaría una rutina que llevara a estado seguro los recursos dependientes de ella.

La combinación de las dos características anteriores conducen al concepto de ***grupo de recuperación*** [Alonso&92c][Alonso&92a][CASA&91a], que está compuesto por:

- Un conjunto de tareas relacionadas funcionalmente, de forma que si una falla, el resto no puede ejecutarse corréctamente. Estas tareas comparten las acciones de tratamiento de fallos de ejecución.
- Una tarea de recuperación, que incluye las acciones de tratamiento de fallos de las tareas del grupo y proporciona la funcionalidad básica del conjunto. Esta tarea permanecerá inicialmente inactiva. Cuando alguna tarea del grupo falla, se detiene la ejecución de todas ellas y se arranca la tarea de recuperación.

Las operaciones necesarias para detener a las tareas del grupo y activar a la tarea de recuperación se deben llevar a cabo de forma que no se interfiera con el cumplimiento de los plazos de respuesta del resto de las tareas. Este requisito es factible de realizar, ya que la ocupación del procesador de la tarea de recuperación será menor que la del conjunto de las tareas del grupo. Por tanto, para activar la tarea de recuperación sólo habrá que esperar el tiempo suficiente para disponer del tiempo de cómputo liberado por las tareas del grupo.

6.4.3 Consideraciones de diseño.

Gestión del estado del grupo. La información de estado del grupo indica si se ha producido o no un fallo de ejecución en alguna de sus tareas. Esta información debe ser accesible a todas ellas, para que notifiquen la ocurrencia de un fallo de ejecución y para que puedan detectar el cambio del estado y actuar en consecuencia.

Esta información condiciona la activación de la tarea de recuperación, que sólo se ejecutará cuando se haya producido un fallo. Por tanto, asociado al estado del fallo debe encontrarse el mecanismo que bloquee y desbloquee a esta tarea.

El estado del grupo es una información que puede ser accedida en paralelo por las tareas del grupo. Por tanto, será necesario encapsular las operaciones en regiones críticas para garantizar su consistencia.

Detección de fallo en el grupo. El comportamiento de las tareas en presencia de fallos del grupo depende de la inmediatez de la ejecución del tratamiento que se requiere. Si los requisitos de seguridad del sistema implican que se debe tratar inmediatamente, será necesario adoptar un modelo asíncrono, según el cuál la tarea que detecta el fallo de ejecución aborta la ejecución de todas las tareas del grupo y activa la tarea de recuperación. Este enfoque es también necesario si el fallo es tal que la ejecución del resto de las tareas sólo puede producir daños al sistema. La realización práctica es muy sencilla. Sólo hay

que ejecutar una secuencia de operaciones de abortar tareas y activar la tarea de recuperación. El retardo en el tratamiento del fallo es muy pequeño. Sin embargo, abortarlas indiscriminadamente podría dejar inconsistentes recursos compartidos.

La alternativa síncrona consiste en dejar que las tareas detecten el fallo del grupo y se detengan por sí sólas. La detección de los fallos se debe realizar cuando los datos están consistentes, lo que ocurre con certeza al final o al principio de una activación. Como es deseable que se detenga la ejecución de la tarea lo antes posible, se consultará el estado del grupo en estas dos circunstancias. El retardo en el tratamiento del fallo es mayor en este caso, pues habrá que esperar a que un número suficiente de tareas hayan terminado su ejecución antes de activar la tarea de recuperación.

En términos absolutos, ninguna de estas alternativas es claramente superior. Sus ventajas e inconvenientes son complementarios. La selección entre ellas dependerá de los requisitos funcionales de las aplicaciones. Para el desarrollo del esquema se ha decidido realizar la segunda, porque se adapta algo mejor a los criterios de diseño que se están siguiendo.

Activación de la tarea de recuperación. La tarea de recuperación sólo se podrá ejecutar cuando haya capacidad suficiente de procesador para que no se interfiera con el cumplimiento de los plazos de respuesta de las restantes tareas. Cuando las tareas del grupo terminan, liberan la capacidad de tiempo de cómputo asociada. Así pues, será suficiente esperar hasta que un número determinado de tareas hayan finalizado. Es importante recordar que cuando una tarea ha completado la ejecución, su tiempo de cómputo no está disponible hasta el siguiente instante de activación.

Fallos de la tarea de recuperación. La tarea de recuperación también puede fallar. Por tanto, es necesario disponer de algún mecanismo de tratamiento de estos fallos. El objetivo principal de esta operación es dejar a los recursos dependientes en un estado seguro y consistente. Por consiguiente, cuando se detecta un fallo en la tarea de recuperación se ejecuta una rutina que realize esta función y se termina su ejecución.

Hay que asegurar que la ejecución de estas operaciones no interfiere con el cumplimiento de los plazos de respuesta de otras tareas. La tarea supervisora es la encargada de realizar esta actividad. El tiempo de cómputo de estas operaciones es tiempo de bloqueo por inversión de prioridades. Si este incremento en el tiempo de bloqueo no reduce significativamente la capacidad de procesador disponible, este enfoque es el más adecuado ya que asegura la ejecución inmediata de la rutina de tratamiento del fallo. Si no fuera así, se podría ejecutar con la prioridad correspondiente de la tarea de recuperación.

6.4.4 Esquema de realización.

A continuación se presenta el esquema de realización de los componentes necesarios para dotar a las tareas de las propiedades anteriores de tratamiento de fallos.

Declaración de los objetos globales.

No es necesario declarar objetos globales adicionales. Por consiguiente, el paquete `Global` definido en la sección anterior es válido.

Gestión del estado de ejecución del grupo.

El estado de la ejecución del grupo de recuperación es una información global a éste y que es accedida concurrentemente. Para garantizar su consistencia, se va a emplear el tipo protegido `Tipo_Info_Grupo_Recup`, que encapsulará el estado del grupo. Este tipo se va declarar en el paquete `Gestor_Grupo_Recup`. Esta decisión permite que la declaración sea visible a todas las tareas de recuperación, sin que las tareas del grupo tengan que conocer su existencia, como se mostrará posteriormente.

El único dato del tipo es la variable lógica `Fallo_Notificado`. Como es obvio, esta variable permanecerá con el valor falso hasta que alguna tarea del grupo notifique un fallo de ejecución. Se proporcionan tres operaciones relacionadas con el estado del grupo.

- `Hay_Fallo`: permite a las tareas del grupo conocer si se ha notificado un fallo de ejecución.
- `Notificar_Fallo`: este procediento lo ejecuta una tarea cuando ha detectado un fallo, para cambiar el estado del grupo. Como consecuencia, se cambia el valor de la variable `Fallo_Notificado`.
- `Esperar_Fallo`: es un punto de entrada que se emplea para bloquear a la tarea de recuperación hasta que se produce un fallo en el grupo. La variable `Fallo_Notificado` es la guarda del punto de entrada. Por tanto, cuando alguna tarea notifica un fallo de ejecución, la tarea de recuperación se desbloquea. El parámetro `Instante_Fallo` indica a la tarea de recuperación el instante en que se produjo el fallo.

```
with Calendar;
with System;

   package Gestor_Grupo_Recup is
      protected type Tipo_Info_Grupo_Recup is

         function Hay_Fallo return Boolean;
         procedure Notificar_Fallo;
         entry Esperar_Fallo (Instante_Fallo :  out Calendar.Time);
         pragma priority (System.Priority'Last - 1);

      private
         Fallo_Notificado :  Boolean := False;
      end Tipo_Info_Grupo_Recup;
   end Gestor_Grupo_Recup;

   package body Gestor_Grupo_Recup is
```

```
protected body type Tipo_Info_Grupo_Recup is

function Hay_Fallo return Boolean is
begin
   return Fallo_Notificado;
end Hay_Fallo;

procedure Notificar_Fallo is
begin
   Fallo_Notificado := True;
end Notificar_Fallo;

entry Esperar_Fallo (Instante_Fallo : out Calendar.Time) when Fallo_Notificado is
begin
   Instante_Fallo := Calendar.Clock;
end Esperar_Fallo;

end Tipo_Info_Grupo_Recup;
end Gestor_Grupo_Recup;
```

Tarea de recuperación.

El esquema de realización y la ejecución de una tarea de recuperación sigue las pautas de una tarea periódica. Por consiguiente, el contenido del paquete en el que se declara es prácticamente el mismo que el de aquella. Las diferencias estriban en que la tarea de recuperación incluye el código necesario para gestionar el estado del grupo de recuperación. Para tal fin:

- Declara la variable `Info_Grupo_Recup`, que es un objeto del tipo protegido definido anteriormente.
- Proporciona a las tareas de su grupo los procedimientos necesarios para consultar y notificar el estado del grupo. Para que sean visible exteriormente, estas operaciones se declaran en la especificación del paquete. Como es habitual, ocultan al exterior las llamadas reales a las operaciones homónimas del objeto protegido `Info_Grupo_Recup`.

Para programar el instante de la primera activación de la tarea de recuperación, se declara la variable `Retardo_Inicial`, cuyo valor es el retardo necesario para que esté disponible el tiempo de cómputo liberado por las tareas del grupo.

El bloqueo inicial de la tarea de recuperación implica el bloqueo de la tarea funcional y de la supervisora. Para realizar esta operación, se ha añadido a la tarea funcional el punto de entrada `Activar_Supervisora`. El parámetro `Instante_Inicio`, indica a la tarea supervisora el instante de la primera activación.

```
package Tarea_Recuperación is
   function Hay_Fallo return boolean;
```

```
   procedure Notificar_Fallo;
end Tarea_Recuperación;


with System;
with Global;
with CPU_Time_Accounting;
with Gestor_Modo_Tarea;
package body Tarea_Recuperación is

   Perfiles_Tarea :  Global.Tipo_Perfiles_Tarea :=
      ((Estado_En_Modo=> Activo, Período=> 0.1, Fase=> 0, Prioridad=> 3,
        T_Cómputo=> 0.01, Plazo_Respuesta=> 0.1)
      (Activo, 0.5, 0, 4, 0.1, 0.4)
      (Activo, 0.25, 0, 2, 0.05, 0.25));

   Perfil_Ejecución :  Global.Tipo_Perfil_Ejecución;

   Info_Modo_Tarea :  Gestor_Modo_Tarea.Tipo_Info_Modo_Tarea;

   Contador_T_Cómputo :  CPU_Time_Accounting.Overrun_Signal;

   Info_Grupo_Recup:  Gestor_Grupo_Recup.Tipo_Info_Grupo_Recup;

   Retardo_Inicial :  Duration := .   .   .   .

   function Hay_Fallo return boolean is
      begin
         return Info_Grupo_Recup.Hay_Fallo;
      end Hay_Fallo;

   procedure Notificar_Fallo is
      begin
         Info_Grupo_Recup.Notificar_Fallo;
      end Notificar_Fallo

   task Tarea_Supervisora is
   end Tarea_Supervisora;

   task body Tarea_Supervisora is separate;

   task Tarea_Funcional is
      entry Fin_Activación(Ejecución :  out Tipo_Ejecución);
      entry Activar_Supervisora (Instante_Inicio :  out Calendar.Time);
   end Tarea_Funcional;

   task body Tarea_Funcional is separate;

end Tarea_Recuperación;
```

El esquema de la tarea funcional es exáctamente igual al presentado en el apartado anterior, excepto en las operaciones iniciales. Esto es lógico porque el tratamiento de

fallos que se aplica ahora es idéntico al que se aplicaba en aquella tarea. Cuando se detecta un fallo se comunica a la tarea supervisora y se termina. Por la razón expuesta, sólo se mostrará el código de las operaciones iniciales de la tarea.

Las operaciones iniciales son iguales hasta que llega el momento de programar el siguiente instante de activación. Entonces la tarea funcional se bloquea a la espera de que una tarea del grupo falle. Antes de realizar esta operación, se cambia la prioridad de la tarea al valor de la supervisora para asegurar que al activarse se adoptará el perfil de ejecución inmediatamente. Cuando se produce un fallo, se adapta el perfil de ejecución de la tarea al modo de ejecución del sistema y se detiene la ejecución hasta el instante de la primera activación. Este valor se calcula a partir del instante del fallo y del retardo inical de activación.

No se contempla la posibilidad de que el perfil de la tarea de recuperación correspondiente al modo del sistema sea inactiva. Sólo habrá un fallo en las tareas del grupo si hay alguna activa, y en este caso es necesario que se ejecute alguna las operaciones de tratamiento del fallo.

Un aspecto a considerar es a quién se contabiliza el tiempo de cómputo que la tarea funcional consume cuando ejecuta las operaciones necesarias para programar el siguiente instante de activación. Como estas operaciones no consumen mucho tiempo y se ejecutan con la prioridad de la tarea supervisora, se puede contabilizar como tiempo de cómputo provocado por la tarea que notificó el error. Esta es la suposición que se hace en el esquema de la tarea. Si este tiempo fuera excesivo, se debería asegurar que el código en cuestión se ejecuta con la prioridad que le corresponde, de acuerdo al perfil. En este caso, se podría cambiar la prioridad de la tarea funcional dentro del punto de entrada `Esperar_Llamada`.

Una vez que la tarea comienza a ejecutar la primera activación, despierta a la tarea supervisora. A partir de este momento, se comporta igual que una tarea periódica.

```
with Calendar;
with Ada.Dynamic_Priorities;

separate (Tarea_Recuperación);
task Tarea_Funcional is

   Instante_Fallo,
   Siguiente_Activación :  Calendar.Time;

-- ***************************************************

   procedure Operaciones_Iniciales is

   begin
      Gestor_Modo_Tarea.Asignar_Perfiles
            (Info_Modo_Tarea, Perfiles_Tarea, Perfil_Ejecución);
      CPU_Time_Accounting.Attach_Overrun_Signal
            (Tarea_Funcional, Contador_T_Cómputo);

      Ada.Dynamic_Priorities.Set_Priority
         (System.Priority'Last - 1, Tarea_Supervisora);
```

```
        Info_Grupo_Recup.Esperar_Fallo(Instante_Fallo);

        Gestor_Modo_Tarea.Detectar_Cambio_Modo(Info_Modo_Tarea);
        Gestor_Modo_Tarea.Ejecutar_Cambio_Modo_Func
           (Info_Modo_Tarea, Siguiente_Activación, Perfil_Ejecución);
        Siguiente_Activación := Instante_Fallo + Retardo_Inicial;
        delay until Siguiente_Activación;

        accept Activación_Supervisora(Instante_Inicio) do
           Instante_Inicio := Siguiente_Activación;
        end Activación_Supervisora;

     end Operaciones_Iniciales;

  -- ****************************************************

        . . . . . . .

  end Tarea_Funcional;
```

El esquema de la tarea supervisora sólo se diferencia del presentado en la sección anterior en las operaciones iniciales y en el procedimiento `Tratar_Fallo`. En las operaciones iniciales hay que añadir una llamada al punto de entrada `Activar_Supervisora`. Cuando la tarea funcional inicia la ejecución de la primera activación, despierta a la tarea supervisora. Esta ejecuta las operaciones necesarias para controlar la ejecución de la primera. Nótese que el comportamiento de la tarea supervisora en esta activación es diferente respecto al resto de los esquemas. La razón es que en esta primera activación, se ejecutan las operaciones de control al principio y al final de la activación. Esto supone aumentar el tiempo de cómputo de la tarea de recuperación por este único caso y aumentar el tiempo de inversión de prioridades en una activación. La primera cuestión no parece problemática si se tiene en cuenta que en general el tiempo de cómputo que consume la tarea de recuperación es menor que el del resto de las tareas del grupo, mientras que el resto de las tareas del sistema no aumentan su carga. En relación al segundo, aunque aumenta el tiempo de inversión de prioridades por la ejecución de la tarea supervisora, lo hace en dos partes, cuya duración es la misma que el tiempo de inversión de prioridades de los casos anteriores. Por consiguiente, no aumentará el bloqueo máximo por este motivo que sufran tareas más prioritarias.

En cuanto al procedimiento `Tratar_Fallo`, ahora hay que añadir las operaciones que llevan a los recursos controlados por la tarea de recuperación a un estado seguro. El tiempo de cómputo que se consume en esta operación es tiempo de inversión de prioridades, que habrá que tener en cuenta al realizar el análisis de planificabilidad. Si se quisiera disminuir bastaría con cambiar la prioridad al valor de la tarea en el modo y programar su ejecución en la siguiente activación.

```
with Calendar;
with Ada.Dynamic_Priorities;
```

```
separate (Tarea_Recuperación);
task body Tarea_Supervisora is

    Fallo_T_Funcional : Boolean := False;
    Result_Ejecución : Tipo_Ejecución := Correcta;
    Siguiente_Activación : Calendar.Time := Global.Instante_Inicio;

    -- ***************************************************
    procedure Tratar_Fallo(Fallo_T_Funcional : in Boolean;
              Result_Ejecución : in Global.Tipo_Ejecución) is

    begin
        if Fallo_T_Funcional then
            abort Tarea_Funcional;
        end if;


    -- Operaciones de parada segura de la tarea
        . . . . . . . . . . . . . .

    end Tratar_Fallo;
    -- ***************************************************
    procedure Operaciones_Iniciales is

    begin
        Ada.Dynamic_Priorities.Set_Priority
            (System.Priority'Last - 1, Tarea_Supervisora);

        Tarea_Funcional.Activación_Supervisora(Siguiente_Activación);

    end Operaciones_Iniciales;

-- ***************************************************

    . . . . . . . . . . .

end Tarea_Supervisora;
```

Tarea de tiempo real.

El paquete en el que se declara la tarea periódica es el mismo que en el caso anterior. Esto se debe a que las referencias a la tarea de recuperación se realizan en los cuerpos de la tarea funcional y supervisora.

Las tareas de tiempo real deben comprobar al principio y al final de cada activación el estado del grupo de recuperación. Para minimizar el tiempo de inversión de prioridades producida por la tarea supervisora, la tarea funcional será la que realize las operaciones de comprobación del estado. En caso de que se haya producido un fallo de ejecución, aceptará la cita con la tarea supervisora indicándo que la ejecución ha fallado.

La detección del fallo se debe realizar antes de las operaciones previas o posteriores

de una activación. Si se ha producido un fallo de ejecución, se debe terminar la ejecución y no es necesario realizar operaciones adicionales. En el código del bucle principal se han incluido dos sentencias para terminar la ejecución de éste si se ha producido un fallo en el grupo. Los procedimientos característicos de la tarea se describen a continuación:

- `Operaciones_Iniciales`. Son las mismas que en el caso anterior.
- `Operaciones_Previas`. Antes de iniciar una activación se comprueba si ha habido un fallo en el grupo. Si es así, se desbloquea a la tarea supervisora y se termina la ejecución de la tarea funcional.
- `Operaciones_Posteriores`. En primer lugar, se comprueba si ha habido un fallo en el grupo. Si es asím se actúa como en el caso anterior. Si no lo habido se realizan las operaciones del algoritmo de cambio de modo y de preparación de la siguiente activación.

```
with Calendar;
with Tarea_Recuperación;

separate (Tarea_Periódica);
task Tarea_Funcional is

   Siguiente_Activación :  Calendar.Time := Global.Instante_Inicio;
   Fallo_Grupo :  Boolean:= False;

-- **************************************************

   procedure Operaciones_Iniciales is
   begin
      . . . . . . . . .
   end Operaciones_Iniciales;

-- **************************************************

   procedure Operaciones_Previas is

   begin
      if Tarea_Recuperación.Hay_Fallo
      then
         Fallo_Grupo := True;
         accept Fin_Activación(Ejecución);
            Ejecución := Global.Fallo_Externo;
         end Fin_Activación;
      end if;
   end Operaciones_Previas;

-- **************************************************
   procedure Actividad_Periódica is

   begin
      . . .
```

```
   end Actividad_Periódica;

-- ***************************************************
   procedure Operaciones_Posteriores is
   begin
      if Tarea_Recuperación.Hay_Fallo
      then
         Fallo_Grupo := True;
         accept Fin_Activación(Ejecución);
            Ejecución := Global.Fallo_Externo;
         end Fin_Activación;
      else
         Gestor_Modo_Tarea.Detectar_Cambio_Modo(Info_Modo_Tarea);
         accept Fin_Activación(Ejecución);
            Ejecución := Global.Correcta;
         end Fin_Activación;
         Siguiente_Activación := Siguiente_Activación + Perfil_Ejecución.Período;
         Gestor_Modo_Tarea.Ejecutar_Cambio_Modo_Func
            (Info_Modo_Tarea, Siguiente_Activación, Perfil_Ejecución);
         delay until Siguiente_Activación;
      end if;
   end Operaciones_Posteriores;

-- ***************************************************
begin -- Tarea_Funcional
   Operaciones_Iniciales;
   loop
      Operaciones_Previas;
      exit when Fallo_Grupo;
      Actividad_Periódica;
      Operaciones_Posteriores;
      exit when Fallo_Grupo;
   end loop;

exception
   when others
      accept Fin_Activación(Ejecución);
         Ejecución := Global.Fallo_Interno;
      end Fin_Activación;
end Tarea_Funcional;
```

El código de la tarea supervisora es prácticamente el mismo que el caso anterior. La modificación consiste en que cuando se produce un fallo de la tarea funcional, se notifica un fallo a la tarea de recuperación. El fallo de la tarea funcional puede ser detectado por ella misma o por la tarea supervisora. En el primer caso, la variable `Result_Ejecución` tomará el valor `Fallo_Interno`. En el segundo caso la variable `Fallo_Tarea_Funcional` tomará el valor cierto. En cualquiera de estas circunstancias, se notifica el fallo a la tarea de recuperación y se termina.

```
with Calendar;
```

```
with Ada.Dynamic_Priorities;
with Tarea_Recuperación;

separate (Tarea_Periódica);
task body Tarea_Supervisora is

   Fallo_T_Funcional :  Boolean := False;
   Result_Ejecución :  Tipo_Ejecución := Correcta;
   Siguiente_Activación :  Calendar.Time := Global.Instante_Inicio;

   -- ****************************************************

   procedure Tratar_Fallo(Fallo_T_Funcional :  in Boolean;
         Result_Ejecución :  in Global.Tipo_Ejecución) is

   begin
      if Fallo_T_Funcional then
         abort Tarea_Funcional;
      end if;
      if Result_Ejecución = Global.Fallo_Interno or
         Fallo_T_Funcional then
         Tarea_Recuperación.Notificar_Fallo;
   end Tratar_Fallo;

   -- ****************************************************

   procedure Operaciones_Iniciales is

   begin
      Ada.Dynamic_Priorities.Set_Priority
         (System.Priority'Last - 1, Tarea_Supervisora);
      Gestor_Modo_Tarea.Ejecutar_Cambio_Modo_Sup
         (Info_Modo_Tarea, Siguiente_Activación);
   end Operaciones_Iniciales;

-- ****************************************************

begin -- Tarea_Supervisora

   Operaciones_Iniciales;

   loop

      CPU_Time_Accounting.Set_Value(Tarea_Funcional, Perfil_Ejecución.T_Cómputo);

      select
         Contador_T_Cómputo.Wait;
         Fallo_T_Funcional := True;
         Tratar_Fallo(Fallo_T_Funcional, Result_Ejecución);
      then abort
         select
            delay until Siguiente_Activación + Plazo_Respuesta;
            Fallo_T_Funcional := True;
```

```
            Tratar_Fallo(Fallo_T_Funcional, Result_Ejecución);
         then abort
            Tarea_Funcional.Fin_Activación(Result_Ejecución);
            if Result_Ejecución <> Global.Correcta then
               Tratar_Fallo(Fallo_T_Funcional, Result_Ejecución);
            end if;
         end select;
      end select;

      exit when Fallo_T_Funcional or Result_Ejecución <> Global.Correcta;

      Siguiente_Activación := Siguiente_Activación + Perfil_Ejecución.Período;

      Gestor_Modo_Tarea.Ejecutar_Cambio_Modo_Sup
         (Info_Modo_Tarea, Siguiente_Activación);

   end loop;

exception
   when others
      Fallo_T_Funcional := True;
      Tratar_Fallo(Fallo_T_Funcional, Result_Ejecución);
end Tarea_Supervisora;
```

Las modificaciones realizadas a la tarea períodica no impiden que se comporte de acuerdo a las hipótesis del análisis de planificabilidad. Unicamente se ha aumentado el tiempo de cómputo en cada activación.

6.5 Análisis de la planificabilidad

Como ya se ha indicado, una de las grandes ventajas de los métodos de planificación basados en prioridades estática es que permite comprobar analíticamente la planificabilidad de un conjunto de tareas. Los esquemas de tareas se han diseñado de forma que se puede saber en todo momento a cúal se contabiliza el tiempo de cómputo consumido en cada operación y, por consiguiente, se puede realizar el análisis.

Los problemas reales son calcular los tiempo de cómputos y de bloqueo de las tareas y analizar los casos en que éstas se desvían del comportamiento adecuado. A continuación se muestra como realizar estas cuestiones.

6.5.1 Cálculo del tiempo de cómputo de una tarea.

La medida del tiempo de cómputo de una tarea no es una labor evidente e inmediata. En [AdaPI90] se presentan fundamentos y métodos para su realización. Con respecto al ejecutivo multitarea, el tiempo máximo de cómputo es el máximo del medido en las siguientes circunstancias:

- Ejecución normal de la tarea. (Sin cambios de modo ni errores).
- Ejecución de la tarea cuando se produce un cambio de modo, hasta que se completa el algoritmo de cambio.
- Ejecución de la tarea, cuando se detectan fallos y se tratan.

En el cálculo del tiempo de cómputo de las tarea esporádicas, se debe considerar a cuál se contabiliza el tiempo de cómputo consumido en la notificación del evento.

La tarea gestora de inactivas debe incluirse en el análisis de planificabilidad como una más y según el perfil reseñado anteriormente. Es importante recordar que el tiempo de cómputo que consumen las tareas inactivas para cambiar al modo de ejecución, se contabiliza a esta tarea.

Dos factores adicionales a tener en cuenta están relacionados con tiempos de cómputo del núcleo básico. Estos valores conviene contabilizarlos, aunque en muchos casos se pueden despreciar.

- *Cambios de contexto.* Cuando una tarea pasa a ejecutar, en general, expulsará a otra del procesador. Para contabilizar este tiempo de cómputo, se debe añadir el valor de cuatro cambios de contexto al valor de tiempo calculado anteriormente. Nótese que cada tarea de tiempo real está formada por dos tareas de Ada, que introducen los cambios de contexto adicionales.
- *Temporizadores que expiran.* Las tarea establecen temporizadores para determinar cuando se deben activar. Cuando expiran, se produce una interrupción que trata el núcleo básico. Las operaciones habituales serán pasar la tarea en cuestión de la cola de retardos *(delay queue)* a la cola de tarea listas para ejecutar o ejecutarlo, si es la más prioritaria. A cada tarea se le debe añadir al tiempo de cómputo antes calculado, el valor de esta temporización previa a cada activación.

  Si esta situación la provoca una tarea de menor prioridad, el efecto es una cierta inversión de prioridad. Por tanto, habrá que incrementar el tiempo de cómputo calculado con el tiempo de cómputo de los temporizadores de tareas de menor prioridad que pueden expirar en una activación.

6.5.2 Cálculo del tiempo de bloqueo de una tarea.

Las directrices que se presentan para calcular tiempos de bloqueo de las tareas son aplicables tanto si se tienen tiempos de cómputo estimados o reales. Al calcular el tiempo de bloqueo de una tarea, para el análisis de planificabilidad propuesto, hay que tener en cuenta que cuando se produce inversión de prioridad, se ven afectados todos las tarea listas con prioridad intermedia entre la que debería ejecutar y la que ha producido la inversión de prioridad.

Una tarea de tiempo real lista para ejecutar estará bloqueada por dos razones básicas:

- La tarea que se ejecuta pertenece a una tarea de tiempo real más prioritaria. Este tiempo está contemplado en el análisis de planificabilidad y no se contabiliza como tiempo de bloqueo.
- La tarea que se ejecuta no pertenece a una tarea de tiempo real más prioritaria. En este caso se produce inversión de prioridad. En el contexto de la teoría de planificación utilizada, el tiempo de bloqueo de una tarea está relacionado con este fenómeno.

En los esquemas de tareas presentados, se produce inversión de prioridad en las siguientes circunstancias:

- *Tarea supervisora.* Las tareas de tiempo real se implementan mediante dos tareas Ada, una que supervisa la ejecución de la actividad del usuario, para que se comporte de acuerdo al perfil, y otra que ejecuta esta actividad. La segunda tarea tiene la prioridad que le corresponde de acuerdo con el método de planificación. La primera tiene prioridad mayor que todas las tareas funcionales. Esto es así porque esta tarea debe detectar los errores en la ejecución de la primera y comunicarlos inmediatamente a la tarea de recuperación correspondiente.

  Por tanto, la tarea supervisora puede provocar inversión de prioridad cuando ejecuta. Este efecto se admite en aquellos casos en que es vital tratar un posible fallo tan pronto como sea posible. Para analizar la influencia de la tarea supervisora en este problema, se consideran tres casos:

  - *Ejecución sin errores.* Cuando la tarea que ejecuta la actividad del usuario completa un activación, comunica este hecho a la tarea supervisora.y se ejecuta inmediatamente. Si en esta situación queda lista para ejecutar una tarea más prioritaria se produce inversión de prioridad. Su duración está acotada por el tiempo de cómputo consumido por la tarea supervisora, para comprobar si se han producido errores en el grupo o cambio de modo del sistema.
  - *Ejecución con errores de la tarea funcional.* Si se produce un error de ejecución de la tarea funcional, inmediatamente se comunica a la tarea supervisora, para que se ejecuten las acciones de recuperación de fallos. Por tanto, la situación es similar a la anterior, ya que la tarea supervisora sólo ejecuta después de la de actividad y las características de la inversión de prioridad en este caso son análogas al anterior.
  - *Incumplimiento de plazos de respuesta.* Este caso es más complejo de considerar. Cuando la tarea funcional excede el plazo de respuesta expira un temporizador y ejecuta la tarea supervisora. Desde el punto de vista de una tarea más prioritaria, es impredecible cuando puede ocurrir.

- *Comunicación entre tareas.* La comunicación entre tareas provoca inversiones de prioridades. El protocolo de herencia del techo de prioridad permite acotar su duración y garantiza que una tarea sólo se sufrirá este problema una vez en cada activación. La duración de la inversión de prioridad en este caso coincide con la

duración máxima entre las regiones críticas protegidas por objetos de comunicación con techo de prioridad mayor que la prioridad de la tarea en cuestión y que son usados por tareas con menor prioridad que ella.

Para analizar la planificabilidad es necesario acotar estos tiempos de bloqueo. La inversión de prioridad producida cuando una tarea supervisora detecta el incumplimiento de un plazo se tratará posteriormente. En primer lugar se cuantifican los bloqueos con cota conocida, que son las inversiones de prioridad producidas por la comunicación entre tareas o por una tarea supervisora y que tienen las siguientes propiedades:

1. *Son excluyentes.* Ambas ocurren como respuesta a una acción de la tarea funcional. En efecto:

   - La tarea funcional termina su ejecución en la presente activación y se lo comunica a la tarea supervisora para que haga las comprobaciones habituales.
   - Falla la ejecución de la tarea funcional y la tarea supervisora se ejecuta para realizar las acciones de tratamiento adecuadas.
   - La tarea funcional realiza operaciones de comunicación.

   Por consiguiente una tarea funcional no podrá provocar inversión de prioridad de los dos tipos simultáneamente.

2. *Una tarea sólo será bloqueada una vez en cada activación.* Como el bloqueo se produce como consecuencia de una acción de una tarea funcional menos prioritaria, está se habrá producido antes de la activación de la tarea más prioritaria. Hasta que finalice esta activación ninguna otra tarea funcional menos prioritaria se ejecutará y por consiguiente no se volverá a bloquear a la tarea por este motivo.

El tiempo máximo de bloqueo, por estas causas, en la activación de una tarea es:

$$\max(t_{comunic}, t_{supervisora})$$

Donde:

- $t_{supervisora}$: el tiempo de cómputo máximo de una tarea supervisora en ejecutar las comprobaciones pertinentes al final de cada activación y ejecutar las operaciones de tratamiento de fallos.

- $t_{comunic}$: el tiempo de cómputo mayor entre la regiones críticas protegidas por objetos de comunicación con techo de prioridad mayor que la prioridad de la tarea en cuestión y que son usados por tareas con menor prioridad que ella.

En este apartado no se ha considerado el efecto de operaciones como tratamiento de señales o interrupciones que podrían producir casos adicionales de inversión de prioridad. El comportamiento de estas operaciones depende del entorno de ejecución seleccionado, por lo que no parece sencillo dar un método concreto de tratarlo. En cualquier caso, las directrices propuestas pueden guiar el desarrollo del método de tratamiento en casos particulares.

6.5.3 Activación autónoma de la tarea supervisora.

El caso más problemático es la inversión de prioridad producida cuando una tarea supervisora detecta un fallo de temporización. El efecto de esta tarea no se puede incluir en el análisis de planificabilidad porque su prioridad no es la que le corresponde por sus características de ejecución y no se puede predecir cuando se ejecutará. Por consiguiente, no se pueden garantizar los plazos con absoluta certeza.

La consecuencia es que el análisis de planificabilidad puede determinar que el conjunto cumplirá los plazos de respuesta, pero puede ser falso en algunas circunstancias. En efecto, puede ocurrir que una tarea de baja prioridad incumpla un plazo de respuesta. Entonces se ejecuta la tarea supervisora, que expulsaría a la tarea más prioritaria y, posiblemente, impida que cumpla su plazo de respuesta. Este incumplimiento provoca la ejecución de otra tarea supervisora y podría producir una reacción en cadena.

Esta situación sólo acontece si concurren una serie de circunstancias desfavorables, como son que todas las tarea se ejecuten con el tiempo de cómputo máximo y que esta bloqueo en cuestión evite que otra tarea cumpla su plazo de respuesta. Estas condiciones no ocurrirán habitualmente, más aún teniendo en cuenta que la duración de esta inversión de prioridad es muy pequeña. por lo que en determinados casos se puede despreciar su efecto. Se sabe que puede ocurrir y se deja que los mecanismos de tratamiento de fallos traten esta situación. No hay que olvidar, que esta situación ocurre porque se quiere tratar inmediatamente los fallos de ejecución de las tareas.

En sistema críticos en cuanto a seguridad, será necesario considerar este efecto. Una posible solución consiste en reservar una cierta cantidad de tiempo para tratar el problema. De esta forma, se incluye el estudio de su efecto sobre el sistema en el análisis de planificabilidad. Para tal fin, se puede crear una tarea ficticia con período igual a la tarea más prioritaria y el tiempo de cómputo necesario para tratar el caso más desfavorable. Sin embargo, la capacidad de cómputo reservada podría ser excesiva.

En conclusión, la mejor solución a este problema es reservar una cantidad de tiempo que no sea excesiva, y en el improbable caso de que concurran las circunstancia más desfavorables, dejar que los mecanismos de tratamiento de fallos gestionen la situación.

6.6 Conclusiones

En este capítulo se han presentado un conjunto de esquemas correspondientes a los componentes más importantes de los sistema de tiempo real críticos basados en prioridades. En el diseño de estos esquemas se han tenido en cuenta los estudios más recientes sobre este tema, presentados en el capítulo 4.

Los esquemas satisfacen los requisitos funcionales identificados, lo que permite augurar su aplicación en sistemas industriales. Por otro lado, cumplen las hipótesis de comportamiento de los métodos de planificación basados en prioridades. Esto permite aprovechar una de las grandes ventajas de estos métodos, que es la posibilidad de garantizar los plazos de respuesta, mediante la determinación analítica de la planificabilidad del sistema.

Un criterio básico que ha guiado el desarrollo de estos esquemas, es que en su programación se emplearan servicios proporcionados por núcleos de ejecución normalizados. Es decir, no se han empleado características del sistema operativo especialmente dirigidas a la realización de alguno de los protocolos, si no están disponibles en la realidad. Por este motivo se puede afirmar que, aunque los esquemas de las tareas se han desarrollado en Ada 9X, su adaptación a otros núcleos de ejecución debería ser relativamente sencilla.

Una parte de este trabajo ha consistido en identificar los requisitos funcionales mínimos que deben cumplir los núcleos de ejecución para realizar sistemas de tiempo real críticos basados en prioridades. En la tabla siguiente se indican las características requeridas por programar una determinada funcionalidad de estos sistemas, de forma incremental.

Funcionalidad.		**Características requeridas**
Método de planificación	Planificación de recursos	Concepto de prioridad básico en el sistema.
	Análisis de planificabilidad	Comportamiento determinista, tiempos de ejecución de servicios del núcleo conocidos, acotados y suficientemente pequeños.
Tarea Periódica	Activación	Temporizadores absolutos con precisión suficiente
Tarea Esporádica	Notificación evento	Comunicación entre tareas, interrupciones, señales.
Comunicación entre tareas	Pasiva	Protocolo del techo de prioridad
	Activa	Mecanismo basado en paso de mensajes
Detección de Fallos	Hardware-Sistema Operativo	Mecanismos de tratamiento de fallos del entorno de ejecución
	Tiempo de cómputo	Temporizadores relativos al tiempo de cómputo de las tareas
	Plazo de respuesta	Temporizadores absolutos.
Tratamiento de fallos	Tratamiento básico	Transferencia asíncrona de control asociada a los temporizadores.
	Grupo de recuperación	———
Cambio de Modo	Cambio del perfil	Prioridades dinámicas

Capítulo 7

Reemplazamiento dinámico de software en sistemas de tiempo real críticos.

Existen sistemas informáticos cuya detención puede provocar daños económicos y materiales. Sin embargo, durante la vida operativa de estos sistemas, es necesario cambiar partes de la aplicación bien para corregir errores en el código, bien para adaptar su funcionalidad a requisitos nuevos [Bae91].

El objetivo de este capítulo es presentar un método de reemplazamiento dinámico de software en sistemas de tiempo real, como solución parcial a la modificación de sistemas sin posibilidad de detención. Para tal fin, se introduce la problemática asociada y algunas de las soluciones desarrolladas, se presentan las líneas maestras de un protocolo de reemplazamiento dinámico en sistemas generales y finalmente un modelo original para reemplazar componentes de software en sistemas de tiempo real crítico.

7.1 Planteamiento del problema.

El mantenimiento del software en sistemas informáticos es la fase más extensa de su ciclo de vida [Sommerville89]. Esta fase incluye al conjunto de actividades necesarias para modificar un sistema después de haberse completado y estar en explotación. Entre estas modificaciones suelen considerarse desde cambios del código para resolver fallos de codificación, hasta el rediseño del sistema para ampliar su funcionalidad.

La incorporación de estos cambios en un sistema que está funcionando se suele hacer de forma estática. En primer lugar se genera una nueva versión del sistema, se detiene la versión en ejecución y se instala y se inicia la nueva.

Este enfoque no es siempre posible. Algunos sistemas informáticos no se pueden detener sin provocar daños materiales y económicos. Un ejemplo lo constituyen cierto tipo de aplicaciones espaciales. El tiempo de vida operativa estimado a la estación espacial *Columbus* es de treinta años y la parada de cualquier sistema de abordo podría generar graves problemas. Esta situación es también común en ciertos sistemas de telecomunicaciones que requieren un funcionamiento continuo para proporcionar un servicio

adecuado.

La alternativa consiste en diseñar los sistemas como un conjunto de componentes reemplazables que se pueden sustituir sin necesidad de detener el sistema. La investigación sobre las técnicas de diseño y realización de este tipo de sistemas es reciente y aún no hay muchos trabajos publicados. En la mayoría de éstos, se aborda el tema desde la perspectiva de sistemas de propósito general distribuidos.

Uno de los primeros trabajos sobre este tema fue desarrollado bajo contrato de la Agencia Espacial Europea [Amador&88]. En él se analiza el concepto de Unidad Reemplazable de Software y se define un modelo de reemplazamiento. En el citado documento, se define el concepto de Unidad Reemplazable de Software (URS) como:

> una porción de código, datos o texto que se puede tratar y considerar como una unidad que puede ser reemplazada dentro de su entorno operativo bajo determinadas condiciones.

Esto quiere decir que se puede reemplazar tanto código como datos y texto. Sin embargo, en el resto de este documento sólo se va a considerar el reemplazamiento de software, por ser el más complicado, desde el punto de vista técnico.

En la Universidad Politécnica de Madrid se continuó este trabajo, según dos líneas de desarrollo:

1. El estudio del concepto de URS y la integración de este concepto con el método de diseño HOOD [Amador&91].

2. La definición de un método básico de diseño de URS y la implementación del modelo de reemplazamiento en Ada y el desarrollo de componentes de software de apoyo al desarrollo de URS [Vicente90] [Vicente&91].

Estos desarrollos forman el punto de partida del estudio que se presenta en este documento y se comentarán brevemente con posterioridad.

El desarrollo de sistemas reconfigurables [Kramer92], de los que CONIC [Magee&89] es pionero, forman la base de otros trabajos sobre sistemas con componentes reemplazables. Los sistemas reconfigurables se basan en la separación de la descripción funcional del comportamiento de las tareas individuales, de la descripción de la estructura del sistema, contemplado como un conjunto de tareas y sus interconexiones. El desarrollo de aplicaciones distribuidas se realiza en dos niveles: funcional y de configuración. Éste permite expresar la estructura inicial del sistema y cambios en esta estructura, lo que permite realizar operaciones de reconfiguración.

En este marco de referencia, [Etzkorn92] describe las líneas maestras de un enfoque de reemplazamiento que se basa en especificar en el nivel estructural una serie de estados en los que es posible el reemplazamiento. La especificación de estos estados se realiza en base al comportamiento externo observable de los componentes involucrados. El objetivo de este enfoque es evitar que las unidades funcionales deban incluir código específico para el reemplazamiento. Las ideas presentadas son exploratorias y, en palabras del autor, es necesario refinarlas antes de su aplicación.

En la Universidad de York también se ha estudiado el reemplazamiento de software [Tindell90] [Burns&91a]. En estos documentos se analiza, principalmente, el código que puede ser reemplazado y en qué circunstancias.

Las soluciones técnicas presentadas no son aplicables en sistemas de tiempo real críticos. Todas ellas se desarrollan en un sistema distribuido complejo y suponen una sobrecarga computacional importante, distribuida (en el sentido que no se sabe con certeza a que unidad contabilizar el tiempo de cómputo consumido) y esporádica al sistema. En consecuencia no se puede analizar la planificabilidad del sistema y garantizar el cumplimiento de los plazos de respuesta de las tareas cuando se reemplazan componentes dinámicamente.

7.2 Conceptos básicos

7.2.1 Requisitos del reemplazamiento dinámico.

El reemplazamiento dinámico de componentes de software permite sustituir estos elementos, cuando su funcionalidad no es la adecuada o cuando son erróneos, sin necesidad de detener la ejecución del resto del sistema. Por tanto, la aplicación continuará realizando sus funciones y se evitan las consecuencias negativas de su detención. Sin embargo, el reemplazamiento de un componente supone un estado transitorio del sistema durante el cuál su comportamiento puede degradarse levemente, especialmente en lo que respecta al componente reemplazado y a aquellos directamente relacionados con él. Por tanto el protocolo de reemplazamiento debe tratar de minimizar este efecto.

Partiendo de esta premisa fundamental, se pueden deducir una serie de requisitos exigibles a un protocolo de reemplazamiento dinámico:

- *Operación transparente al resto de los componentes.* Los componentes de la aplicación que no están directamente involucrados en una operación de reemplazamiento no deben percibir que se está realizando esta operación. Así, por ejemplo, si el componente reemplazado prestaba servicios a otros elementos del sistema, podría ocurrir que en el momento del reemplazamiento hubiera peticiones pendientes. El reemplazamiento se debe realizar de forma que estas peticiones sean tratadas por el nuevo componente, sin que los clientes noten el cambio.

- *Mantenimiento de la consistencia del componente reemplazado.* Un componente se debe reemplazar únicamente cuando su estado y sus datos sean consistentes. Es evidente que, en general, no se puede parar asíncronamente la ejecución del componente que va a ser reemplazado, pues pueden interrumpirse operaciones en curso y, como consecuencia, corromper los datos o producir daños en el entorno con el que interacciona.

- *Conservación del estado de ejecución del componente reemplazado.* Una característica deseable para asegurar la continuidad de la funcionalidad prestada por

los componentes involucrados en una operación de reemplazamiento, es que el componente nuevo se ejecute teniendo en cuenta el estado del reemplazado.

- *No pueden estar activos los dos componentes al mismo tiempo.* Es previsible que el componente reemplazado y el nuevo realizen operaciones sobre datos e interaccionen con otros elementos comunes. En este caso, es previsible que el diseño de la aplicación no contemplará esta actuación simultánea. La consecuencia probable es que se produzcan efectos no deseados que pongan en peligro la ejecución correcta del sistema. El proceso de reemplazamiento debe asegurar que esta situación no se dará. En cada momento uno y sólo uno de los componentes involucrados realizará las operaciones relativas a su funcionalidad específica.

Además de estos requisitos básicos, un protocolo de reemplazamiento para sistemas de tiempo real crítico debe ser consistente con el método de planificación empleado. En particular, un protocolo será aplicable en sistemas de tiempo real críticos diseñados en base a los métodos de planificación basados en prioridades si se cumple la siguiente condición:

Sean $\mathcal{P} = \{\tau_1, \tau_2, \ldots, \tau_i, \ldots, \tau_n\}$ y $\mathcal{P}' = \{\tau_1, \tau_2, \ldots, \tau'_i, \ldots, \tau_n\}$ dos conjuntos de tareas. $\mathcal{P}'$ se obtiene reemplazando en $\mathcal{P}$, la tarea τ_i por τ'_i. Si los plazos de respuesta de las tareas de $\mathcal{P}$ y de $\mathcal{P}'$ se pueden garantizar, entonces estos plazos de respuesta también se deberán garantizar durante y después del reemplazamiento dinámico de τ_i por τ'_i.

7.2.2 Unidad de reemplazamiento

Una cuestión básica cuando se aborda el problema de reemplazamiento es determinar cuál es la unidad mínima reemplazable. Entonces se puede prever el impacto de esta operación en el resto del sistema y se puede definir un modelo de reemplazamiento acorde.

El modelo de sistema en el que se definirán operaciones de reemplazamiento es un sistema lógicamente distribuido que se ejecuta en un sistema monoprocesador, y por tanto, físicamente no distribuido [Bal90]. Un sistema lógicamente distribuido está compuesto por múltiples objetos activos (tareas) que se comunican mediante un mecanismo de paso de mensajes.

El reemplazamiento en sistemas lógicamente no distribuidos, es decir, sistemas con comunicación por memoria compartida es menos general y plantea dificultades difíciles de resolver. Si la unidad en la que se declaran los datos compartidos es activa, i.e. una tarea, su reemplazamiento implica que los datos no estarán accesibles durante un cierto tiempo y la posibilidad de que los mecanismos de acceso tengan que modificarse al variar la unidad que incluye los datos compartidos. Si la unidad es pasiva, entonces no podrá realizar por sí misma operación alguna de reemplazamiento, complicando considerablemente el código de gestión del protocolo.

El modelo de sistemas lógicamente distribuido es el seleccionado por la práctica totalidad de los trabajos relacionados con el diseño de sistemas reconfigurables dinámicamente

[Kramer92]. A la unidad de distribución se le denominará nodo virtual[1] y se caracteriza como sigue [Burns&89][Atkinson&88]:

- Es la unidad de modularidad en sistemas distribuidos.
- Es la unidad de configuración y reconfiguración.
- Proporciona interfases bien definidos para la comunicación con otros nodos virtuales.
- Encapsula completamente su estado interno y no permite el acceso directo de otros nodos virtuales a sus variables internas.
- La comunicación con otros nodos virtuales se realiza mediante las operaciones que se expresan en el interfaz de los nodos virtuales. Esta comunicación se basa en algún mecanismo de paso de mensajes.
- No debe referirse a componentes compartidos con estado interno. Si comparten componentes con otros nodos, estos elementos compartidos no deben tener estado interno.
- Debe tener su propio flujo de control.
- Varios nodos virtuales se pueden ejecutar en la misma máquina física.

El nodo virtual es una entidad apropiada como unidad mínima de reemplazamiento por las siguientes razones:

- *Flexibilidad.* El concepto de nodo virtual admite múltiples realizaciones prácticas. En cuanto a su estructura interna, puede estar compuesto por una o varias tareas. Puede emplearse tanto para el desarrollo de sistemas distribuidos como sistemas centralizados. En consecuencia, el tamaño y complejidad del nodo virtual es muy variable y, por tanto permite adecuar sus características al tipo de sistema en desarrollo.
- *Metodologías de diseño de software.* El nodo virtual se adapta a metodologías de diseño orientadas a objetos [Hood89a] [Hood89b] [Sommerville89] y funcionales [Sommerville89] [Ward&85].
- *Adecuación a los requisitos de reemplazamiento.* Las características reseñadas de los nodos virtuales permiten su adaptación a los requisitos básicos de un protocolo de reemplazamiento. En concreto, es posible emplear mecanismos de comunicación entre nodos virtuales que permitan el reemplazamiento transparente y se puede dotar a un nodo virtual de la funcionalidad necesaria para asegurar que sólo se produce reemplazamiento desde los estados consistentes y para arrancar la ejecución a partir de un estado determinado.

[1] El nombre con el que se designa a la unidad de distribución varía en los diferentes trabajos. Sin embargo, sus características generales son muy similares.

Posteriormente se relacionará este modelo de sistema formado por unidades reemplazables con el modelo de sistema de tiempo real empleado en el capítulo anterior.

7.2.3 Apoyo del sistema operativo

Una primera reflexión sobre el proceso de reemplazamiento permite determinar una secuencia general de pasos necesarios:

1. El sistema recibe una petición externa de reemplazamiento.
2. Se asigna espacio de memoria para el software que hay que cargar.
3. Se carga el código correspondiente a la unidad reemplazadora y se crea ésta.
4. Cuando la unidad a reemplazar está en un estado consistente, se transfieren los datos de estado a la unidad reemplazadora.
5. Los mensajes enviados a la unidad reemplazada deben llegar a la unidad reemplazadora.
6. Liberación de la memoria correspondiente a la unidad reemplazada.

Esta secuencia de operaciones implica la necesidad de emplear un núcleo de ejecución que proporcione determinada funcionalidad de apoyo al protocolo de reemplazamiento. La complejidad de estas funciones de apoyo es muy variable. En un extremo el núcleo puede proporcionar primitivas que a partir de información de la unidad a reemplazar y la reemplazadora, ejecuten automáticamente el protocolo de reemplazamiento, sin necesidad de incluir código específico para la ejecución del protocolo en las unidades involucradas. El extremo contrario es un núcleo que proporcione cierta funcionalidad necesaria, pero que no intervenga diréctamente en las operaciones de reemplazamiento.

La alternativa más flexible es la segunda, pues impone menos requisitos al núcleo de ejecución y permite programar los protocolos de reemplazamiento con las características específicas de cada sistema. Además, es la alternativa más adecuada en sistemas de tiempo real crítico, donde es interesante disponer de núcleos con tiempos de cómputo y comportamiento determinista. Si el sistema operativo realiza las funciones de carga, creación, reemplazamiento y liberación de la memoria es muy probable que no se sepa con certeza su duración y el instante en que ocurren y, consecuentemente, sea muy complicado considerarlas en el análisis de planificabilidad.

La funcionalidad básica que debe proporcionar un núcleo de ejecución para llevar a cabo operaciones de reemplazamiento es la siguiente:

1. Gestión dinámica de memoria. Se debe asignar espacio para la ejecución de las unidades nuevas y disponer de la capacidad liberada por las unidades reemplazadas. Además, si la vida del sistema es larga, es razonable pensar que se realizará un cierto número de operaciones de reemplazamiento. Por consiguiente, será necesario gestionar dinámicamente la memoria, de forma que la memoria liberada pueda

volverse a reutilizar. Esta gestión podría involucrar ocasionalmente su compactación.

2. Gestión dinámica de tareas. La ejecución de una unidad implica la creación del conjunto de tareas concurrentes que la forman. Análogamente, el borrado de una unidad implica el borrado de sus tareas internas.

3. Mecanismo de comunicación indirecta entre las tareas [Goscinski91] [Tanenbaum92]. El núcleo debe proporcionar directamente o permitir la realización de mecanismos de comunicación indirectos entre tareas. Este tipo de comunicación se caracteriza porque los mensajes no se intercambian diréctamente entre el emisor y el receptor, sino que se emplea un objeto intermedio en el que el emisor deposita el mensaje y del que el receptor lo extrae. Las colas de mensajes son un ejemplo de este tipo de objetos. Esta alternativa facilita la realización de operaciones de reemplazamiento transparentes. Si se sustituye al receptor, la nueva unidad continuará tratando los mensajes pendientes en la cola, sin que los emisores perciban la sustitución.

7.3 Protocolo de reemplazamiento dinámico de software

En este apartado se va a describir sucintamente el protocolo de reemplazamiento inicialmente propuesto en [Amador&88] y posteriormente refinado e implementado en Ada [Vicente90] [Vicente&91] [Amador&91]. Este protocolo es el origen del protocolo de reemplazamiento en sistemas de tiempo real que se describirá con detalle posteriormente. Se basa en la definición de una unidad global al sistema que gestiona las operaciones de reemplazamiento y en la especificación de la funcionalidad necesaria para convertir un módulo software en reemplazable.

Una aplicación esta compuesta por unidades reemplazables y se ejecuta con un sistema operativo de propósito general, que proporciona la funcionalidad descrita en el apartado 7.2.3. La carga y ejecución de las unidades reemplazadoras no presenta dificultad alguna, pues se realiza con las primitivas habituales, en concurrencia con la ejecución de la aplicación.

La unidad de reemplazamiento es el proceso del sistema operativo. La comunicación entre procesos es indirecta y se realiza mediante colas de mensajes proporcionados por el sistema operativo.

La gestión del reemplazamiento se centraliza en un supervisor de reemplazamiento. Este componente es responsable de iniciar el protocolo de reemplazamiento y de comprobar la correcta realización del mismo. Se han identificado [Amador&88] [Vicente90] cinco primitivas de reemplazamiento que las unidades reemplazables deben aceptar y tratar adecuadamente:

- `Parar`. Detiene la ejecución de la unidad, después de completarse la operación de reemplazamiento.

- `Ejecutar`. La unidad comienza a ejecutar a partir de la información de estado recibida por la unidad a reemplazar.
- `Continuar`. La unidad continúa la ejecución teniendo en cuenta el estado en que se encontraba cuando se paró la ejecución para ejecutar las operaciones de reemplazamiento.
- `Almacenar_Estado`. Cuando la unidad está cosistente recopila la información de estado, se comunica con la unidad reemplazadora y, finalmente, confirma al supervisor el resultado de la operación.
- `Recoger_Estado`. La unidad reemplazadora se comunica con la unidad a reemplazar para recibir su estado interno y confirma el resultado de la operación al supervisor.

Todas las primitivas anteriores, a excepción de `Parar`, implican la respuesta a la unidad supervisora, indicando la corrección de la operación. De esta forma, puede saber si el protocolo se ha ejecutado corréctamente y, en caso contrario, cancelar la operación de reemplazamiento. La comunicación entre las unidades implicadas se realiza mediante colas de mensajes dedicados.

A partir de estas consideraciones, el protocolo de reemplazamiento diseñado consta de las siguientes fases [Vicente90]:

1. *Almacenamiento del estado interno*
   La unidad supervisora recibe un comando de reemplazamiento. Como consecuencia, comunica a la unidad a reemplazar la orden de almacenamiento del estado interno. Cuando haya completado esta operación, la unidad reemplazadora lo comunica a la unidad supervisora.

2. *Entrega del estado interno*
   El supervisor le indica a la unidad reemplazadora que debe adquirir el estado interno. Esta acción se produce como resultado de una interacción entre las dos unidades implicadas. Finalmente, la unidad reemplazadora comunica al supervisor la conclusión de la operación.

3. *La unidad reemplazada abandona la ejecución*
   Si las operaciones anteriores se han realizado correctamente, el supervisor indica a la unidad reemplazada que concluya su ejecución.

   Si la comunicación entre las unidades hubiera fallado, la unidad supervisora ordena a la unidad reemplazada que continúe la operación.

4. *La unidad reemplazada inicial la ejecución*
   Finalmente, la unidad reemplazadora recibirá la orden inicio de la ejecución del código propio de la aplicación.

7.4 Protocolo de reemplazamiento dinámico de software en sistemas de tiempo real críticos.

7.4.1 Modelo del sistema.

El objetivo de esta sección es plantear un protocolo de reemplazamiento dinámico [Alonso&93] adecuado para sistemas de tiempo real crítico. Este protocolo debe satisfacer los requisitos enumerados en el apartado 7.2.1 y, en especial, el relativo al cumplimiento de los plazos de respuesta de las tareas durante y después de una operacion de reemplazamiento.

El modelo de sistemas de tiempo real en el que se quiere integrar el protocolo de reemplazamiento dinámico coincide con el descrito en el capítulo 4. Este modelo es compatible con el modelo de sistema formado por unidades reemplazables descrito en el apartado 7.2.2, teniendo en cuenta las siguientes consideraciones:

- El nodo virtual se realiza prácticamente como una única tarea. Esta tarea podrá ser de cualquiera de los tipos existentes.
- La comunicación se realiza mediante paso de mensajes y siguiendo el modelo de cliente-servidor, según el esquema presentado en 6.2.2. Si el núcleo de ejecución no proporciona mecanismos indirectos de comunicación entre tareas, su realización mediante mecanismos de paso de mensajes es inmediata. En cualquier caso, hay que tener en cuenta que el acceso al objeto indirecto de comunicación se debe realizar según el protocolo del techo de prioridad.

7.4.2 Crítica al protocolo de reemplazamiento general.

El protocolo de reemplazamiento anterior es adecuado para desarrollar aplicaciones convencionales. Sin embargo, cuando se acometen sistemas empotrados o sistemas de tiempo real, aparecen una serie de problemas que no se han tenido en cuenta. A continuación se realiza una análisis crítico de problemas que necesitan ser resueltos para poder reemplazar software dinámicamente en este tipo de sistemas:

1. *Protocolo complejo y centralizado.* La unidad supervisora de reemplazamiento dirige el protocolo de reemplazamiento llamando ordenadamente a las primitivas de reemplazamiento de las unidades implicadas. La ejecución de las primitivas finaliza con un mensaje de reconocimiento a la supervisora. Si las unidades implicadas cooperan más activamente en el protocolo de reemplazamiento, parte de los mensajes intercambiados se pueden eliminar y algunas fases del protocolo pueden simplificarse.
2. *Ejecución indeterminista de la supervisora.* El protocolo de reemplazamiento no determina los instantes de tiempo concretos, ni la prioridad prefijada de ejecución de la supervisora de reemplazamiento. Cuando éste recibe una petición ejecuta el

protocolo sin restricción alguna. Este comportamiento no es adecuado cuando se desarrollan sistemas de tiempo real. La ejecución indeterminista de la supervisora puede provocar inversión de prioridades y provocar que alguna tarea de tiempo real crítica incumpla sus plazos de respuesta.

3. *Ejecución de funciones del sistema.* El reemplazamiento dinámico implica la gestión de memoria y la creación y borrado de tareas dinámicamente. La concurrencia de estas operaciones con la ejecución de la aplicación no plantea problemas en sistemas operativos convencionales. En general, la sobrecarga que se produce no será suficientemente importante como para desaconsejar este paralelismo.

   En sistemas de tiempo real las operaciones de carga, creación e iniciación de la unidad reemplazadora se tienen que ejecutar de forma que se garantice que no se va interferir con el cumplimiento de las especificaciones temporales y funcionales de la unidad

4. *Sobrecarga.* El protocolo de reemplazamiento no restringe la estructura interna de las unidades de reemplazamiento, porque no es importante para su validez. Las características específicas de los sistemas de tiempo real hacen recomendable modelar una unidad reemplazable mediante un sólo componente activo.

7.4.3 Protocolo de reemplazamiento dinámico.

Consideraciones previas.

Antes de detallar el protocolo de reemplazamiento, es necesario realizar unas consideraciones previas sobre el sistema donde se van a reemplazar dinámicamente unidades software:

- Estructura de la unidad reemplazable.
  La unidad reemplazable es la tarea de tiempo real. Teniendo en cuenta su estructura, se puede asegurar que una tarea se encuentra en un estado consistente al final de una activación. Por consiguiente, una tarea ejecutará las operaciones de reemplazamiento cuando se encuentre en esta situación.

- Apoyo del sistema operativo.
  El sistema de ejecución debe proporcionar cierta funcionalidad que no es completamente necesaria, ni a veces deseable, para desarrollar sistemas de tiempo real tradicionales:

  - Creación y borrado de tareas dinámicamente. Para poder reemplazar unidades, hay que crear unidades nuevas. Las unidades reemplazadas, por tanto, se deben borrar para poder disponer de los recursos que ocupan.
  - Gestión de memoria dinámica. Es necesario asignar memoria para cargar las unidades reemplazadoras y ser capaces de disponer de la memoria liberada por las unidades reemplazadas. Ocasinalmente, puede ser necesario compactar la memoria para evitar que su fragmentación impida su empleo óptimo.

Estas operaciones no suelen ser deseables porque se corre el riesgo de que el sistema se quede si recursos y no pueda ejecutar convenientemente y porque el sistema operativo puede realizar operaciones asociadas a la funcionalidad anterior de forma no determinista y no integrada en el esquema de prioridades, lo que puede dar lugar a que algunas tareas no cumplan sus plazos de respuesta.

El problema de la disponibilidad de recursos (memoria o posibilidad de crear tareas adicionales) no debe ocurrir, pues se sabe con anticipación los recursos que necesitará el sistema después de realizar una operación de reemplazamiento y por tanto se puede predecir si estarán disponibles. En cuanto ejecución de estas operaciones de forma integrada con el esquema de prioridades se puede resolver haciendo que se ejecuten a petición de una tarea de tiempo real dedicada a esta labor.

Las operaciones de compactación de memoria son problemáticas por las mismas razones expuestas anteriormente. El enfoque para su realización práctica también implica su ejecución a petición de un tarea de tiempo real dedicada. Por otro lado, el encapsulamiento de la información de un nodo virtual permite augurar que será factible cambiar su situación en memoria de forma sencilla y eficiente.

- Unidad supervisora de reemplazamiento.
  El protocolo que se va a describir requiere de la existencia de una unidad supervisora de reemplazamiento que se encargue de realizar las operaciones de gestión de recursos reseñadas y de gestión del reemplazamiento. Estas operaciones incluyen: comunicarse con el nodo externo encargado de iniciar las operaciones de reemplazamiento y de enviar el código de las unidades nuevas, cargar en memoria y crear estas unidades, borrar la unidad reemplazada y supervisar el correcto funcionamiento del protocolo.

  Es evidente que la ejecución de estas operaciones implica el consumo de un determinado tiempo de cómputo. De acuerdo con el método de planificación basado en prioridades, el análisis de planificabilidad se realiza a partir de los períodos y tiempos de cómputo máximos de las tareas del sistema. Por tanto, es imprescindible conocer en todo momento por cuenta de qué tarea se ejecutan las operaciones de reemplazamiento e integrarlas en el análisis.

  Como consecuencia, la supervisora se modela como una tarea esporádica, a la que se le asigna una prioridad y un tiempo de cómputo acorde a las características del sistema. De esta forma, se puede incluir su carga en el análisis de planificabilidad y se puede garantizar que no interferirá el cumplimiento de las restricciones temporales de otras unidades. Las operaciones que impliquen un tiempo de cómputo mayor que el asignado, se deberán dividir y realizarse en activaciones sucesivas

Fases del protocolo de reemplazamiento dinámico.

A continuación se muestra la secuencia de operaciones del protocolo de reemplazamiento. Es importante hacer énfasis en que el tiempo de cómputo de cada una de las operaciones se debe tener en cuenta en el análisis de planificabilidad, para garantizar los plazos de

respuesta del conjunto de tareas. Por consiguiente hay que saber con certeza a qué tareas se contabilizan estos tiempo. Las fases del protocolo están delimitadas por cambios en las tareas a las que se contabiliza el tiempo de cómputo consumido en las operaciones de reemplazamiento.

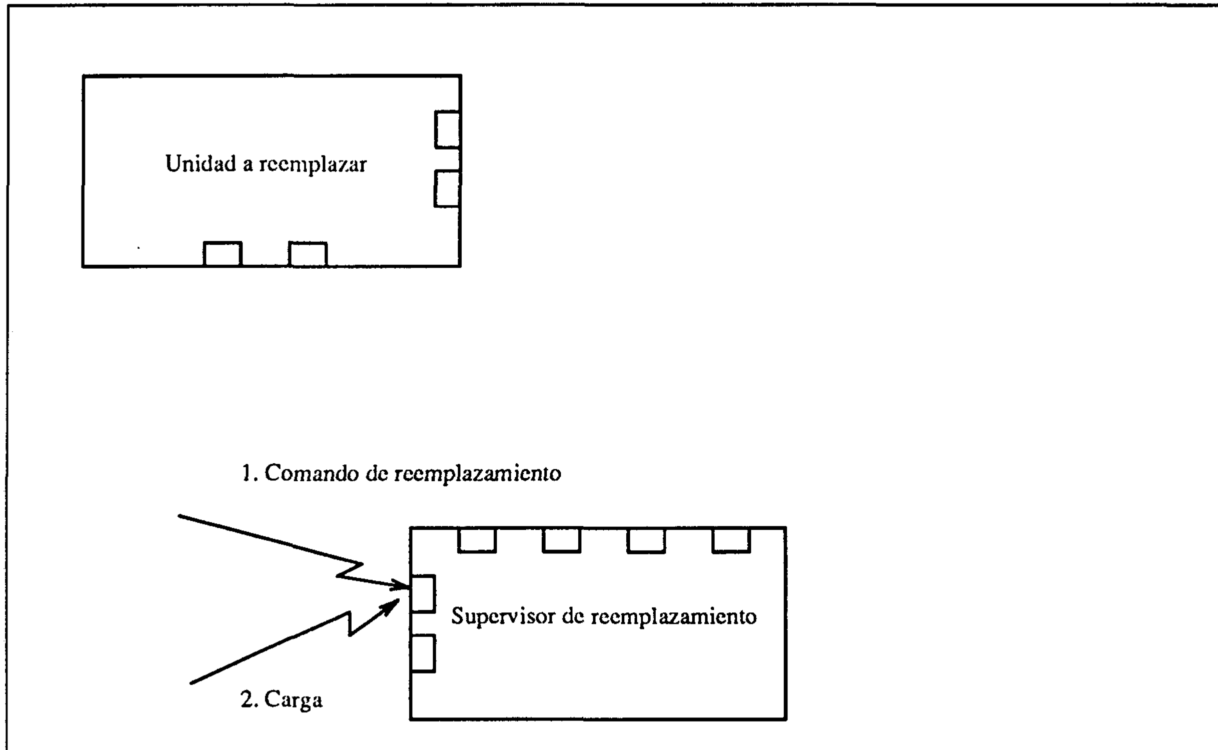

Figura 7.1: Fase de carga del protocolo de reemplazamiento en sistemas de tiempo real.

- *Fase de Carga.*

  1. La unidad supervisora, que estaba bloqueado, recibe una orden externa de reemplazamiento.
  2. La supervisora solicita espacio al núcleo y comienza a cargar la unidad reemplazadora. La carga se realiza en rodajas de tiempo de duración menor o igual al tiempo de cómputo asignado a la supervisora.

- *Fase de Creación.*

  3. La supervisora crea la unidad reemplazadora.

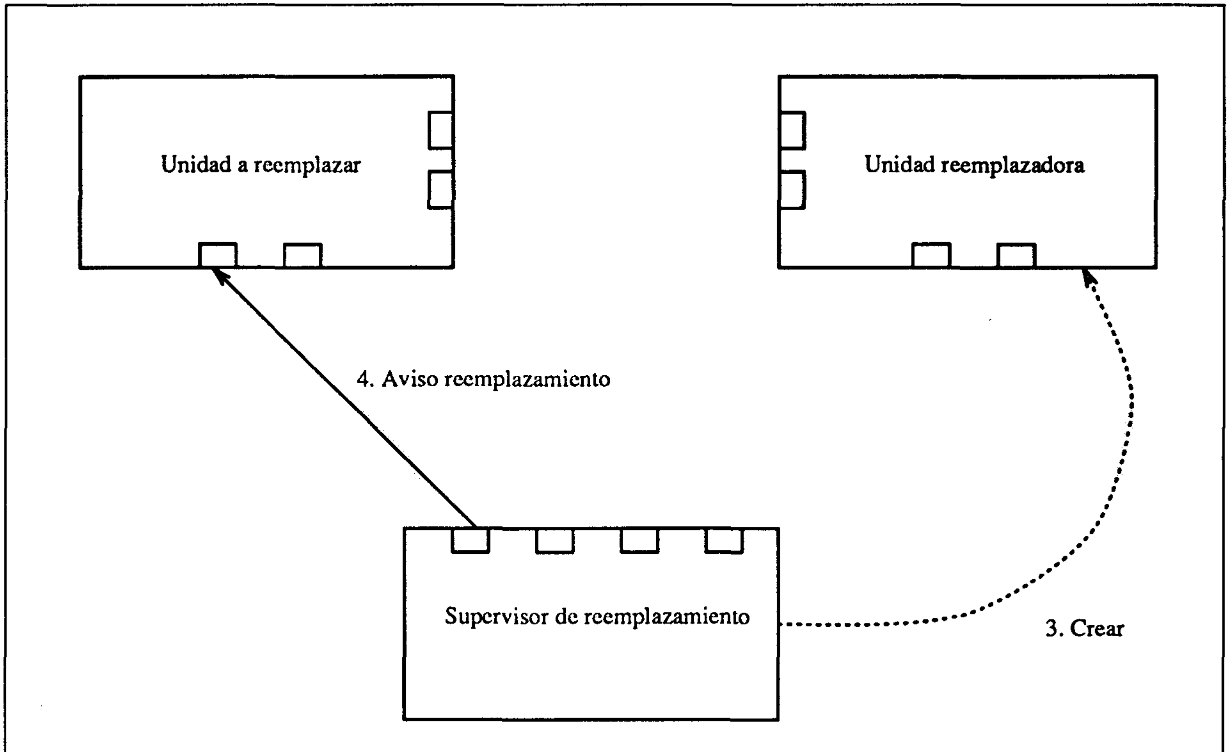

Figura 7.2: Fase de inicio del protocolo de reemplazamiento en sistemas de tiempo real.

4. Se ejecuta el código inicial de la unidad reemplazadora, con la prioridad de la supervisora del sistema.
5. La unidad reemplazadora queda en espera de recibir la información de estado. Antes de bloquearse, cambia su prioridad al valor de la unidad a reemplazar.
6. La supervisora notifica a la unidad a reemplazar, para que ejecute el protocolo de reemplazamiento.

- *Fase de Intercambio.*

7. La unidad a reemplazar manda información de estado a la unidad reemplazadora.
8. La unidad a reemplazar comunica a la supervisora de reemplazamiento información sobre el resultado del intercambio.
9. La unidad reemplazadora comienza a ejecutar a partir de la siguiente activación de la unidad reemplazada más una fase de inicio, si el estado se recibe

satisfactoriamente. La unidad reemplazadora se ejecuta con la misma prioridad que la reemplazada y emplea tiempo de cómputo incluido en el tiempo de cómputo de la segunda.

10. Si la operación se realiza sin errores, la unidad reemplazadora termina. En caso contrario, se prepara para ejecutar la siguiente activación.

- *Fase de Limpieza.*

11. La unidad supervisora notifica al centro externo el resultado de la operación. Si la operación no ha sido correcta, se aborta a la unidad reemplazadora.
12. La supervisora libera la memoria ocupada por la unidad.
13. La supervisora queda en disposición de recibir otra orden de reemplazamiento.

En las figuras 7.1, 7.2, 7.3 y 7.4 se muestra la comunicación entre las unidades en este protocolo.

Propiedades del protocolo de reemplazamiento.

La propiedad más importante del protocolo de reemplazamiento se caracteriza por el siguiente teorema:

Teorema 7.1 *Sean* $\mathcal{P} = \{\tau_1, \tau_2, \ldots, \tau_i, \ldots, \tau_n, Su\}$ *y* $\mathcal{P}' = \{\tau_1, \tau_2, \ldots, \tau'_i, \ldots, \tau_n, Su\}$ *dos conjuntos de tareas.* $\mathcal{P}'$ *se obtiene reemplazando en* $\mathcal{P}$*, la tarea* τ_i *por* τ'_i*. Los conjuntos* $\mathcal{P}$ *y* $\mathcal{P}'$ *son planificables.*

Si el reemplazamiento dinámico de τ_i *por* τ_i' *se realiza según el protocolo de reemplazamiento dinámico, entonces el conjunto de tareas cumplirá sus plazos de respuesta durante y después de una operación de reemplazamiento.*

La prueba de este teorema, se realiza comprobando que las tareas cumplen sus plazos de respuesta en las diversas fases del protocolo.

- *Fase de Carga.* La supervisora de reemplazamiento es la única tarea que ejecuta operaciones de reemplazamiento durante esta fase. La supervisora de reemplazamiento es por naturaleza una tarea aperiódica y se implementa según el método del servidor esporádico. Esta tarea forma parte forma parte del conjunto $\mathcal{P}$ que es planificable. En consecuencia el sistema será planificable durante esta fase.

- *Fase de Creación.* Una vez cargada y creada, la tarea reemplazadora, τ'_i ejecuta las operaciones de inicio con la prioridad de la supervisora y utilizando parte del tiempo de cómputo asignado a ésta. Por tanto, durante esta fase, se sustituye lógicamente la supervisora original, Su, por dos tareas Su' y τ'_i, con los siguientes perfiles de ejecución:

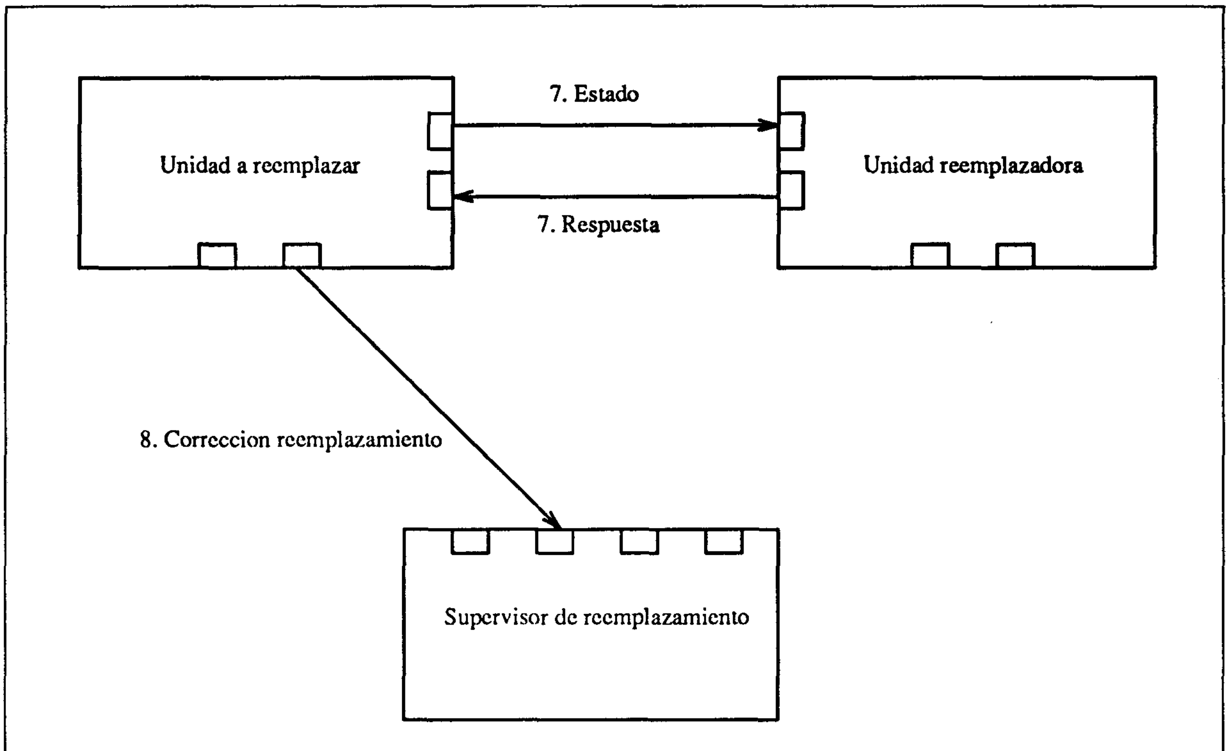

Figura 7.3: Fase de intercambio del protocolo de reemplazamiento en sistemas de tiempo real.

$S\{c_{S_u}, p_{S_u}, pr_{S_u}\}$ se sustituye por
$S'\{c_{S_u} - c_{rT'_i}, p_{S_u}, pr_{S_u}\}$ y $\tau'_i\{c_{rT'_i}, p_{S_u}, pr_{S_u}\}$,
donde $c_{rT'_i}$ es el tiempo de cómputo de τ'_i mientras realiza las operaciones de inicio.

Sprunt y otros [Sprunt&89] han demostrado que, desde el punto de vista del análisis de planificabilidad, el conjunto resultante de realizar la sustitución anterior es equivalente al original. Por consiguiente, las tareas de tiempo real cumplirán sus plazos de respuesta durante esta fase.

- *Fase de Intercambio.* El intercambio del estado implica la ejecución de τ_i y τ'_i. Según el protocolo ambos tiempos se consideran incluidos en el tiempo de cómputo asignado a τ_i y como se realizan con la misma prioridad, el conjunto seguirá siendo planificable. Con respecto al momento en que τ_i deja de ejecutar y comienza τ_i', es equivalente a un cambio de modo de sistema en que sólo cambian estas dos tareas. Para garantizar la planificabilidad del conjunto es necesario determinar, mediante el método incluido en el apéndice, A una fase de inicio, a partir del siguiente instante

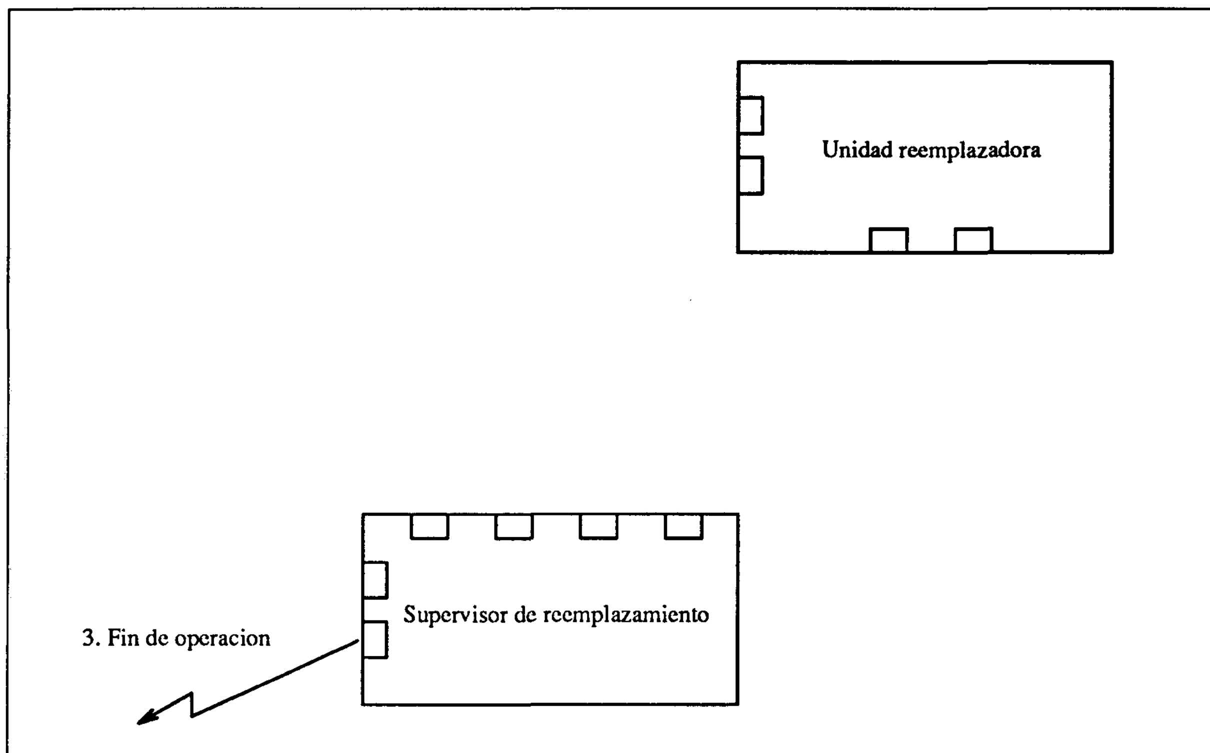

Figura 7.4: Fase de limpieza del protocolo de reemplazamiento en sistemas de tiempo real.

de activación, suficiente.

- *Fase de limpieza.* En esta fase ejecutan la supervisora, Su, y una de las tareas τ_i y τ_i', dependiendo si el reemplazamiento se ha realizado sin errores. En cualquier caso, el conjunto tareas en ejecución será planificable.

El protocolo de reemplazamiento enunciado satisface los restantes requisitos enumerados en la sección 7.2.1:

- La transparencia de las operaciones se garantiza por los mecanismos indirectos de comunicación y por la ejecución, en cualquier circunstancia, de las tareas de acuerdo con su perfil.

- La consistencia del componente se mantiene al realizar las operaciones de reemplazamiento dinámico al final de una activación.

- El estado de ejecución de la unidad a reemplazar se transmite a la reemplazadora. Así se asegura que la ésta comenzará a ejecutarse a partir del estado anterior.
- El protocolo asegura que sólo una de las unidades se ejecutará en la activación siguiente al reemplazamiento. Si el intercambio del estado se ha realizado adecuadamente, se ejecutará la unidad nueva. En caso contrario lo hará la antigua.

Comparación de los protocolos.

El protocolo de reemplazamiento en sistemas de tiempo real presenta las siguientes ventajas:

1. *Aplicable en sistemas empotrados.* El protocolo contempla la problemática específica de los sistemas empotrados y determina la funcionalidad mínima del núcleo de ejecución para realizar reemplazamiento.
2. *Integrado con el método de planificación.* Como se ha demostrado en el anterior teorema, el protocolo se puede emplear en el desarrollo de sistemas de tiempo real crítico basados en prioridades.
3. *Es más sencillo.* El protocolo de reemplazamiento en sistemas de tiempo real es más sencillo. Las simplificaciones respecto al protocolo en sistemas de propósito general son las siguientes:
   - La unidad a reemplazar determina si se ha realizado o no el reemplazamiento. Inmediatamente después de enviar el estado a la unidad reemplazadora, decide si debe continuar ejecutando o no.
   - La unidad reemplazadora ejecuta el protocolo de reemplazamiento, después de ser creada, sin necesidad de esperar una directiva de la unidad de reemplazamiento.
   - La comunicación entre las unidades involucradas en el reemplazamiento es más sencilla. En concreto se han eliminado mensajes de confirmación de la realización de operaciones a la supervisora.
4. *Protocolo determinista.* Los instantes en que se deben realizar las operaciones de reemplazamiento y tareas a las que se contabiliza el tiempo de cómputo empleado están claramente identifcados.

7.5 Realización práctica del protocolo.

El propósito de esta sección es desarrollar una serie de directrices para la realización práctica del protocolo de reemplazamiento dinámico. El resultado final es un esquema de realización de la unidad reemplazable de software, de la unidad supervisora y de una serie de componentes auxiliares. El código resultante no será diréctamente ejecutable. Esto

se debe a que no se conoce un sistema operativo o lenguaje de programación de tiempo real normalizado y suficientemente extendido que proporcione la funcionalidad requerida para la realización práctica del protocolo.

Con objeto de ilustrar el problema, se analizan las características de Ada 9X [Ada9XRM93] y POSIX [POSIX.1b 93] [POSIX.1c 93] en relación a la funcionalidad que se requiere:

- POSIX: La norma POSIX para desarrollo de sistemas de tiempo real permite la creación dinámica de tareas. La información básica que hay que proporcionar es la dirección de inicio del código. Por consiguiente, es posible crear la unidad reemplazadora una vez que se ha cargado en memoria.

  Por el contrario, POSIX no proporciona un modelo normalizado de gestión de memoria. Se presupone que la gestión de memoria dinámica se realiza empleando los mecanismos propios del lenguaje. Este enfoque amplia el conjunto de arquitecturas que pueden proporcionar el interfaz de POSIX. Sin embargo, impide realizar la gestión de memoria preconizada por el protocolo de reemplazamiento empleando los mecanismos proporcionados por la norma.

- Ada 9X: Este lenguaje ofrece aún menos facilidades para la realización del protocolo de reemplazamiento. El modelo de tareas permite crear tareas dinámicamente, pero estás tienen que estar incluidas en la aplicación en tiempo de compilación.

  Una alternativa posible es desarrollar la aplicación empleando los mecanismos que proporcionra el anexo de distribución de Ada. Según este enfoque, una unidad reemplazable tomaría la forma de un programa Ada. La comunicación entre estos programas Ada se realizaría con los mecanismos incluidos en el anexo. En esta situación, existe una forma de enlace dinámico de código (*dynamic binding*) que se puede emplear como base del reemplazamiento dinámico.

  Sin embargo, esta opción presenta algunos inconvenientes. Parece razonable pensar que las prestaciones de una aplicación realizada según este esquema serán bastante peores que empleando el esquema tradicional de concurrencia de Ada. La ejecución de varios programas en la misma máquina será más lenta que si fueran tareas dentro de una aplicación única. El contexto asociado a los programas debe ser mayor y operaciones como cambio de contexto, creación y borrado de unidades serán más costosas. Por otro lado, el último borrador de Ada 9X no describe cómo ha de ser la planificación de tareas que pertenecen diferentes programas Ada y que se ejecutan en el mismo procesador, ni hace referencia alguna a la gestión de la memoria en este caso.

El interés en desarrollar el protocolo con herramientas normalizadas ha motivado que no se hayan estudiado sistemas operativos o lenguajes de programación alternativos. El enfoque que se ha decidido seguir es:

- El protocolo se va a codificar en Ada 9X, aunque el resultado no es diréctamente ejecutable, pues se emplearán algunas funciones que no están permitidas por el lenguaje, pero que son necesarias para el desarrollo del protocolo. La claridad del

lenguaje y la transportabilidad de sus conceptos facilitaría la posible realización del protocolo en otro entorno de ejecución.

- Se proporciona un paquete en el que se incluyen aquellas funciones que debería proporcionar el sistema operativo y cuya funcionalidad no está incluida o permitida por el lenguaje. Es necesario que el entorno de ejecución del protocolo permita la realización de estas funciones y con las características precisas que se expondrán posteriormente.

En la realización práctica de las unidades reemplazables de software (tareas de tiempo real), se ha eliminado la funcionalidad relativa al cambio de modo y al tratamiento de fallos, con objeto de simplificar el código y centrar los conceptos de la realización práctica del protocolo. Las suposiciones de base son:

- Sólo se va a contemplar el reemplazamiento de unidades que no han fallado. Se supone que las unidades del sistema están siempre activas cuando se las quiere reemplazar.
- El sistema no tiene varios modos de funcionamiento.
- El cambio de una unidad no supone cambio en el perfil de ejecución del resto.

En las siguientes secciones se expone una forma de realizar prácticamente el procolo. Inicialmente se definen los mecanismos de comunicación entre las unidades involucradas. Posteriormente se presentan la unidad supervisora y la unidad reemplazable.

7.5.1 Comunicación entre unidades.

El objetivo de este apartado es describir los mecanismos de la comunicación, relativa al protocolo de reemplazamiento, entre las unidades implicadas. Como ya se ha comentado, las colas de mensajes constituyen un medio suficientemente conocido y con las características requeridas por el protocolo, ya que no se basan en mecanismos de memoria compartida y constituyen un método de comunicación indirecto.

En cuanto a las colas de mensahes a emplear, se han preferido diseñadar un conjunto de tipos específicos orientadas a la ejecución del protocolo, en lugar de emplear un modelo general y único. De esta forma se simplifica la codificación de la unidad reemplazable y de la supervisora. Las colas son estáticas y sólo tienen capacidad para almacenar un mensaje. Esto se debe a que no es posible ejecutar operaciones de reemplazamiento simultáneas. Cuando una operación comienza, la anterior habrá finalizado por completo y consecuentemente las colas de mensajes estarán vacias. En la figura 7.5 se muestra as colas implicadas en la comunicación entre las diversas unidades.

Las colas se declaran en el paquete `Comunicación_Reemplazamiento`, según se muestra a continuación. Las colas se han diseñado como objetos protegidos, porque serán accedidas concurrentemente. No se ha incluido el cuerpo por su sencillez.

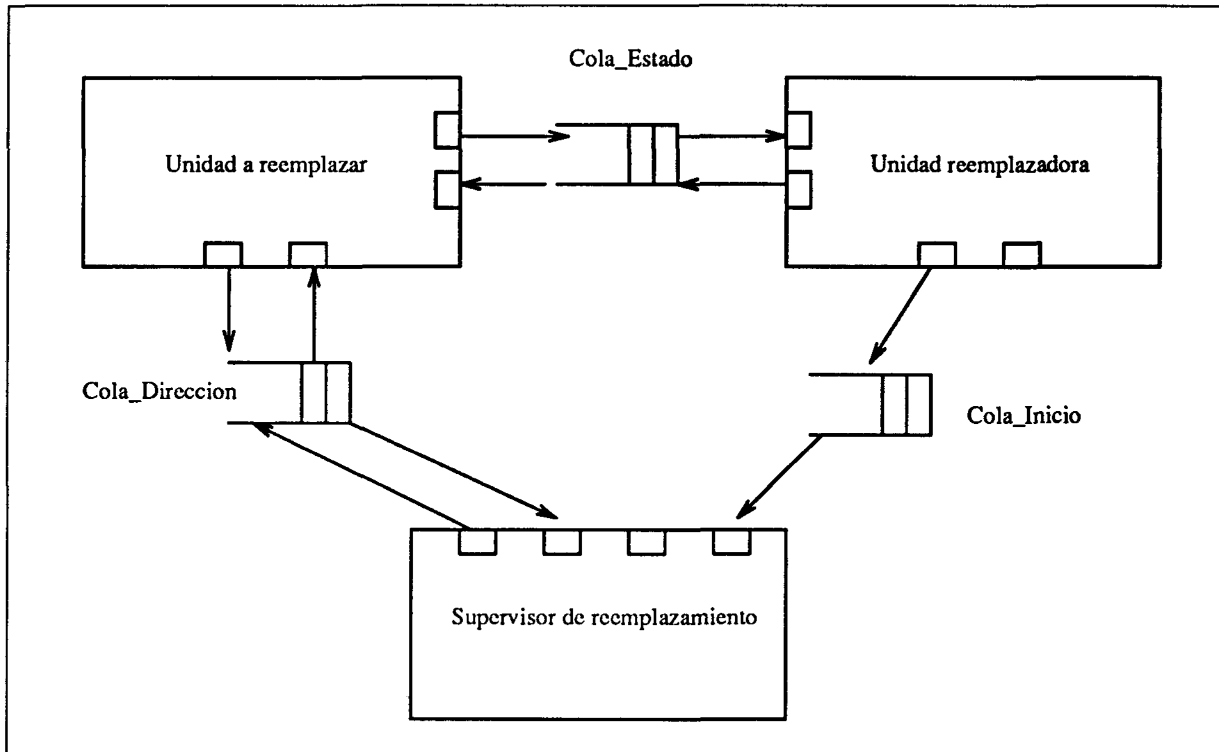

Figura 7.5: Esquema de las comunicaciones entre unidades durante un operación de reemplazamiento.

```
with System.Task_Identification;

package Comunicación_Reemplazamiento is

   type Tipo_Estado is ...  ;

   protected Cola_Estado is
      entry Recibir_Estado(Estado : out Tipo_Estado);
      procedure Mandar_Estado(Estado: in Tipo_Estado; Op_Correcta : Boolean);
   private
      Estado : Tipo_Estado;
      Estado_Recibido : Boolean := False;
   end Cola_Estado;

   type Tipo_Mensaje_C_Dirección is (Nulo, Reemplazar, Op_Correcta, Op_Errónea);

   protected Cola_Dirección is
```

```
      procedure Recibir(Receptor :  in System.Task_Identification.Task_Id;
                        Un_Mensaje :  out Tipo_Mensaje_C_Dirección);
      procedure Enviar(Receptor :  in System.Task_Identification.Task_Id;
                        Un_Mensaje :  in Tipo_Mensaje_C_Dirección);
   private
      Receptor :  System.Task_Identification.Task_Id;
      Mensaje :  Tipo_Mensaje_C_Dirección;
   end Cola_Dirección;

   protected Cola_Inicio is
      entry Recibir;
      procedure Enviar;
   private
      Recepción_Envío :  Boolean := False;
   end Cola_Inicio;

end Comunicación_Reemplazamiento;
```

La funcionalidad de las colas de mensajes es la siguiente:

- `Cola_Estado`: Esta cola se emplea para el intercambio del estado entre la unidad a reemplazar y la reemplazadora. Su funcionalidad está orientada a simplificar la realización del protocolo. En efecto, se supone que cuando la unidad a reemplazar manda la información de estado, la unidad reemplazadora debe estar lista para recibirla. La realización de la cola se basa en este principio, de forma que si cuando la unidad a reemplazar manda el estado la unidad reemplazadora no está lista para recibirla, se supone que hay un error de funcionamiento y se le comunica a la unidad a reemplazar. Como consecuencia, se aborta la operación de reemplazamiento.

  La información de estado que se incluye en el mensaje no puede ser de un tipo concreto. Cada unidad reemplazable del sistema tendrá información de estado diferente. Como es interesante mantener una sóla cola de mensajes para todas las unidades reemplazadoras, se definirá un tipo común, cuyo interés principal es determinar el tamaño máximo que debe tener la información de estado. Para adaptar este tipo a cada las características del estado de cada unidad, se emplean las operaciones de conversión de tipos sin chequeo que proporciona el lenguaje. Las unidades deben disponer de las funciones necesarias para convertir del tipo del estado individual a la información del estado del mensaje y viceversa.

- `Cola_Dirección`: Esta cola de mensajes permite comunicar a la unidad supervisora con las unidades reemplazables, de forma que la primera indica a una unidad su reemplazamiento y ésta contesta con el resultado de la operación (i.e., si se ha realizado corréctamente). El direccionamiento de los mensajes se realiza mediante la indicación explícita del receptor. El procedimiento de envío incluye un campo en el que se especifica la dirección del receptor y en el de recepción se debe indicar la identidad de la tarea receptora para comprobar si hay mensajes para ella. Este esquema permite emplear una única cola de mensajes para intercambiar información entre las unidades reemplazables y la supervisora.

El contenido del mensaje está definido por el tipo enumerado `Tipo_Mensaje_C_Dirección` y el significado de los valores posibles es:

- `Reemplazar`: La supervisora envía este mensaje para indicar a una unidad su reemplazamiento.
- `Op_Correcta`: La unidad a reemplazar indica a la supervisora que ha sido reemplazada satisfactoriamente.
- `Op_Errónea`: La unidad a reemplazar indica a la supervisora que la operación de reemplazamiento ha fallado.

Para especificar la dirección de los mensajes se emplea el tipo `Task_Id`, según se define en el anexo de programación de sistemas de Ada 9X [Ada9XRM93].

- `Cola_Inicio`: Esta cola se emplea para que la unidad reemplazadora indique a la supervisora que ya ha completado las operaciones de inicio y que queda lista para recibir la información de estado. La operación de recepción es bloqueante.

7.5.2 Unidad supervisora del reemplazamiento.

La realización de la unidad supervisora ha implicado el desarrollo de dos paquetes auxiliares. Uno de ellos incluye las operaciones que debería proporcionar el núcleo de ejecución. El otro proporciona las primitivas de comunicación con el nodo externo que se encarga de iniciar las operaciones de reemplazamiento y de proporcionar las unidades reemplazadoras. A continuación se presentan estos dos paquetes y la unidad supervisora.

Apoyo del sistema operativo.

En este apartado se describen las funciones necesarias para la realización del protocolo que no proporciona el núcleo del sistema operativo. Es importante recordar que algunas de estas funciones deben ejecutarse con la prioridad de la tarea que realizó la llamada. Este es el caso cuando la operación solicitada consume una cantidad de tiempo de cómputo significativa, como se indicará en su momento.

El apoyo que debería proporcionar el núcleo de ejecución se encapsula en el paquete `Apoyo_SO`, cuya especificación se muestra a continuación:

```
with Comunicación_Externa;
with System.Task_Identification;

package Apoyo_SO is

   function Solicitar_Memoria(Cantidad_Memoria :  in integer)
            return System.Address;

   procedure Liberar_Memoria(ID_Unidad :  in System.Task_Identification.Task_ID);

   procedure Crear_Tarea(Dirección :  in System.Address;
```

```
                    ID_Unidad :  out System.Task_Identification.Tas_ID);

   procedure Cargar_En_Memoria (Dirección_Inicial :  in System.Address;
                    El_Código :  in Comunicación_Externa.Tipo_Paquete_Código;
                    Dirección_Final :  out System.Address);

   procedure Compactar_Memoria (Tiempo_Cómputo :  in ; Fin_Operación out :  boolean);

end Apoyo_SO;
```

La descripción funcional de las operaciones que exporta es la siguiente:

- `Solicitar_Memoria`: La unidad supervisora emplea esta función para reservar la memoria necesaria para cargar la unidad reemplazadora. El argumento es la cantidad de memoria que se necesita. La función devuelve la dirección de inicio del bloque de memoria solicitada. En caso de que no esté disponible la memoria solicitada, se devuelve el valor `Null_Address`.

- `Cargar_En_Memoria`: Esta operación permite cargar en memoria un determinado bloque de código en la dirección inicial que se proporciona. El código se incluye en una variable del tipo `Tipo_Paquete_Código`, que es el formato en que se reciben los paquetes del nodo exterior. El procedimiento en cuestión devuelve la dirección siguiente al código cargado.

- `Crear_Tarea`: Esta función crea una tarea a partir de su dirección de inicio. Por simplicidad, no se han añadido parámetro adicionales, aunque en algunos casos podrían ser necesarios. Como resultado de la ejecución se devuelve el identificador de la tarea creado. Si por cualquier caso no se ha podido realizar esta labor satisfactoriamente, entonces se retorna el identificador nulo de tarea `Null_Task_Id`.

- `Liberar_Memoria`: El objetivo de este procedimiento es indicar al núcleo de ejecución que una determinada porción de memoria se ha liberado y no está en uso. Este bloque de memoria se corresponde con el asignado a una unidad reemplazable.

- `Compactar_Memoria`: Esta operación es la más complicada de realizar. Su objetivo es conseguir que la memoria libre del procesador sea contigua. En este contexto se supone que esta operación se realiza moviendo bloques de memoria. Es evidente que esta operación no se puede realizar de forma arbitraria, ya que podría interferir en la ejecución de las tareas. La directriz de diseño referente a la no compartición de memoria entre las unidades debe facilitar esta operación. En este apartado no se profundiza en los mecanismos para realizar esta operación porque son muy dependientes del núcleo de ejecución y, en cierta medida, está alejados del tema central de este trabajo.

  En cualquier caso, sí se puede afirmar que la ejecución de este procedimiento se debe realizar con la prioridad de la tarea que lo solicita, dado que su duración puede ser grande. Además, el tiempo de cómputo que emplea está limitado por el tiempo

de cómputo de que dispone la unidad que lo invoca. Esta restricción puede imponer la necesidad de realizar la compactación de la memoria en activaciones sucesivas de la unidad encargada de comenzar esta operación. El estado de la memoria entre activaciones del procedimiento debe ser consistente, para no influir en la ejecución de las tareas. Es importante enfatizar que se debe tener cuidado en el instante en que se cambian de lugar los bloques de memoria de una unidad activa. Estos dos requisitos son imprescindibles para asegurar la planificabilidad del sistema.

El procedimiento en cuestión dispone de un parámetro de entrada en el que se indica el tiempo de cómputo máximo del que dispone el núcleo de ejecución para ejecutar. Es evidente que en este tiempo es posible que no se hayan completado la compactación. El final de la operación se indica con el argumento de salida. De esta forma, la tarea que gestiona esta operación puede invocar esta operación en las activaciones sucesivas que sean necesarias.

Comunicación con el exterior.

La unidad supervisora necesita comunicarse con el nodo externo en el que se encuentra el sistema que ordena las operaciones de reemplazamiento y que envía el código correspondiente. Para tal fin, se define el paquete `Comunicación_Externa`. Este paquete define el tipo de mensaje que se intercambia entre el citado nodo y la unidad supervisora y una serie de operaciones asociadas. No se detalla la realización concreta de este paquete, pues depende en el método de conexión entre los computadores en comunicación.

```
with System.Tasks_Identification;

package Comunicación_Externa is

   type Tipo_Clase_Mensaje is (Código, Orden_Reemplazamiento, Fin_Código,
                 Fin_Reemplazamiento, Fallo_Reemplazamiento);

   type Tipo_Paquete_Código is private;

   type Tipo_Mensaje_Externo is private;

   procedure Obtener_Mensaje (Un_Mensaje :  out Tipo_Mensaje_Externo);

   function Extraer_Clase_Mensaje (Un_Mensaje :  in Tipo_Mensaje_Externo)
                 return Tipo_Clase_Mensaje;

   procedure Enviar_Mensaje (Clase :  in Tipo_Clase_Mensaje);

   procedure Extraer_Info_Orden (Un_Mensaje :  in Tipo_Mensaje_Externo;
                       Id_Unidad :  out System.Task_Identification.Task_Id;
                       Tamaño_Memoria :  out Integer);

   function Extraer_Paquete_Código(Un_Mensaje :  in Tipo_Mensaje_Externo)
                 return Tipo_Paquete_Código;
```

```
private

   type Tipo_Paquete_Código is ...  ;

   type Tipo_Mensaje_Externo is ...;

end Comunicación_Externa;
```

Los elementos que exporta el paquete `Comunicación_Externa` son:

- `Tipo_Clase_Mensaje`: Este tipo indica las clases de mensajes que se intercambian:
  - `Orden_Reemplazamiento`: El nodo externo indica a la supervisora que se debe comenzar una operación de reemplazamiento. En el mensaje le indica información sobre la unidad que va a ser reemplazada y el tamaño de memoria que se debe reservar.
  - `Código`: El mensaje incluye parte del código de la unidad reemplazadora. Se supone que el código de esta unidad se envía en una serie de paquetes.
  - `Fin_Reemplazamiento`: La supervisora notifica a la unidad externa que la operación de reemplazamiento se ha completado satisfactoriamente.
  - `Fallo_Reemplazamiento`: La supervisora notifica a la unidad externa que se ha producido un fallo en el reemplazamiento.
  - `Fin_Código`: El nodo externo indica el final del envío del código.
- `Obtener_Mensaje`: La unidad supervisora queda a la espera de recibir un mensaje.
- `Extraer_Tipo_Mensaje`: Esta operación permite conocer cúal es la clase de un mensaje.
- `Enviar_Mensaje`: La unidad supervisora envía un mensaje al nodo externo.
- `Extraer_Info_Orden`: Este procedimiento permite extraer la información asociada a una orden.
- `Extraer_Código`: Este procedimiento extrae el código de un mensaje de la clase `Código`.

Unidad supervisora.

La unidad supervisora está compuesta por seis llamadas a procedimientos que se ejecutan en secuencia. Cada uno de éstos forma parte de una fase del protocolo de reemplazamiento, según se indica en su descripción. En el momento que se detecta un fallo, la ejecución del protocolo se aborta. A continuación se comentan estos procedimientos:

- Esperar_Mensaje: La unidad supervisora queda bloqueada a la espera de recibir una orden de reemplazamiento del nodo exterior. Cuando recibe un mensaje comprueba que es del tipo correcto. Si es así, se bloquea hasta su siguiente activación. De esta forma se inicia la carga de código con la capacidad de tiempo de cómputo al completo. Esta operación forma parte de la fase de carga del protocolo.

- Cargar_Codigo: Este procedimiento es el encargado de recibir e instalar el código de la unidad reemplazadora. Inicialmente solicita al núcleo de ejecución la memoria necesaria para la creación de la unidad. Si esta operación no tiene exito, se termina la operación de reemplazamiento. Esta operación completa la fase de carga del protocolo.

  La realización práctica de la carga del código depende en gran medida de las características del medio y del protocolo de comunicación entre la unidad supervisora y el nodo externo. Por un lado la recepción de los mensajes puede implicar un conjunto de operaciones de duración fija o no. Factores de fiabilidad del medio y predictibilidad del protocolo de comunicación influyen en esta propiedad. En caso de que la duración sea fija, entonces se puede programar la recepción de los mensajes mediante un bucle que se ejecuta un número fijo de veces en cada activación. En caso contrario, habría que recibir los mensajes en una activación mientras hubiera capacidad de cómputo disponible. En el momento que esta se consumiera se detendría inmediatamente la ejecución de la supervisora hasta la siguiente activación. Para tal fin se pueden emplear los mecanismos de transferencia asíncrona de control, junto con un temporizador asociado al tiempo de cómputo de las tareas. Esta estructura es la empleada en la sección 6.2.3 para asegurar que las tareas de tiempo real no consumen más tiempo de cómputo que el asignado.

  El código de la unidad se recibe en paquetes de tamaño fijo. Si la recepción es correcta, se procede a cargarlos en la zona de memoria asignada previamente. En cada activación de la supervisora se ha de consumir, como máximo, el tiempo de cómputo asignado. En el código que se presenta, se supone que la comunicación es fiable y que, por tanto, la duración del envío de cada paquete es fijo. En cada activación la unidad supervisora podrá recibir un número máximo de paquetes de datos, representado por N_Máx_Mensajes.

- Crear_Tarea: Este procedimiento se encarga de crear la tarea y de esperar hasta a que ésta ejecute el código de inicio. Los errores que pueden aparecer en esta fase consisten en que la tarea no pueda crearse o que una vez creada no complete las operaciones de inicio y antes de un plazo de tiempo concreto. En ambos casos, la acción que se toma consisten en abortar el reemplazamiento. Esta operación realiza las funciones correspondientes a las fase de creación del protocolo.

- Ordenar_Intercambio_Estado: Una vez que la unidad reemplazadora ha completado satisfactoriamente las operaciones de inicio, La supervisora indica a la unidad a reemplazar, para que se ejecute protocolo, es decir, se debe intercambiar el estado entre las unidades. La supervisora queda a la espera del final de esta

operación. Si transcurrido un plazo de tiempo dado no se produce la notificación del fin de la comunicación, la supervisora aborta la operación de reemplazamiento. Este procedimiento ejecuta las acciones de las fase de intercambio.

- `Liberar_Memoria`: Esta operación libera la memoria asignada a una de la unidades reemplazables implicadas. Si la operación de reemplazamiento ha sido correcta, entonces se libera el espacio de la unidad a reemplazar y en caso contrario, la de la unidad reemplazadora.

  En la realización que se presenta se supone que el tiempo de cómpuo necesario para liberar la memoria es menor que el asignado a la unidad supervisora. En caso contrario esta operación se debería partir y realizarla en varias activaciones sucesivas, de forma análoga a la carga de la unidad reemplazadora. Esta operación forma parte de la fase de limpieza del protocolo.

- `Compactar_Memoria`: Este procedimiento gestiona las operaciones de compactación de memoria. La operación se lleva a cabo mediante llamadas sucesivas al procedimiento homónimo del paquete `Apoyo_SO`, hasta completar satisfactoriamente la labor. Mediante esta operación se completa la ejecución del protocolo de reemplazamiento.

El código de este paquete se presenta a continuación:

```
package Unidad_Supervisora is
end Unidad_Supervisora;

with Apoyo_SO;
with Comunicación_Externa;
with Comunicación_Reemplazamiento;
with Calendar;
with System.Task_Identification;
with CPU_Time_Accounting;
package body Unidad_Supervisora is

   task body Supervisora is

      Período :  Duration := ...  ;
      Prioridad :  System.Priority := ...  ;
      T_Cómputo :  CPU_Time_Accounting.Accounting_Range:= ...  ;
      Dirección_Unidad :  System.Address;
      Máx_Espera :  Duration;
      Id_U_Reemplazadora,
      Id_U_a_Reemplazar :  System.Task_Identification.Task_Id;
      Id_Supervisor :  constant System.Task_Identification.Task_Id
                    := System.Task_Identification.Current_Task;
      Tamaño_Memoria :  Integer;
      Siguiente_Activación :  Calendar.Time;
      Correcto :  Boolean := True;

      --************************************************************
```

```
procedure Esperar_Mensaje(Id_Unidad : out
          System.Task_Identification.Task_Id;
          Tamaño : out integer) is

   Mensaje_Externo : Comunicación_Externa.Tipo_Mensaje_Externo;

begin

   loop
      Comunicación_Externa.Obtener_Mensaje(Mensaje_Externo);
      exit when Comunicación_Externa.Extraer_Clase_Mensaje(Mensaje_Externo)
            = Comunicación_Externa.Orden_Reemplazamiento;
   end loop;

   Comunicación_Externa.Extraer_Info_Orden(Mensaje_Externo, Id_Unidad ,Tamaño);

   Siguiente_Activación := Siguiente_Activación + Período;
   delay until Siguiente_Activación;
end Esperar_Mensaje;

--*********************************************************

procedure Cargar_Código(Tamaño_Memoria : in Integer;
                  Dirección_Unidad : out System.Address;
                  Correcto : out Boolean) is

   Mensaje_Externo : Comunicación_Externa.Tipo_Mensaje_Externo;
   N_Máx_Mensajes : constant Integer := ... ;
   Dirección_Carga,
   Dirección_Final : System.Address;
   Fin_Transmisión : Boolean := False;

begin
   Correcto := True;
   Dirección_Unidad := Apoyo_SO.Solicitar_Memoria(Tamaño_Memoria);
   if Dirección_Unidad /= System.Null_Address then
      Dirección_Carga := Dirección_Unidad;
      while Correcto and not Fin_Transmisión loop
         for I in 1..N_Máx_Mensajes loop
            Comunicación_Externa.Obtener_Mensaje(Mensaje_Externo);
            case Comunicación_Externa.Extraer_Clase_Mensaje(Mensaje_Externo) is
               when Comunicación_Externa.Código =>
                  Apoyo_SO.Cargar_En_Memoria(Dirección_Carga,
                  Comunicación_Externa.Extraer_Paquete_Código(Mensaje_Externo),
                  Dirección_Final);
                  Dirección_Carga:=Dirección_Final;
               when Comunicación_Externa.Fin_Código =>
                  Fin_Transmisión := True;
                  exit;
               when others =>
                  Correcto := False;
                  Comunicación_Externa.Enviar_Mensaje
```

```
                (Comunicación_Externa.Fallo_Reemplazamiento);
             exit;
          end case;
       end loop;
       Siguiente_Activación := Siguiente_Activación + Período;
       delay until Siguiente_Activación;
    end loop;
  end if;
end Cargar_Código;

--*************************************************************

procedure Crear_Tarea(Dirección_Unidad :  in System.Address;
       Id_Unidad :  out System.Task_Identification.Task_Id;
       Correcto :  out Boolean) is
begin
  Apoyo_SO.Crear_Tarea(Dirección_Unidad, Id_Unidad);
  if ID_Unidad /= System.Task_Identification.Null_Task_ID
    then
      select
        delay Máx_Espera;
        Correcto:=False;
        Comunicación_Externa.Enviar_Mensaje
              (Comunicación_Externa.Fallo_Reemplazamiento);
      then abort
        Comunicación_Reemplazamiento.Cola_Inicio.Recibir;
      end select;
    else
      Comunicación_Externa.Enviar_Mensaje
              (Comunicación_Externa.Fallo_Reemplazamiento);
  end if;
  Siguiente_Activación := Siguiente_Activación + Período;
  delay until Siguiente_Activación;
end Crear_Tarea;

--*************************************************************

procedure Esperar_Fin_Intercambio
              (Id_Unidad :  in System.Task_Identification.Task_Id;
               Correcto :  out Boolean) is

    Mensaje :  Comunicación_Reemplazamiento.Tipo_Mensaje_C_Dirección;

begin

  Comunicación_Reemplazamiento.Cola_Dirección.Enviar
              (Id_Unidad, Comunicación_Reemplazamiento.Reemplazar);
  select
    delay Máx_Espera;
    Correcto:=False;
    Comunicación_Externa.Enviar_Mensaje
              (Comunicación_Externa.Fallo_Reemplazamiento);
  then abort
```

```
        Comunicación_Reemplazamiento.Cola_Dirección.Recibir
                     (Id_Supervisor, Mensaje);
        if Mensaje = Comunicación_Reemplazamiento.Op_Errónea then
           Correcto:=False;
           Comunicación_Externa.Enviar_Mensaje
                     (Comunicación_Externa.Fallo_Reemplazamiento);
        end if;
     end select;

     Siguiente_Activación := Siguiente_Activación + Período;
     delay until Siguiente_Activación;

  end Esperar_Fin_Intercambio;

--*************************************************************

  procedure Liberar_Memoria (Id_Unidad :  in System.Task_Identification.Task_Id) is
  begin
     Apoyo_SO.Liberar_Memoria(Id_Unidad);
     Siguiente_Activación := Siguiente_Activación + Período;
     delay until Siguiente_Activación;
  end Liberar_Memoria;

--*************************************************************

  procedure Compactar_Memoria is
     Fin :  Boolean := False;
  begin
     while not Fin loop
        Apoyo_SO.Compactar_Memoria(T_Cómputo, Fin);
        Siguiente_Activación := Siguiente_Activación + Periódo;
        delay until Siguiente_Activación;
     end loop;
  end Compactar_Memoria;

--*************************************************************

  begin
     loop
        Esperar_Mensaje(Id_U_a_Reemplazar, Tamaño_Memoria);
        Cargar_Código(Tamaño_Memoria, Dirección_Unidad, Correcto);
        if Correcto then
           Crear_Tarea(Dirección_Unidad, Id_U_Reemplazadora, Correcto);
        end if;
        if Correcto then
           Esperar_Fin_Intercambio(Id_U_a_Reemplazar, Correcto);
        end if;
        if Correcto then
           Liberar_Memoria(Id_U_a_Reemplazar);
           Compactar_Memoria;
        else
           Liberar_Memoria(Id_U_Reemplazadora);
        end if;
```

```
        end loop;

    end Supervisora;

end Unidad_Supervisora;
```

7.5.3 Unidad reemplazable de software.

La realización que se va a presentar en este apartado está de acuerdo con la idea de considerar a la unidad reemplazable como una tarea de tiempo real. En el presente apartado se mostrará la realización de una tarea periódica. En el código no se va a incluir la funcionalidad relativa al cambio de modo ni al tratamiento de fallos, pues ya se ha ilustrado suficientemente y podría complicar la compresensión del código.

La integración del código de reemplazamiento en una tarea esporádica es inmediata. Únicamente hay que tener en cuenta que estas tareas pueden permanecer largos períodos de tiempo ociosa, en ausencia de las condiciones de activación de la tarea. Por esta razón la unidad supervisora no es compatible con el esquema de tarea esporádica presentado anteriormente. Una alternativa sería incluir un punto de entrada adicional a la tarea supervisora para activar la ejecución del protocolo de reemplazamiento.

La unidad reemplazable de software debe contener el código necesario para reemplazar a otra y, a su vez, ser reemplazada. Esta funcionalidad se incluirá en el esquema de tarea periódica presentado en el apartado 6.2. Las definiciones de tipos básicos son las que aparecen en el paquete `Global` que se expone en el citado anterior. A continuación se muestra este código y se comenta la funcionalidad de la tarea en referencia al protocolo.

El código del paquete donde se declara la unidad es análogo al denominado `Tarea_Periódica` en el apartado 6.2. Se han añadido declaraciones de tipos variables y funciones relacionadas con el protocolo de reemplazamiento. La mayoría de ellas están relacionadas con el almacenamiento y la transmisión del estado de la unidad. En primer lugar hay que tener en cuenta que la información de estado puede variar, de forma que el estado que recibe una unidad de la que reemplaza puede ser diferente del suyo propio. Por esta razón, se incluye la definición de los tipos `Tipo_Estado_Previo`, que es el tipo de la información de estado de la unidad a la que reemplaza, y `Tipo_Estado_Actual` que es la información de estado de la propia unidad.

Un problema adicional es que el tipo `T_Estado` que proporciona la cola de mensajes `Cola_Estado` es un tipo fijo e independiente del estado específico de las unidades. Para resolver este problema se emplean los mecanismos que proporciona el lenguaje para la conversión entre tipos sin comprobaciones. De esta forma, el tipo `T_Estado` únicamente define el tamaño máximo que puede tener la información de estado de las unidades. Se definen funciones para convertir `T_Estado` en `Tipo_Estado_Previo` y `Tipo_Estado_Actual` en `T_Estado`.

```
package Unidad_Reemplazable is
end Unidad_Reemplazable;
```

```
with Global;
with Comunicación_Reemplazamiento;
with Ada.Unchecked_Conversion;
with Ada.Task_Identification;

package body Unidad_Reemplazable is
   Perfil_Tarea : Global.Tipo_Perfil_Ejecucion :=
        (Período=> 0.1, Fase=> 1.0, Prioridad=> 3,
        T_Cómputo=> 0.01, Plazo_Respuesta=> 0.1);

   type Tipo_Estado_Previo is . . .;
   function Conversión_A_Tipo_Previo is new
      Ada.Unchecked_Conversion(Comunicación_Reemplazamiento.T_Estado,
           Tipo_Estado_Previo);

   type Tipo_Estado_Actual is . . .;
   function Conversión_De_Tipo_Actual is new
      Ada.Unchecked_Conversion(Tipo_Estado_Actual,
           Comunicación_Reemplazamiento.T_Estado);

   Estado : Comunicación_Reemplazamiento.T_Estado;
   Máx_Espera : Duration := . . .;
   Período_Reemplazadora : Duration := . . .;
   Prioridad_Reemplazamiento : System.Priority := . . . ;
   Prioridad_Supervisora : System.Priority := . . . ;
   U_Supervisora_Id : Ada.Task_Identification.Task_Id := . . . ;


   task Tarea_Periódica is
   end Tarea_Periódica;
   task body Tarea_Periódica is separate;
   pragma priority (Prioridad_Supervisora);

end Unidad_Reemplazable;
```

El esquema de la tarea periódica no varía. Las operaciones del protocolo de reemplazamiento cuando la unidad reemplaza a otra se han incluido dentro del procedimiento `Operaciones_Iniciales`. Las operaciones del protocolo cuando la unidad es reemplazada se han incluido en el procedimiento `Operaciones_Posteriores`. Esto se debe a que el reemplazamiento se produce al final de una activación. A continuación se describen estos casos;

- Unidad como reemplazadora: El código se ejecuta como parte del procedimiento de inicio. El protocolo establece que la tarea realiza las operaciones de inicio con el perfil de la unidad supervisora. Al completar esta operación, la unidad lo notifica, cambia su prioridad y queda a la espera del estado de la unidad a reemplazar. Para evitar que la espera sea durante un tiempo indefinido, se establece un temporizador. En caso de que expire, se aborta inmediatamente a la unidad.

Una vez recibida la información de estado, se cambia el perfil en consonancia. A partir de este momento se supone que el protocolo se ha completado y la tarea comienza a ejecutar el código referente a su funcionalidad específica.

- Unidad como reemplazada: La primera condición para reemplazar a una URS es que esté en una situación consistente. Como se vió, esto ocurre cuando la tarea termina una activación. Así pues, el lugar más adecuado para incluir este código es en el procedimiento `Operaciones_Posteriores`.

  La tarea comienza la ejecución del protocolo comprobando si va a ser reemplazada. Esto se realiza mediante una petición de información a la cola de mensajes `Cola_Dirección`. Si no hay ningún mensaje para la unidad, se recibe un mensaje nulo y se continúa con la ejecución de la tarea. Si se recibe un mensaje de reemplazamiento, entonces se ejecutan las operaciones de reemplazamiento.

  Entonces, la unidad manda el estado a la cola `Cola_Estado`. Si la operación es correcta, la tarea completa su ejecución y en caso contrario, continúa la ejecución. En cualquiera de los casos comunica el resultado de la operación al supervisor.

```
with Ada.Dynamic_Priorities;

separate (Tarea_Periódica);

   task Tarea_Periódica is

   Estado_Previo :  Tipo_Estado_Previo;
   Estado_Actual :  Tipo_Estado_Actual;
   Transferencia_Correcta :  Boolean;
   Mi_Id :  Ada.Task_Identification.Task_Id :=Ada.Task_Identification.Current_Task;
   Siguiente_Activación :  Calendar.Time;

   - - ****************************************************

   procedure Operaciones_Iniciales is
   begin

      . . .  Conjunto de operaciones de inicio.

      Comunicación_Reemplazamiento.Cola_Inicio.Enviar;
      Ada.Dynamic_Priorities.Set_Priority(Prioridad_Reemplazamiento);

      select
         delay Máx_Espera;
         abort Tarea_Periódica;
      then abort
         Comunicación_Reemplazamiento.Cola_Estado.Recibir_Estado(Estado);
      end select;

      Estado_Previo := Conversión_A_Tipo_Previo(Estado);
      Ada.Dynamic_Priorities.Set_Priority(Perfil_Tarea.Prioridad)
      Siguiente_Activación := Calendar.Clock + Período_Reemplazadora
            + Perfil_Tarea.Fase;
```

```
end Operaciones_Iniciales;

-- ****************************************************
procedure Operaciones_Previas is

begin
   . . .
end Operaciones_Previas;

-- ****************************************************

procedure Actividad_Periódica is
begin
   . . .
end Actividad_Periódica;

-- ****************************************************

procedure Operaciones_Posteriores is

begin
   Comunicación_Reemplazamiento.Cola_Dirección(Mi_Id,Mensaje);
   if Mensaje = Comunicación_Reemplazamiento.Reemplazar
      then
         Comunicación_Reemplazamiento.Cola_Estado.Mandar_Estado
                  (Conversión_De_Tipo_Actual(Estado_Actua),
                  Transferencia_Correcta);
         if Transferencia_Correcta
            then
               Comunicación_Reemplazamiento.Cola_Dirección.Enviar
               (Tarea_Supervisora_Id,Comunicación_Reemplazamiento.Op_Correcta);
            else
               Comunicación_Reemplazamiento.Cola_Dirección.Enviar
               (Tarea_Supervisora_Id,Comunicación_Reemplazamiento.Op_Errónea);
         end if;
   end if;
end Operaciones_Posteriores;

-- ****************************************************

begin -- Tarea_Periódica
   Operaciones_Iniciales;
   loop
      Operaciones_Previas;
      Actividad_Periódica;
      Operaciones_Posteriores;
      if Transferencia_Correcta
         then
            exit;
         else
            Siguiente_Activación := Siguiente_Activación + Perfil_Tarea.Período;
            delay until Siguiente_Activación;
      end if;
```

```
        end loop;

    end Tarea_Periódica;
```

7.6 Ampliaciones.

En el apartado anterior se ha presentado un esquema de realización del protocolo. Para centrar la descripción en los detalles relativos al protocolo, se ha eliminado la funcionalidad relativa al tratamiento y detección de fallos y al cambio de modo. A continuación se proporcionan unas directrices básicas para la ampliación de la unidad de software reemplazable para añadir estas funciones:

- **Reemplazamiento de una unidad errónea.** Si se sigue un modelo en el que la tarea al detectar un fallo acaba, entonces el protocolo se simplifica pues no habrá estado inicial y la podrá comenzar su ejecución inmediatamente después de ser creada.

- **Integración del cambio de modo.** Si se supone que una operación de reemplazamiento no implica cambiar los perfiles de ejecución de las tareas del sistema, entonces es inmediato integrar el cambio de modo con las operaciones de reemplazamiento. Es suficiente con asegurar que cuando se va a realizar un reemplazamiento no se puede lanzar una orden de cambio de modo y viceversa.

  En caso de que no se cumpla la anterior premisa, sería necesario adoptar un sistema diferente de gestión de los perfiles de las tareas del sistema. En el modelo presentado en el apartado 6.3.1, el perfíl de las tareas se gestiona de forma distribuida, ya que cada tarea conoce su perfil, pues está almacenado en una variable interna. En caso de que los perfiles de las tareas cambien como resultado de un reemplazamiento, habría que centralizar la gestión de los perfiles de ejecución. Cuando se produce un cambio de modo, las tareas acceden a este repositorio central de perfiles.

  Si una operación de reemplazamiento implica cambio de perfiles de tareas no implicadas diréctamente, entonces sería necesario actualizar a los datos con los nuevos. La activación de la nueva tarea se sincroniza con una cambio de modo que implique volver a consultar los perfiles de cada tarea.

- **Integración con grupos de recuperación.** Los mecanismos de recuperación de fallos basados en grupo de recuperación se pueden mejorar en combinación con el reemplazamiento dinámico. En efecto, se puede reemplazar a las tareas que han fallado y a continuación restaurar la ejecución del conjunto del grupo. Esto se puede hacer parando a la tarea de recuperación y rearrancando al resto. En este caso sólo la tarea defectuosa se termina y las demás quedan en un estado en espera, del que se sale cuando la tarea de recuperación recibe información de que la tarea fallida ya se ha reemplazado y está lista. Entoces la tarea de recuperación activaría a las de su grupo.

7.7 Conclusiones.

Una características deseable de los sistemas de tiempo real es su capacidad de adaptación dinámica a los cambios que ocurren en su su ciclo de vida. Estos se pueden deber a múltiples razones, como detección de errores, cambios imprevistos de las especificaciones, etc. La transición entre diferentes versiones de un sistema se suele realizar deteniendo su ejecución e instalando la nueva versión. Esto no es siempre posible, pues hay sistemas que no se pueden detener si el riesgo de causar daños importantes. Una solución es desarrollar sistemas compuestos por unidades reemplazables y definir mecanismos de reemplazamiento dinámico. Este problema no se ha tratado en sistemas de tiempo real.

En este apartado se ha presentado un protocolo de reemplazamiento dinámico de software para sistemas de tiempo real críticos basados en prioridades. Este protocolo permite asegurar el cumplimiento de los plazos de respuesta del sistema durante el reemplazamiento. Como parte del problema, se han definido los requisitos del núcleo de ejecución para realizar esta operación.

Finalmente se ha desarrollado un esquema de realización práctica del protocolo. En su diseño se han intentado seguir los mismos principios en relación al uso de lenguajes normalizados. Sin embargo, esto no ha sido completamente posible, dado que no hay un núcleo de ejecución normalizado que proporcione la funcionalidad requerida. El esquema de la unidad reemplazable que se ha desarrollado sigue la estructura de las tareas desarrolladas en el capítulo anterior.

Capítulo 8

Aplicación: Biblioteca de Componentes Ada

Entre los años 1989-1992 se ha desarrollado el proyecto Biblioteca de Componentes Ada, cuyo objetivo básico era estudiar la viabilidad del lenguaje Ada para desarrollar componentes software para su aplicación en el desarrollo de sistemas de tiempo real críticos. Uno de los componentes producidos es un ejecutivo multitarea[1]. El diseño y realización de este componente está basado en el algoritmo de planificación de prioridad al proceso más urgente y en las ampliaciones presentadas en el apartado 4.

Este proyecto marcó el comienzo del trabajo sobre esquemas de ejecución, que ha producido los esquemas presentados en el capítulo anterior. En este contexto se realizaró prácticamente la primera versión de los esquemas realizados con Ada, los cuáles fueron exhaustivamente probados. Las conclusiones extraidas de su análisis han permitido mejorarlos, empleando la funcionalidad adicional que proporciona Ada 9X.

En el transcurso del capítulo se pueden encontrar algunos razonamientos sobre el diseño de los elementos del ejecutivo, que pueden resultar algo reiterativos, respecto a los proporcionados en los capítulos anteriores. Sin embargo, se ha decidido mantenerlos por su valor documental del ejecutivo multitarea y porque su implementación en Ada invalida algunas decisiones de diseño de los esquemas anteriores.

El objetivo de este capítulo es presentar el ejecutivo multitarea, haciendo especial enfasis en sus características para desarrollo de sistemas de tiempo real crítico. En apartado posteriores se describe el componente, se presentan las decisiones de diseño tomadas y se enumeran sus características y restricciones.

[1]En este documento, se emplea el término *tarea* en referencia a la unidad de concurrencia. Por razones de consistencia con los términos empleados en el desarrollo del ejecutivo multitarea, sólo en este capítulo se va a utilizar el término *proceso* en referencia a un objeto concurrente del ejecutivo multitarea.

8.1 Planteamiento

8.1.1 Reutilización de software

El tamaño y complejidad de los sistemas informáticos aumentan contínuamente. Como consecuencia, aumenta su coste y se hace más complicado su gestión y desarrollo. Uno de los problemas que se han detectado a este respecto, es que los proyectos software se suelen comenzar desde cero en cada ocasión.

El panorama en otros campos es radicalmente distinto. Así por ejemplo el diseñador de hardware dispone de un conjunto amplio, bien elegido y muy probado de componentes, de gran utilidad en el desarrollo del sistema en cuestión. Desde hace algún tiempo se está trabajando en la posibilidad de obtener una aproximación similar en el desarrollo de sistemas software, que permita reutilizar componentes software. Sin embargo, hasta hace algún tiempo, los únicos componentes software reutilizados eran librerías de subrutinas relacionadas con aplicaciones específicos.

En el contexto de la ingeniería del software, se puede definir reutilización como [Fernández&90a]:

> Un componente de software es reutilizable cuando puede utilizarse varias veces en el desarrollo de sistemas.

En esta definición se plantea la reutilización de componentes, no de aplicaciones globales. En general los elementos que se reutilizan son subprogramas, objetos o subsistemas que realizan una función específica dentro de un sistema completo. No sólo es posible la reutilización de software. También se pueden reutilizar especificaciones, diseños o documentación.

Debe tenerse en cuenta que a veces es necesario adaptar un componente para poder emplearlo en una aplicación. Sin embargo el coste de adaptación debe ser pequeño en comparación al beneficio obtenido.

La reutilización de software permite ventajas como [Sommerville89] [Syms&91]:

- Reducir costes de desarrollo.
- Aumentar la fiabilidad del sistema.
- Reducir los riesgos del conjunto.
- Aprovechar los conocimientos de especialistas.
- Encapsular normas de programación en componentes reutilizables.
- Reducción del tiempo de desarrollo.

La reutilización de software sistemática, como una técnica general de desarrollo no es una práctica común. Hay diversas razones técnicas y de gestión a esta situación [Syms&91]:

- Coste añadido de desarrollo de software reutilizable
- Difícil cooperación entre cliente y contratista en relación a reutilización de código.
- Algunos ingenieros de software prefieren escribir su código desde cero, porque piensan que pueden mejorar los componentes reutilizables:
- No hay herramientas eficaces para clasificar, catalogar y recuperar componentes software.

8.1.2 Planteamiento general del proyecto

En este entorno se encuentra el enmarcado el proyecto Biblioteca de Componentes Ada (BCA) [Fernández&91][Puente&92][Puente&92a][Fernández&93], cuyo objetivo fue estudiar la viabilidad de Ada para desarrollar componentes software reutilizables de aplicación en sistemas informáticos aeroespaciales.

Dada la dificultad de determinar un conjunto de componentes adecuados para cualquier aplicación, la aproximación más realista en la actualidad se basa en seleccionar componentes susceptibles de reutilización en un dominio de aplicación concreto. Esta labor se lleva a cabo realizando un análisis del dominio en cuestión.

En el proyecto BCA, este análisis de dominio consistió en el estudio de varias aplicaciones de aviónica. Un grupo de los componentes seleccionados estaba relacionados con ejecutivos reutilizables para sistemas de aviónica. En concreto, el análisis de dominio permitió identificar dos ejecutivos de interés:

- En la mayoría de las aplicaciones de aviónica, el software empotrado de tiempo real crítico se desarrolla según un modelo de ejecución cíclico [Baker&89]. Las ventajas de esta aproximación son un esquema de ejecución determinista, sobrecarga al sistema pequeña y gran experiencia en su uso. Como parte del proyecto se desarrolló un ejecutivo cíclico según este esquema [Zamorano&92].
- Una aproximación nueva al desarrollo de sistemas de tiempo real se basa en el método de planificación de prioridad a la tarea más urgente. Sus ventajas son mayor nivel de abstracción, flexibilidad y facilidad de mantenimiento. Se ha desarrollado un ejecutivo multitarea basado en este método de planificación [Alonso&92c][Alonso&92a][Alonso&92b].

8.1.3 Ejecutivo multitarea

El desarrollo del ejecutivo multitarea comenzó con la definición de los requisitos. El punto de partida de este trabajo fue una descripción de los requisitos de ejecutivos reutilizables para sistema de aviónica desarrollado por personal de CASA (Construcciones Aeronaúticas, S.A.) [CASA&91a], con experiencia en este tipo de aplicaciones. La funcionalidad básica requerida se resume en los siguientes puntos:

1. Concurrencia. Se deben permitir crear, activar y abortar procesos.
2. Coexistencia de procesos periódicos, aperiódicos y de segundo plano.
3. Establecimiento de plazos de respuesta de los procesos y comprobación de su cumplimiento.
4. Comunicación entre procesos.
5. Cambio de modo. Se debe permitir definir diversos modos de ejecución del sistema y operaciones para cambiar de modo.
6. Tratamiento de errores. Los errores se deben detectar y tratar inmediatamente y de acuerdo a los mecanismos generales del ejecutivo. Los errores no se deben propagar entre procesos.
7. Caracterización por parte del usuario de rutinas para iniciar el sistema y para detener la ejecución del sistema desde un estado seguro.
8. Comprobación analítica de la planificabilidad del sistema.

El diseño del ejecutivo estuvo influenciado por el desarrollo de la teoría de planificación basada en prioridades y por las características específicas del software reutilizable [Fernández&90a].

8.1.4 Inconvenientes de Ada.

El objetivo de este apartado es plantear los problemas que presenta el lenguaje de programación Ada para la programación de sistemas de tiempo real crítico. Estos problemas, en su mayoría resueltos en el lenguaje Ada 9X, han obligado a desarrollar un arquitectura software y unos esquemas diferentes a los presentados anteriormente.

A continuación se enumeran los problemas más importantes [Burns&87b] [Cornhill&87a] [Alonso&90]:

1. Modelo de concurrencia. Los siguientes problemas están relacionados con el modelo:
   (a) La asignación de prioridades a tareas es opcional. El modelo de concurrencia no concreta el comportamiento de las tareas sin prioridad.
   (b) No está definido el comportamiento del planificador cuando dos tareas tienen la misma prioridad.
   (c) Si en una sentencia de selección múltiple hay varias llamadas pendientes, la elección del punto de entrada es arbitraria.
   (d) La asignación de prioridades a tareas es estática.

   (e) El orden de aceptación de las tareas que llaman a un punto de entrada es FIFO, sin que las prioridades tengan efecto alguno.

   (f) Tratamiento de interrupciones. La cita que se produce como consecuencia de una interrupción se ejecuta con prioridad mayor a cualquier otra en el sistema.

2. Gestión del tiempo. Las facilidades que proporciona Ada para la gestión del tiempo presenta las siguientes restricciones:

   (a) El lenguaje sólo permite establecer retardos relativos.

   (b) No está acotado el plazo máximo en que se reanuda la ejecución de una tarea retardada.

   (c) Se pueden establecer plazos de tiempo respecto al inicio de una cita, pero no respecto a su conclusión.

Finalmente, cabe señalar que se ha desarrollado un importante esfuerzo de investigación para solucionar algunos de estos problemas, que ha dado lugar a documentos [Sha&90][Borger&89] donde se exponen directrices para emplear el lenguaje de forma que se eviten estos problemas. Estas directrices se han seguido, siempre que haya sido posible, en el desarrollo del ejecutivo multitarea [Alonso&92c].

8.2 Descripción general del ejecutivo multitarea

8.2.1 Descripción funcional.

El ejecutivo multitarea es un subsistema reutilizable cuya función es ejecutar un conjunto de procesos concurrentes, de forma que se garanticen los plazos de ejecución de los procesos de tiempo real. En el contexto de este documento, un subsistema es un componente software reutilizable compuesto por un conjunto de (sub)componentes que cooperan [Booch87]. Este tipo de componente se emplea cuando se quiere modelar una abstracción demasiado compleja como para diseñarse mediante un único elemento.

El diseño de los componentes se basa en la teoría de planificacióm presentada anteriormente. A los procesos se les asignarán prioridades estáticamente, de acuerdo a los métodos presentados. El modelo de concurrencia de Ada permite planificar los procesos así creados, siempre que se sigan unas directrices de diseño básicas. Sin embargo, ha sido necesario modificar el núcleo básico de Ada (*run time system*) para satisfacer los requisitos del ejecutivo multitarea y adaptar algunos algoritmos a las características específicas del lenguaje.

A continuación se describen las características principales del ejecutivo multitarea.

Procesos

La abstracción básica del ejecutivo multitarea es el proceso. El sistema estará compuesto por un conjunto de procesos que ejecutan concurrentemente. El ejecutivo proporciona

componentes para crear procesos periódicos, esporádicos, de recuperación y de segundo plano.

Los dos primeros son procesos de tiempo real y tienen las características generales siguientes:

- Ejecutan una actividad proporcionada por el usuario con una periodicidad fija (procesos periódicos) o como respuesta a un evento asíncrono (procesos esporádicos).
- Admite diversos modos de ejecución. Cada uno de ellos se caracteriza por un perfil de ejecución, en el que se determinan datos como el período (el plazo de respuesta de estos procesos coincide con el período), la prioridad y la fase del proceso para procesos periódicos y para procesos esporádicos, el tiempo entre activaciones sucesivas, la prioridad y el plazo de respuesta.
- Detecta cambios en el modo del sistema y cambia el perfil de ejecución en consecuencia.
- Detecta incumplimientos del plazo de respuesta y desbordamiento del espacio de memoria asignado al proceso.
- Cuando detecta un fallo en su ejecución, se lo comunica al proceso de recuperación asociado.
- Detecta fallos en otros procesos del grupo de recuperación.
- En caso de detectar errores en su ejecución o en otros procesos del grupo de recuperación, termina inmediatamente.

Los procesos de recuperación son los encargados de ejecutar las actividades de recuperación de fallos. Su estructura es muy similar a la de un proceso periódico. Se activan cuando se detecta un fallo en un proceso del grupo.

Los procesos de segundo plano son procesos sin restricciones de tiempo. Deben tener menor prioridad que cualquier proceso de tiempo real.

Varios modos de ejecución

El ejecutivo multitarea permite definir los modos de ejecución del sistema y determinar el perfil de ejecución de los procesos para cada uno de ellos. El ejecutivo multitarea proporciona los medios para cambiar el modo del sistema y garantiza que los procesos ejecutan en consecuencia. Los cambios de modos se realizan de forma que se mantiene la consistencia de los datos de los procesos implicados y todos los procesos del sistema cumplen su plazo de ejecución.

Comunicación entre procesos

El ejecutivo multitarea permite la comunicación entre procesos mediante monitores. En el contexto del ejecutivo, un monitor es un servidor que proporciona una serie de servicios que se ejecutan con exclusión mutua. La implementación realizada permite acotar los tiempos de bloqueo por inversión de prioridades.

Recuperación de fallos

A pesar de todas las pruebas del sistema, siempre pueden aparecer errores en ejecución (error de hardware, incumplimiento de un plazo de ejecución, etc.). Así pues, si se quiere desarrollar sistemas robustos, es necesario considerar algún mecanismo de recuperación de fallos. En este ejecutivo se sigue el enfoque del *grupo de recuperación*

Parada segura del sistema

El ejecutivo multitarea permite al usuario proporcionar un procedimiento que se ejecuta en caso de fallos graves del sistema. Este procedimiento debe llevar al sistema a un estado seguro y detener la ejecución. Un caso en que se puede realizar esta función es cuando un proceso de recuperación falla.

Comprobación analítica de la planificabilidad

Una característica del algoritmo de prioridad al más frecuente es que permite comprobar analíticamente la planificabilidad del sistema, es decir, si se pueden planificar los procesos garantizando el cumplimiento de sus plazos de ejecución.

8.2.2 Componentes reusables

Los componentes reutilizables desarrollados se han agrupado de acuerdo a su nivel de abstracción. A continuación se enumeran estos grupos y se comentan brevemente los componentes desarrollados:

1. Componentes de interfaz

   - `Hardware_Interface`: Este componente proporciona una serie de definiciones de tipos básicos cuyo objetivo global es mejorar la transportabilidad de una aplicación. Se incluyen tipos definidos en paquetes estándar en Ada dependientes de implementación y otros tipos fuertemente dependientes del hardware.
   - `Extended_Time`:Este componente constituye una alternativa a la utilización de los tipos `Duration` y `Time` proporcionados por el compilador. Se pretende con ello garantizar un intervalo de definición y una precisión intrínseca independiente de la configuración hardware/software utilizada. Se proporcionan

tipos similares conceptualmente, pero más precisos y operaciones aritméticas y lógicas necesarias para mantener la abstracción.

- `Run_Time_System_Interface`: Este componente debe considerarse como una extensión a la funcionalidad proporcionada por el núcleo básico (*run-time system*) de Ada. Esta extensión incorpora dos utilidades fundamentales:
  - Esperas (*Delays*) en tiempo absoluto.
  - Cambio dinámico de prioridad de las tareas.

2. Estructuras básicas.
Este grupo incluye componentes con los tipos abstractos de datos básicos necesarios para configurar los procesos. Esto tipos incluyen los perfiles de ejecución de los procesos, asociación entre perfiles de ejecución y modos del sistema y asociación de tiempos de retardo de los procesos en efectuar ciertas operaciones, a cada modo del sistema.

   - `Executive_Exceptions`: Este componente define el conjunto de excepciones para tratar los posibles errores producidos al definir las características básicas de los procesos de tiempo real.
   - `Execution_Profiles`: Este componente define el conjunto de tipos abstractos de datos mencionados.

3. Constructores de procesos.
Este grupo incluye los elementos básicos del sistema.

   - `System_Supervisor`: Realiza funciones globales al sistema, como gestión de cambio de modo y ejecuta las rutinas, proporcionadas por el usuario, de inicialización y parada segura.
   - `Periodic_Process`: Este componente implementa el proceso periódico, con la funcionalidad mencionada y permite crear réplicas. Sus parámetros genéricos permiten indicar, entre otros, los perfiles de ejecución, actividad y grupo de recuperación de cada una de ellas.
   - `Sporadic_Server_Process`: Componente análogo al anterior para crear procesos esporádicos.
   - `Recovery_Process`: Componente para creación de procesos de recuperación. Presenta algunas características diferentes a los anteriores, como la determinación del componente en caso de fallo.

4. Esquemas de aplicación.
Los esquemas de aplicación son plantillas cuyo objetivo es auxiliar al usuario en la definición de elementos específicos de la aplicación. Los primeros tres elementos son plantillas en las que se proporciona una estructura para determinar los parámetros genéricos necesarios para crear procesos a partir de constructores de procesos. Los restantes componentes son plantillas donde se incluyen directrices sobre el uso de características del lenguaje para crear los objetos correspondientes con la funcionalidad requerida por el conjunto de la aplicación.

- `Periodic_Process_Shell`
- `Sporadic_Server_Process_Shell`
- `Recovery_Process_Shell`
- `Monitor_Shell`
- `Interrupt_Handler_Shell`
- `Background_Process_Shell`

8.2.3 Estructura de una aplicación

En la figura 8.1 se muestra el esquema de un sistema creado con el ejecutivo multitarea.

En el primer nivel está el computador del sistema. El nivel más elemental de software está compuesto por el núcleo básico de Ada. En la capa más externa se encuentran los elementos creados por el usuario con los componentes del ejecutivo y que se introducen a continuación.

La concepción del ejecutivo multitarea impone que en todo sistema que se desarrolle a partir de sus componentes, debe existir un supervisor. Este es el proceso más prioritario del sistema y ejecuta funciones globales al mismo.

Los procesos de tiempo real son los periódicos y esporádicos. Se agrupan en grupos de recuperación, que es la unidad básica de recuperación de errores. No existe límite en el número de estos procesos que pertenecen a cada grupo, pero cada proceso sólo puede pertenecer a un grupo. En cada grupo existe un único proceso de recuperación, que es el encargado de mantener el estado de error del grupo y de ejecutar la actividad de recuperación cuando algún proceso del grupo falle.

La comunicación entre procesos se realiza mediante monitores, para acotar y reducir la inversión de prioridades. Son procesos servidores, por lo tanto sólo ejecutan cuando algún proceso les solicita un servicio.

Los manejadores de interrupciones se construyen usando el mecanismo original de Ada. Los procesos de segundo plano son procesos sin restricciones temporales y deben ejecutar con menor prioridad los procesos de tiempo real. No pertenecen a ningún grupo de recuperación, pues suelen realizar operaciones que no son críticas al sistema.

8.3 Diseño del ejecutivo multitarea

8.3.1 Proceso supervisor del sistema

El modo del sistema es una característica global al mismo. Por este motivo el componente que le gestione debe ser visible a todos los procesos. Para tal fin se ha creado un proceso llamado el supervisor del sistema [2]. Este proceso es el más prioritario para asegurar la correcta e inmediata ejecución del protocolo de cambio de modo.

[2] Este proceso no tiene ninguna relación con la tarea supervisora empleada en los esquemas de las tareas.

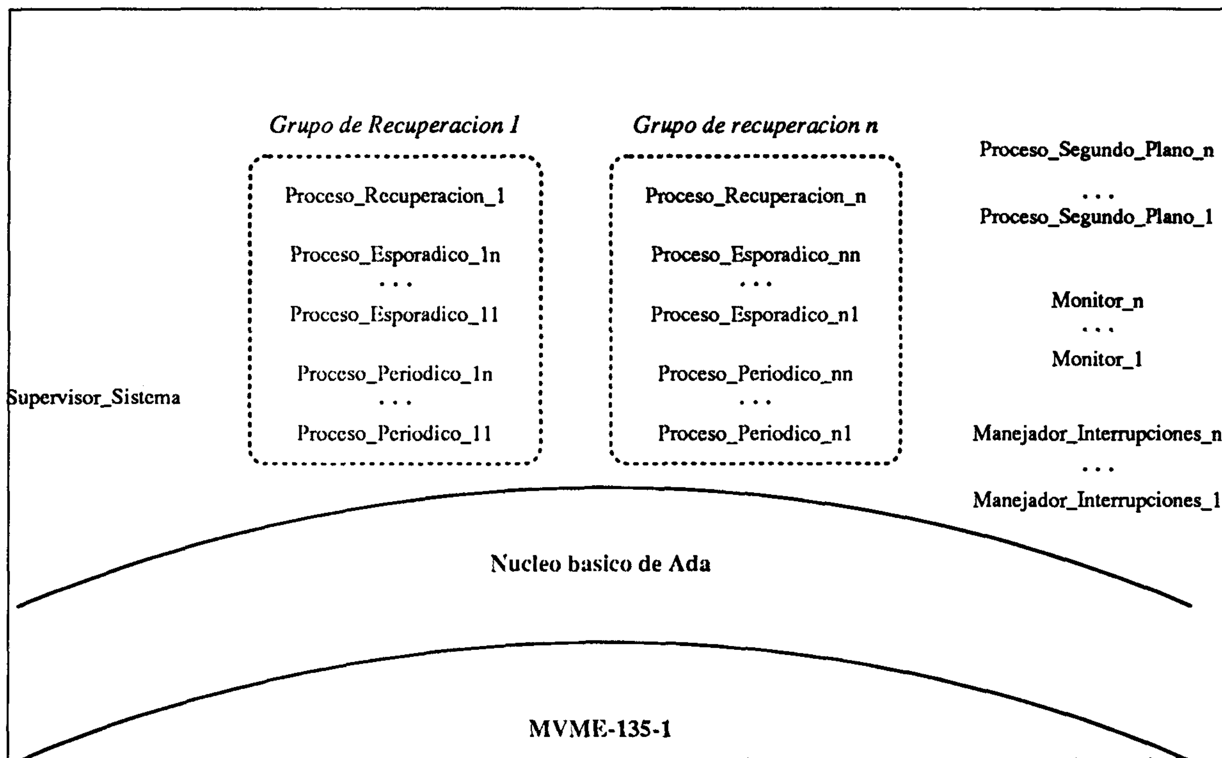

Figura 8.1: Esquema de un sistema creado con el ejecutivo multitarea

Existen dos procedimientos proporcionados por el usuario que el sistema debe ejecutar en circunstancias especiales:

- Procedimiento de inicio del sistema: debe ser el primer código de usuario que se ejecute.
- Procedimiento de parada segura del sistema: se debe ejecutar cuando se detecten fallos graves en la ejecución del sistema.

Por la importancia de estos dos procedimientos y por su carácter global, es inmediato encargar al proceso supervisor su gestión. De esta forma se garantiza, que dada la prioridad de este proceso, cuando sea necesario se ejecutarán inmediatamente.

El proceso supervisor del sistema exporta procedimientos para la gestión del modo y para ejecutar la rutina de parada segura del sistema. El procedimiento de inicio es el primer código que ejecuta el supervisor del sistema.

8.3.2 Procesos de tiempo real

Los procesos de tiempo real son los periódicos, esporádicos y de recuperación. En su concepción general, los procesos de recuperación son iguales a los periódicos. Las diferencias estriban en su inicio y el comportamiento ante errores. Estos aspectos de los procesos de recuperación se presentan en el apartado 8.3.3.

Durante la ejecución de un proceso de tiempo real, acontecen una serie de eventos que se deben detectar y tratar adecuadamente. Estos eventos son:

- Cada activación del proceso.
- Incumplimiento del plazo de respuesta.
- Uso de más memoria que la asignada.
- Cambio de modo del sistema.
- Errores de ejecución.
- Error en algún proceso del grupo de recuperación.

Requisitos de los procesos

Los procesos de tiempo real tienen que cumplir una serie de requisitos para ser útiles en el desarrollo de aplicaciones críticas y para adecuarse a las características deseadas del ejecutivo:

- Comportamiento general según la teoría de planificación.
- Activación de la actividad del usuario.
- Detección y tratamiento de errores de ejecución.
- Comprobación de cumplimiento de las restricciones temporales.

Esquema de ejecución

La mayoría de los requisitos anteriores se pueden satisfacer con un esquema de ejecución del proceso basado en una única tarea. Sin embargo esto no se cumple con el requisito relativo a la comprobación de cumplimiento de los plazos de respuesta. No es posible en Ada [AdaLRM83] programar una tarea de forma que ella misma evite excederse del plazo de respuesta.

Así pues, la aproximación adoptada se basa en crear por cada proceso de tiempo real dos tareas Ada:

- *Tarea funcional*: Ejecuta la actividad de la aplicación, según el tipo particular de proceso.

- *Tarea de control*[3]: Controla la ejecución de la tarea funcional. En particular sus funciones son:
  - Asegurar la ejecución de la tarea funcional según el perfil correspondiente al estado del sistema. Para ésto debe:
    * Conocer el modo inicial del sistema, para seleccionar el perfil de ejecución adecuado.
    * Detectar los cambios de modo del sistema y cambiar el perfil en consonancia.
    * Asegurar que la tarea funcional ejecuta con el perfil adecuado.
  - Detectar fallos en la tarea funcional y comunicarlos al grupo.
  - Detectar los fallos del grupo de recuperación y operar consecuentemente.

Las tareas de control de los procesos tienen todas la misma prioridad, que es mayor que cualquier tarea funcional y menor que la del proceso supervisor del sistema. Con este enfoque se asegura que cuando la tarea de control detecta un fallo, se trata inmediatamente. Esta aproximación puede generar inversión de prioridades, si mientras ejecuta la tarea de control se activa un proceso periódico más prioritario. Sin embargo, en ausencia de fallos se puede asegurar que cada proceso se verá en esta situación, como máximo, una vez en cada activación.

Durante la fase de diseño se estudió la opción consistente en asignar a la tarea de control una prioridad inmediatamente superior que la tarea funcional correspondiente. De esta forma se evita la inversión de prioridades. Sin embargo, si se incumple un plazo de respuesta, la tarea de control sólo tratará el fallo cuando sea la más prioritaria, lo que en determinadas situaciones tardará en ocurrir. Este retardo en el tratamiento del error puede afectar a la seguridad del sistema.

Es importante señalar que estas tareas se han diseñado para que su ejecución conjunta se comporte según las hipótesis de partida del comportamiento de los procesos periódicos, según la teoría de planificación de base: cuando un proceso se puede ejecutar, lo debe hacer sin pausas hasta completar el código de la activación en curso. Posteriormente se mostrará el cumplimiento de esta característica.

Esta implementación de procesos presenta dos problemas a tener en cuenta cuando se desarrollen aplicaciones con el ejecutivo multitarea:

1. El ejecutivo multitarea aunque detecta el incumplimiento de los plazos de respuesta, no detecta que un proceso utilice más tiempo de cómputo que el asignado. Para poder detectar esta situación, sería necesario disponer de temporizadores de tiempo de ejecución de una tarea y esta facilidad no la proporciona Ada. Se podría implementar mediante modificaciones en el núcleo básico, aunque no sería nada fácil.

[3]En los esquemas de tarea que se han presentado en los capítulos anteriores, a esta tarea se le denominaba **tarea supervisora**. En este capítulo se ha modificado la denominación para evitar confusiones con el proceso supervisor del sistema.

2. Si expira el plazo de respuesta de un proceso antes de que haya finalizado su ejecución, la tarea de control ejecuta inmdiatamente. De esta forma se puede producir un bloqueo por inversión de prioridades que afectaría a todos los procesos listos con prioridad intermedia entre el proceso que falla y la prioridad de la tarea de control. Esto es, nuevamente, debido a que en el diseño del ejecutivo multitarea se ha primado la seguridad del sistema. Cuando se detecta un fallo se trata inmediatamente. Este problema se puede solucionar reservando tiempo de cómputo para este fin. En el apartado 6.5 han presentado soluciones con más detalle.

Procesos periódicos

Un proceso periódico ejecuta una actividad con una cierta periodicidad y debe completarla en un plazo de tiempo. La actividad que ejecuta y la periodicidad dependen del modo de ejecución del sistema. Según la aproximación de diseño seguida, a continuación se muestra un esquema de ejecución de los procesos periódicos [4]:

```
task body Control_Task is
begin

  Set_Initial_Mode;
  delay Phase;
  Periodic_Task.Activate;

  loop
    select
      accept Activation_Finished;
      Next_Deadline := Next_Deadline + Process_Period;
     or
      delay Next_Deadline - Calendar.Clock;
      Report_Error;
      abort Periodic_Task;
    end select;

    Error_Checking;
    if Mode_Changing then
      Change_Mode;
    end if;
  end loop;

exception

 when others
     Report_Error;
      abort Periodic_Task;

end Control_Task;

task body Periodic_Task is
begin
```

[4]Se han mantenido los identificadores originales del ejecutivo multitarea.

```
  Activate;
    Set_Priority;
  end Activate;
  loop
    Activity;
    Control_Task.Activation_Finished;
    Next_Activation := Next_Activation + Process_Period;
    delay_until(Next_Activation);

    Error_Checking;
    if Mode_Changing then
      Control_Task.Activation_Finished;
    end if;

    if Mode_Changing then
      accept Activate do
        Set_Priority;
      end Activate;
  end loop;

exception
  when others
       Report_Error;
end Periodic_Task;
```

Con este esquema, cuando un proceso periódico se puede ejecutar, lo hace sin pausas hasta completar el código de la activación en curso. En efecto, cada activación del proceso periódico, coincide con el temporizador establecido en la tarea periódica mediante la instrucción `Delay_Until`. Entonces la tarea periódica, ejecuta `Activity`. Al terminar, comunica a la tarea de control el final de la ejecución. Inmediatamente, la tarea de control realiza una serie de operaciones y ejecuta una instrucción de aceptación temporizada, para detectar incumplimiento del siguiente plazo de respuesta. A continuación, el proceso periódico se bloquea hasta la siguiente activación.

Se comprueba si ha habido cambio de modo y si ha fallado algún proceso del grupo de recuperación al principio (tarea periódica) y al final (tarea de control) de cada activación. Si la tarea periódica detecta cambio de modo, despierta a la de control, para que sea ésta la que se encargue de ejecutar el protocolo de cambio de modo.

Procesos esporádicos.

Los procesos esporádicos son procesos que ejecutan como respuesta a un evento y tienen restricciones de tiempo críticas. El diseño de estos procesos se basa en el protocolo del servidor esporádico [Sprunt&89].

El algoritmo original prevee dar soporte a tareas aperiódicas, que no tienen restricciones críticas. El mecanismo consiste en asignar varias tareas aperiódicas a un servidor esporádico. Mientras que haya tiempo de ejecución disponible, se atenderán las peticiones existentes. Cuando se acabe el tiempo de ejecución disponible, se parará el proceso aperiódico y cuando llegue el siguiente instante de restauración y se reponga tiempo de

cómputo, continuará su ejecución exactamente en el instante donde se interrumpió. Dado que la única manera que se tiene para parar asíncronamente la ejecución de una tarea es mediante la sentencia `abort`, no es fácil modelar el comportamiento del servidor esporádico cuando da soporte a varias tareas aperiódicas. Además no se disponen de mecanismos suficientemente precisos para medir el tiempo de cómputo consumido en el tratamiento de un evento. Por tanto cada servidor esporádico sólo dará servicio a una tarea aperiódica. El tiempo de ejecución que se le asocia debe ser suficiente para ejecutar completemante una activación de la tarea aperiódica. A continuación se presenta un esquema simplificado de este proceso:

```
task body Control_Task is
begin
  loop
    accept Signal_Aperiodic_Event(Event_Info, Event_Time) do
       Current_Event_Info := Event_Info;
       Current_Event_Time := Event_Time;
    end accept;

    Delay_Until(Previous_Event_Time + Process_Inter_Arrival_Time);
    Previous_Event_Time := Current_Event_Time;

    Error_Checking;
    if Mode_Changed then
      Change_Mode;
    end if;

    Sporadic_Task.Activate;

    select
      accept Activation_Finished;
     or
      delay Current_Event_Time + Process_Deadline;
      Report_Error;
      abort Sporadic_Task;
    end select;

    Error_Checking;
  end loop;
end Control_Task;

task body Sporadic_Task is
begin
  loop
    accept Activate;
      Set_Priority;
    end Activate;

    Error_Checking;
    Activity(Current_Event_Info);
    Control_Task.Activation_Finished;
```

```
    Error_Checking;

  end loop;
end Sporadic_Task;
```

En este esquema, las dos tareas permanecen bloqueadas mientras no ocurran eventos asociados. Cuando se produce uno, se le comunica a la tarea de control mediante el procedimiento `Signal_Aperiodic_Event` que llama al punto de entrada del mismo nombre. Sus parámetros permiten comunicar el tiempo en que se produjo el evento, para determinar el plazo de respuesta, e información asociada al evento, para la actividad del usuario.

Cuando se activa la tarea de control, lo primero que hace es determinar si ha transcurrido el tiempo mínimo entre los dos últimos eventos. Si no es así, se espera a que transcurra para no ejecutar antes de tiempo. Esta situación se puede considerar como un error de ejecución. En tal caso habría que tomar las medidas adecuadas. En esta versión no se trata como un error.

La tarea de control está bloqueada desde que se produjo el anterior evento. Durante este intervalo de tiempo, algún proceso del grupo puede haber fallado o han podido ocurrir múltiples cambios de modo. Se comprueban estos dos extremos. En el primer caso, se para la ejecución del proceso y en el segundo se selecciona el perfil de ejecución adecuado.

A continuación se activa la tarea esporádica y se inicia un temporizador, para detectar incumplimiento de plazos, de forma análoga al proceso periódico. Finalmente se comprueba si ha fallado algún proceso del grupo durante la ejecución de la actuividad.

La tarea esporádica simplemente espera ser activada, entonces establece su prioridad según el modo del sistema y ejecuta la actividad, comunica su finalización a la tarea de control.

Las comprobaciones de errores en procesos del grupo se han incluido en la tarea esporádica para que termine por sí sóla. En caso contrario, tendría que ser abortada. Por lo demás el tratamiento de errores es igual que el proceso periódico.

8.3.3 Tratamiento de errores

Las características de robustez inherentes a sistemas de tiempo real, imponen la necesidad de detectar y tratar los fallos que acontezcan durante la ejecución. Este requisito se ha plasmado en los siguientes principios del diseño del ejecutivo:

- *Encapsulación de errores.* Todos los componentes del ejecutivo están diseñados de forma que cualquier fallo que se produzca se detecta y se trata internamente. Un fallo en un componenente nunca se va a propagar incontroladamente a otros elementos del sistema.

- *Detección de fallos.* Los componentes de ejecutivo son capaces de detectar fallos en su ejecución. Este principio es especialmente evidente en los procesos de tiempo real donde se detectan fallos en la ejecución del código, desbordamiento de la

zona de memoria asignada, incumplimiento de plazos de respuesta y fallos en algún monitor empleado.

- *Grupos de recuperación.* Para proporcionar un cierto nivel de tolerancia a fallos se ha empleado la idea del grupo de recuperación. Está formado por un comjunto de procesos de tiempo relacionados entre sí y un proceso de recuperación. Cuando algún proceso del grupo falla, el resto detiente la ejecución y se activa al proceso de recuperación.
- *Parada segura del sistema.* Si se producen errores especialmente graves en la ejecución del sistema, se ejecuta una rutina de parada segura del sistema, proporcionado por el diseñador de la aplicación.

A continuación se profundiza en los aspectos de la detección de fallos y en el funcionamiento de los grupos de recuperación.

Detección de errores en los procesos.

Si se quiere implementar un mecanismo eficiente de tratamiento de fallos, el primer paso es detectar su ocurrencia. A continuación se analiza cómo se detectan estos fallos.

Fallos detectados por la tarea de control.

Fallo en algún proceso del grupo de recuperación. Al principio y al final de cada activación se consulta si ha fallado algún proceso del grupo. Si es así, se termina la ejecución y se establecen los medios para que la tarea periódica detecte el evento y termine.

Incumplimiento del plazo de respuesta del proceso periódico. La tarea de control queda bloqueada en una sentencia de aceptación temporizada. El temporizador expira coincidiendo con el final del plazo de respuesta. Si la tarea periódica no ha llamado al punto de entrada cuando éste expira, se ha incumplido el plazo de respuesta y se aborta la tarea periódica, pues interesa que cese su ejecución tan pronto como sea posible. A continuación se comunica el suceso al proceso de recuperación y se completa ejecución del proceso.

Fallo en la ejecución de la tarea de control. La implementación de la tarea de control incluye el código necesario para tratar cualquier fallo que se produzca durante su ejecución. Típicamente este fallo se manifestará como una excepción. Ante tal situación se para la tarea periódica, se comunica el error al proceso de recuperación y se termina la ejecución.

Fallos detectados por la tarea de actividad.

Desbordamiento del área de memoria asignada. Las tareas de actividad se declaran como instancias de un tipo tarea. De esta forma se puede establecer un límite a la memoria que usa la tarea. Si esta zona se desborda, el sistema eleva la excepción `Storage_Error`. Como parte de la implementación de la tarea, se incluye un manejador de esta excepción. El tratamiento consiste en comunicar el fallo a la tarea de control y al proceso de recuperación y finalmente terminar la ejecución.

Fallo en la ejecución de tarea de actividad. La implementación de la tarea de actividad incluye el código necesario para tratar cualquier excepción que se produzca mientras ejecuta. Las acciones que se toman coinciden con el caso anterior.

Fallo en la ejecución de un monitor. La comunicación entre procesos se realiza por medio de monitores. Si un monitor detecta un fallo en su ejecución, termina de ejecutar. Como consecuencia, el proceso periódico no podrá solicitar servicios al monitor. Por tanto, un fallo en un monitor se debe tratar como si fuera el propio proceso el que falla. En este caso, el fallo se detecta porque si el proceso periódico invoca algún servicio del monitor, si no existe, al establecer la llamada, se eleva en la excepción `Tasking_Error` en la tarea de actividad.

Puede ocurrir que el fallo se produzca mientras ejecuta la cita. En este caso se eleva la anterior excepción simultáneamente en la tarea que llama y en el monitor.

Esta excepción se trata como cualquier otra elevada al ejecutar la tarea de actividad

Grupos de recuperación.

Un grupo de recuperación está formado por un conjunto de procesos de tiempo real y un proceso de recuperación. Estos procesos están relacionados en el sentido que si uno de ellos falla, no será posible que el resto ejecute corréctamente. Por tanto, cuando uno de los procesos falla, se lo comunica al proceso de recuperación y termina. Antes y después de cada ejecución el resto de los procesos comprueba si algún proceso del grupo ha fallado y, en caso afirmativo, terminan su ejecución.

El proceso de recuperación está compuesto por tres tareas. Dos de ellas son la tarea de control y la tarea de actividad. Su estructura y funcionamiento es prácticamente la misma que la de un proceso periódico, excepto en referencia al tratamiento de fallos. La tercera es una tarea que gestiona el estado de fallo del grupo de recuperación y activa a la tarea de control cuando se produce un fallo. Los procesos de recuperación exportan procedimientos para que los restantes procesos del grupo de recuperación consulten el estado de fallo del grupo y comuniquen la ocurrencia de un fallo.

Cuando se comunica un fallo, la tarea gestora espera a que todos los proceso del grupo hayan finalizado, para evitar posibles interferencias entre ellos y la ejecución de la actividad de recuperación. A continuación, se activa la tarea de control y termina la tarea gestora. A partir de este momento, la ejecución del proceso de recuperación coincide con un proceso periódico.

Evidentemente, existe diferencia en el comportamiento del proceso ante errores, respecto al de un proceso periódico. Cuando el programador crea un grupo de recuperación, determina su comportamiento en caso de fallo. Las alternativas son:

- *Parar el proceso.* El proceso detiene su ejecución sin ejecutar ninguna acción adicional.
- *Parar el sistema.* El proceso detiene la ejecución del sistema invocando la rutina de parada segura del sistema.

Restricciones del modelo.

Al realizar el análisis de planificabilidad un parámetro básico es el tiempo de cómputo de un proceso. Una situación normal de fallo de un proceso ocurre cuando excede su tiempo de cómputo, aunque cumpla su plazo de respuesta. Además, un proceso que use más tiempo de cómputo que el asignado puede causar que procesos menos prioritarios incumplan sus plazos de respuesta.

Por consiguiente, para que el sistema detectara todos los fallos se debería controlar que los procesos no utilicen más tiempo de cómputo que el asignado. Sin embargo, esta funcionalidad es complicada de implementar, especialmente cuando el núcleo básico de ejecución no proporciona ayuda alguna.

Por este motivo, el ejecutivo multitarea no detecta esta eventualidad. Este problema no es exclusivo de esta aplicación, sino que es un problema del lenguaje de programación Ada cuando se desarrollan sistemas de tiempo real crítico.

8.3.4 Cambio de modo

El protocolo de cambio de modo que se ha implementado está basado en el que se presentó en el apartado 4.5. En la propuesta que se detalla, se utiliza Ada estándar, con las modificaciones mencionadas. Las prioridades dinámicas son imprescindibles, ya que sino no tendría sentido cambiar el modo del sistema.

El protocolo original contempla la adición y borrado de procesos y el consiguiente cambio del techo de prioridad del paradigma de comunicación entre procesos utilizado. En la implementación del ejecutivo multitarea se ha optado por asignar a los monitores una prioridad igual al máximo techo de prioridad entre los modos del sistema. De esta forma el protocolo de cambio de modo es más eficiente y se simplifica su implementación. Por contra, esta aproximación aumenta el número de procesos a los que se les provoca inversión de prioridades. Sin embargo este efecto no es grave y se puede contemplar fácilmente en el análisis de planificabilidad.

A continuación se detalla la implementación desarrollada.

Gestión del modo del sistema

El modo del sistema es una información global al mismo. Por esta razón se decidió que el proceso supervisor del sistema debe gestionarle. El hecho de que sea el proceso más prioritario facilita la implementación de esta funcionalidad. El supervisor del sistema, para gestionar el cambio de modo:

- Proporciona operaciones para que los procesos del sistema puedan conocer el modo de ejecución, cambiar de modo y para quedar bloqueados durante un modo de ejecución. La especificación de las operaciones exportadas por el supervisor del sistema, en relación con el cambio de modo es:

```
generic
   type Mode_Type is (<>);
   --enumeration type that represents
   --system modes
    . . .
package System_Supervisor is

function Current_System_Mode return Mode_Type;
--Returns the current system execution mode.

procedure Change_Mode (New_Mode : in Mode_Type);
--Change the current system mode to a new one
--provided as parameter. If the new mode is the
--same as the current mode, nothing is done.

procedure Get_Mode_Change_Status
   (Is_Mode_Changing     : in out Boolean;
    Mode_Change_End_Time : in out Calendar.Time);
--This procedure returns the current mode change status. This
--consists of an indication whether the system is currently
--performing a mode change and, if so, a forecast of the time
--when the mode change will be finished.

procedure Sleep;
--This procedure suspends the execution of the
--calling process for as long as the system
--remains in the current mode.
    . . .
end System_Supervisor;
```

- Ejecuta el protocolo de cambio de modo.

El supervisor del sistema mantiene el modo del sistema. Los procesos de tiempo real pueden conocer su valor ejecutando el procedimiento Current_System_Mode. No es necesario establecer mecanismos de protección, pues el proceso supervisor es el único que modifica este valor y es el más prioritario del sistema.

Cualquier proceso puede cambiar el modo del sistema, ejecutando el procedimiento Change_Mode. Este procedimiento enmascara una llamada a un punto de entrada del

supervisor. Cuando un proceso cambia el modo del sistema, el supervisor ejecuta el protocolo de cambio, que está básicamente compuesto por las siguientes acciones:

- Despierta a todos los procesos que estuvieran inactivos en el modo anterior.
- Espera el tiempo mínimo necesario para asegurar que todos los procesos del sistema han completado la ejecución en curso. Este retardo es igual al máximo de los períodos o separación entre eventos.
- Modifica una serie de variables internas para que los procesos sepan el instante de finalización del cambio de modo y para que, en circunstancias especiales, sepan si se está ejecutando el protocolo de cambio de modo.

El proceso supervisor utiliza el punto de entrada `Sleep` para bloquear los procesos que deben permanecer inactivos en el modo actual de operación. Cada vez que se produce un cambio de modo, se desbloquean todos los procesos en espera en el citado punto de entrada. Cada procesos determinan si en el modo de operación nuevo son activos o si debe continuar bloqueado en cuyo caso volverán a llamar al procedimietno `Sleep` del proceso supervisor.

Adhesión de los procesos al protocolo de cambio de modo.

Los procesos comprueban si hay cambio de modo al principio y al final de cada activación. Acceden al modo actual del sistema, y si es diferente al modo del proceso, entonces se ha producido un cambio de modo. Si el proceso permanece inactivo se llama al procedimiento `Sleep`. Si es activo calcula el siguiente instante de ejecución a partir del tiempo de finalización del cambio de modo y la fase del proceso. El procedimimiento `Get_Mode_Change_Status` devuelve una variable lógica, que indica si el protocolo de cambio de modo ha finalizado, y el instante de finalización del protocolo. Esta operación la realizan todos los procesos del sistema, aún en el caso que no tengan que cambiar su perfil de ejecución.

Este algoritmo es diferente para los procesos esporádicos, pues si no reciben notificaciones de eventos, pueden permanecer sin ejecutar durante varios modos. Así pues puede ocurrir que cuando son activados, detecten que ha habido un cambio de modo, aunque este ya se haya completado y ejecuten innecesariamente el protocolo de cambio de modo. En este caso, deben ejecutar el procedimiento `Get_Mode_Change_Status` para compronar si se está produciendo efectivamente un cambio de modo. Si es así, el proceso ejecuta el protocolo igual que los procesos periódicos. En caso contrario, cambia su perfil y ejecuta inmediatamente. Como el cambio de modo ha finalizado, el proceso puede ejecutar con la seguridad de que habrá tiempo de cómputo suficiente.

Es importante que cuando se produzca la primera activación de un proceso después de un cambio de modo, ejecute con la prioridad adecuada. Respecto a la tarea de control, su prioridad es constante, y por tanto no hay que realizar acción alguna.

Asignar la prioridad nueva en el instante adecuado a la tarea de actividad es más complicado. En caso de cambio de modo se bloquea a la tarea de actividad en un punto

de entrada esperando la llamada de la tarea de control. Antes de que ésta se bloquee en espera de la siguiente activación, llama al punto de entrada. En el código de la cita se determinan el perfil de ejecución del nuevo período. A continuación la tarea de actividad queda bloqueada hasta la siguiente activación.

Con esta implementación se consigue que todas las tareas de actividad de los procesos comiencen a ejecutar con el perfil adecuado, evitando inversiones de prioridad potenciales. Además, el tiempo de ejecución del protocolo de cambio de modo de cada proceso se puede contabilizar y tener en cuenta en el análisis de planificabilidad.

8.3.5 Comunicación entre procesos

Los requisitos del ejecutivo multitarea imponían la necesidad de disponer de mecanismos para comunicar procesos. Esta operación puede provocar inversión de prioridades.

El lenguaje de programación Ada complica este panorama. Cuando varias tareas llaman a un mismo punto de entrada, se encolan según el orden de llegada. Esta aproximación puede provocar inversión de prioridades, si en la cola de un punto de entrada hay tareas encoladas y se da la situación que tareas menos prioritarias han ejecutado la sentencia de llamada antes una tarea más prioritaria. Algo similar ocurre con la sentencia de selección entre múltiples puntos de entrada. Si hay tareas esperando en diversos puntos de entrada, la selección de la tarea es arbitraria.

En [Sha&90][Sha&89a] se proponen métodos para comunicar procesos en Ada, basándose en el protocolo del techo de la prioridad. El método que se ha usado es el denominado protocolo del techo de prioridad inmediato.

La comunicación entre procesos en el ejecutivo multitarea se basa en el modelo cliente-servidor. Se realiza mediante el componente llamado `Monitor`. Está implementado como una tarea a la que se le asigna una prioridad constante e igual a su techo de prioridad. La estructura del monitor es una bucle infinito que ejecuta una sentencia de selección múltiple. Cada uno de los puntos de entrada permite solicitar un servicio al monitor.

Con este enfoque, se puede asegurar que no habrá ninguna tarea esperando en las colas de los puntos de entrada de un monitor. En efecto, si la tarea de actividad de un proceso llama un punto de entrada de un monitor, se ejecuta la cita con la prioridad de éste, que es mayor que la prioridad de cualquier tarea que se comunique con él. Por consiguiente, no se podrá producir otra petición al monitor en cuestión hasta que se complete la anterior.

Esta estructura es adecuada al mecanismo de tolerancia de fallos del ejecutivo multitarea. Si el monitor falla, los procesos que le usan fallarán, con las consecuencias conocidas en los grupos de recuperación correspondientes. Un proceso detecta el fallo de un monitor porque se activa una excepción Ada, que provoca que el proceso falle.

La prioridad de un monitor es fija en todos los modos del sistema. Su valor es inmediatamente mayor que la prioridad máxima entre todos los procesos que acceden al monitor en cuestión en cualquier modo del sistema. El motivo de esta decisión es simplificar la implementación del cambio de modo. El aspecto negativo es que se produce alguna situación adicional, aunque éstas se consideran en el análisis de planificabilidad.

En cualquier caso, es recomendable reducir en lo posible el tiempo de ejecución de los tiempos de ejecución de los servicios de un monitor. Esta práctica permite reducir cualquier caso de inversión de prioridades y aumentar la capacidad de procesador disponible.

8.3.6 Análisis de la planificabilidad

El análisis de planificabilidad del ejecutivo multitarea sigue las directrices expuestas en el apartado 6.5. La única diferencia es que el manejador de interrupciones que se emplea en el ejecutivo puede provocar casos adicionales de inversión de prioridades. Este caso hay que considerarlo al calcular el tiempo de bloqueo de los procesos.

Los requisitos funcionales del ejecutivo [CASA&91a] especificaban la necesidad de proporcionar una plantilla para crear manejadores de interrupciones según el enfoque de Ada. Cuando se produce una interrupción el núcleo básico realiza una llamada a un punto de entrada concreto. El manejador se ejecuta con prioridad máxima, pudiendo producir inversión de prioridad.

La inversión de prioridad producida por los manejadores de interrupciones está acotada por el tiempo de separación entre eventos. Un proceso sufrirá bloqueos en tantas ocasiones como el máximo número de interrupciones que se puedem producir durante una activación. El tiempo máximo de bloqueo en una activación, $t_{interrupcion}$, será igual a la suma del tiempo de cómputo de los manejadores asociados a estas interrupciones.

El tiempo de cómputo de un manejador de interrupciones se contabiliza como parte del proceso esporádico correspondiente. Por tanto, este efecto sólo se debe contabilizar en aquellos procesos más prioritarios que éste.

El tiempo de bloqueo total de un proceso, incluyendo el efecto de los manejadores de interrupciones, será:

$$B_i = \max(t_{monitor}, t_{control}) + t_{interrupcion}$$

8.4 Desarrollo de sistemas de tiempo real

El objetivo de este capítulo es proporcionar un conjunto de directrices básicas para desarrollar sistemas de tiempo real con el ejecutivo multitarea. Para lograr esta meta se irán determinando las directrices al mismo tiempo que se desarrolla un sistema, a partir de unas especificaciones. De esta forma se podrá comprobar inmediatamente las normas que se proporcionan.

8.4.1 Directrices metodológicas

Prácticamente ninguno de los métodos actuales de diseño de sistemas de tiempo real considera las ventajas metodológicas, inherentes a la teoría de planificación en que se

basa el ejecutivo multitarea. El objetivo de este apartado es proporcionar un conjunto de directrices que permitan obtener los máximos beneficios de esta teoría.

Estas directrices no pretenden ser exhaustivas y están relacionadas con las fases de diseño detallado y codificación. Lo más conveniente sería integrarlas con el método de diseño empleado en cada caso. A continuación se enumeran estas directrices:

1. Especificación inicial del sistema. Las recomendaciones que se van a proporcionar parten de un estado en el desarrollo en que se conocen los procesos que componen el sistema y sus características básicas. En particular, los datos que se deben conocer son:

   (a) Modos de ejecución del sistema.

   (b) Procesos.

   (c) Grupos de recuperación.

   (d) Monitores para comunicación entre procesos.

   (e) Manejadores de interrupciones.

   A partir de estos datos se debe asignar prioridades a los componentes, de acuerdo con la teoría de planificación empleada.

2. Análisis de planificabilidad estimativo. A partir de las características de los procesos y de algunos parámetros estimados, como el tiempo de cómputo, se analiza si el sistema es planificable.

3. Modificación del período de los procesos. Si no se puede garantizar que todos los procesos de tiempo real cumplan sus restricciones de tiempo, se puede seleccionar un subconjunto del que se pueda. Para tal fin, puede ser necesario modificar los períodos de algunos procesos.

4. Programación del sistema. Se desarrolla el código del sistema.

5. Análisis de planificabilidad con datos reales. Se miden los tiempo de cómputo precisos de los procesos y con estos datos se vuelve a realizar el análisis de planificabilidad.

6. Modificación de período de procesos. Si con los datos reales cambian los resultados del análisis de planificabilidad previo y no se pueden garantizar los plazos de respuesta de todos los proceso, puede ser necesario modificar el período de algún proceso.

Estas directrices no se deben seguir, estrictamente, en secuencia. Cuando se llega a un punto no satisfactorio, se puede seguir hacia adelante y tomar medidas correctoras, o volver hacia atrás y replantearse el diseño básico de los procesos. Por ejemplo, si el análisis de planificabilidad estimativo no es satisfactorio, se puede modificar los períodos o replantearse la especificación inicial de los procesos.

8.4.2 Especificación de la aplicación ejemplo

La especificación de un sistema hipotético y con el nivel detalle necesario es el punto de partida de la aplicación ejemplo. Inicialmente no se dispone del código, por tanto, los tiempos de cómputo que se proporcionan son una estimación de los tiempo reales. Evidentemente, estos valores reales no se conocerán con exactitud hasta que no se hagan medidas con el código final. Las características del sistema no son necesariamente realistas. Su único fin es ilustrar las directrices de diseño que se proporcionan.

A continuación se muestra la especificación del sistema. [5]

1. Tres modos de ejecución:
   - `Normal_Mode`.
   - `Critical_Mode`.
   - `Alarm_Mode`.
2. El conjunto de procesos que se detalla a continuación. Inicialmente se conocen los parámetros de ejecución relacionados con las especificaciones funcionales.

   (a) Cuatro procesos periódicos.

Proceso	`Normal_Mode`				`Critical_Mode`				`Alarm_Mode`			
	T	Φ	pr	C	T	Φ	pr	C	T	Φ	pr	C
PP_1	400	0		100	200	0		20	400	0		100
PP_2	600	0		150	250	0		20	600	0		150
PP_3	900	0		150	900	0		100	—	—	—	—
PP_4	—	—	—	—	—	—	—	—	250	0		50

   (b) Dos procesos esporádicos.

Proceso	`Normal_Mode`				`Critical_Mode`				`Alarm_Mode`			
	T	Φ	pr	C	T	Φ	pr	C	T	Φ	pr	C
PE_1	500	300		20	500	300		20	500	300		20
PE_2	800	500		40	600	250		20	600	500		20

   Cada proceso esporádico trata eventos relacionados con dispositivos que interrumpen al procesador. Posteriormente se relacionan los procesos esporádicos con los manejadores de interrupciones que les comunican la ocurrencia de los eventos.

   (c) Dos procesos de segundo plano.

   Los procesos de segundo plano son PSP_1 y PSP_2. Las características de ejecución de los procesos de segundo plano no son relevantes para el estudio en curso. La ejecución de los procesos de segundo plano es independiente del modo de ejecución del sistema.

[5] La unidad de los valores de tiempo que se proporcionan es el milisegundo.

3. Grupos de recuperación.

   Hay dos grupos de recuperación, que se corresponden con cada proceso de recuperación. La relación de procesos en cada grupo es como sigue:

Grupo Rec.	Proceso de Rec.	Procesos de tiempo real
Grupo 1	PR_1	PP_1 PP_2 PE_1 PE_2
Grupo 2	PR_2	PP_3 PP_4

   Recuérdese que después de detectar un error en el grupo, los procesos de recuperación ejecutan como procesos periódicos. Las características de los procesos de recuperación son las que siguen:

Proceso	Normal_Mode				Critical_Mode				Alarm_Mode			
	T	Φ	pr	C	T	Φ	pr	C	T	Φ	pr	C
PR_1	300	0		60	300	0		60	300	0		60
PR_2	500	0		50	500	0		50	250	0		30

4. Dos monitores para la comunicación entre procesos. Los relación entre monitores y procesos que los usan es:

Monitor	Procesos que le usan
M_1	PP_1 PP_3
M_2	PP_2 PE_2

   Cada monitor proporciona dos servicios a los procesos usuario. A continuación se especifica el tiempo estimado máximo de los monitores al proporcionar un servicio:

Monitor	T. cómputo
M_1	10
M_2	20

5. Dos manejadores de interrupciones.

   Se supone que hay conectados al computador central dos dispositivos externos que interactúan mediante interrupciones. En la aplicación habrá dos manejadores de interrupciones para tratarlas. Se tiene la siguiente relación entre manejadores de interrupciones, la interrupción que trata y el proceso que debe tratar el evento:

Manejador de Interrupción	Vector de Interrupción	Proceso Esporádico	T. Cómputo
MI_1	16#140#	PE_1	2
MI_2	16#144#	PE_2	2

8.4.3 Asignación de prioridades

La asignación de prioridades a los procesos de una aplicación desarrollada con el ejecutivo multitarea se hace de acuerdo con el algoritmo de prioridad al proceso con el menor plazo de ejecución. Las reglas básicas son la siguientes:

1. *Prioridades predefinidas.* Las siguientes prioridades son fijas para cualquier aplicación desarrollada con el ejecutivo multitarea. El usuario no tiene que hacer nada porque está asignadas en el código fuente del ejecutivo.

   - Proceso supervisor del sistema: El proceso supervisor del sistema es el más prioritario del sistema. No hay que asignarle prioridades, pues siempre ejecuta con la máxima.
   - Tareas de control de procesos de tiempo real. Los procesos de tiempo real están compuestos por, al menos dos tareas. Las tareas de control siempre ejecutan con la prioridad inmediatamente inferior a la del proceso supervisor.

   Las prioridades que asigne el usuario siempre serán menores que la de las tareas de control.

2. *Monitores.* A los monitores se les asigna una prioridad fija para todos los modos de ejecución del sistema y tiene que ser superior a la mayor para cualquier modo, entre los procesos que les usan. Es recomendable que la prioridad del monitor sea inmediatamente superior a ésta.

3. *Procesos de tiempo real.* Se asignan prioridades a los procesos de tiempo real para cada modo de ejecución del sistema. Para tal fin se ordenan por plazo de ejecución. Nótese que el plazo de ejecución de los procesos periódicos coincide con su período. La prioridad de cada proceso en cada modo debe cumplir:

   - Para todo par de procesos, PTR_i y PTR_j, de tiempo real y para cualquier modo k, si sus plazos de ejecución en el modo k, D_{ik} y D_{jk}, son tales que $D_{ik} > D_{jk}$, entonces sus prioridades en el mismo modo, pr_{ik} y pr_{jk}, deben cumplir: $pr_{jk} > pr_{ik}$
   - Para todo proceso, PTR_i, de tiempo real y para todo monitor, M_j, al que accede, se debe cuplir que la prioridad del proceso para cualquier modo sea menor que la prioridad del monitor.

   Los procesos de recuperación están incluidos en este epígrafe. Al asignarles prioridades hay que tener en cuenta que cuando ejcuten la actividad de recuperación, todos los procesos de su grupo habrán acabado.

4. *Procesos de segundo plano.* A los procesos de segundo plano se les asigna una prioridad menor que a cualquier proceso de tiempo real.

Asignación de prioridades en la aplicación ejemplo

En este ejemplo el rango de prioridades es 1..99. Este rango es el del tipo `Priority_Type`, definido en el paquete `Run_Time_System_Interface`.

Para asignar prioridades a los procesos se ordenan según su plazo de ejecución para cada modo:

Proceso		
Normal_Mode	Critical_Mode	Alarm_Mode
PE_1	PP_1	PR_2
PR_1	PE_2	PP_4
PP_1	PP_2	PE_1
PE_2	PR_2	PR_1
PR_2	PE_1	PP_1
PP_2	PR_1	PE_2
PP_3	PP_3	PP_2

Se determinan los procesos más prioritarios que acceden a cada monitor:

Monitor	Proceso más prioritario	Modo
M_1	PP_1	Normal_Mode
M_2	PP_4	Alarm_Mode

Hay varias formas de asignar prioridades que se ajustan a las condiciones expresadas anteriormente. A continuación se presentan una de ellas:

Proceso	Prioridad		
	Normal_Mode	Critical_Mode	Alarm_Mode
M_1	97	97	97
M_2	95	95	95
PP_1	93	96	91
PP_2	90	93	89
PP_3	89	89	—
PP_4	—	—	94
PE_1	96	91	93
PE_2	92	94	90
PR_1	94	90	92
PR_2	91	92	96
PSP_1	88	88	88
PSP_2	87	87	87

Respecto a esta asignación se puede resaltar:

- Cuando dos procesos tienen el mismo plazo de ejecución, se puede asignar prioridad mayor a uno al otro o asignarles igual prioridad. Esta decisión sólo tiene influencia importante cuando el conjunto no es planificable.

- En este caso, se podría haber asignado la misma prioridad al proceso de recuperación PR_1 que a PE_1 en el modo `Normal_Mode`, aunque el segundo tiene menor plazo de ejecución que el primero. La razón es que PR_1 sólo ejecutará cuando falle algún proceso de su grupo y todos los procesos del grupo hayan terminado. Como PE_1 es del grupo, cuando ejecute PR_1, PE_1 habrá terminado y no habrá problemas por asignarles igual prioridad.

  En cualquier caso, no es aconsejable este ajuste fino de prioridades, a no ser que los niveles de prioridad no sean suficientes.

- Es importante asignar prioridades tan bajas a los monitores como sea posible, para disminuir inversiones de prioridades no deseadas.

8.4.4 Análisis de planificabilidad

El análisis de planificabilidad que se va a presentar emplea los tiempo de cómputo estimativos. De esta forma se puede tener en las primeras fases de diseño una primera estimación sobre la planificabilidad de la aplicación. A continuación, se analiza la planificabilidad del sistema en el modo `Normal_Mode`. En el resto de los modos se haría de forma análoga.

Cálculo de tiempos de bloqueo en la aplicación ejemplo

A continuación se van a calcular los tiempos de bloqueo de los procesos de la aplicación ejemplo. Esta operación sólo se realizará para el modo `Normal_Mode`. Para los otros modos el cálculo es análogo. Todos los valores máximos y comparaciones entre prioridades que se mencionan en el resto del apartado son relativas a este modo de ejecución.

En este caso se desprecia el tiempo de inversión de prioridad producidos por tareas de control al detectar el incumplimiento de un plazo de respuesta.

A continuación se calcula el tiempo de cómputo de los procesos ordenados por su prioridad. El tiempo de bloqueo por las tareas de control es igual en todos los casos y su duración es menor que la producida por monitores, por lo que no se condidera en el cálculo.

1. PE_1:

   - *Monitores*: Sólo puede generar inversión de prioridad el monitor M_1, ya que la prioridad del monitor M_2 es inferior a la del proceso. Por consiguiente el tiempo máximo de bloqueo por esta causa es 10 unidades de tiempo (u.t.).
   - *Manejadores de interrupciones*: El manejador de interrupciones MI_1 está relacionado con el proceso en cuestión y, por tanto, no produce inversión de prioridad. El tiempo de cómputo del manejador está contabilizado en el tiempo máximo de cómputo del proceso.

     El manejador de interrupciones MI_2 puede producir inversión de prioridad. En cada activación del proceso se producirá, como máximo, una interrupción tratada por MI_2. El tiempo de bloqueo máximo por esta causa es 2 u.t.

El tiempo máximo de bloqueo del proceso es 12 u.t.

2. PP_1:

   - *Monitores*: Como la prioridad del proceso se encuentra entre la prioridad de los monitores y la de los procesos menos prioritarios que acceden a ellos, ambos monitores pueden bloquear al proceso. Por consiguiente, el tiempo de bloqueo máximo por esta causa es 20 u.t.
   - *Manejadores de interrupciones*: Como el manejador MI_1 está relacionado con el proceso más prioritario PE_1, no produce inversión de prioridad a PP_1. Este razonamiento es aplicable al resto de los procesos.

     El manejador MI_2 puede producir inversión de prioridad y en cada activación de PP_1 se producirá una interrupción tratada por aquél, como máximo.

     El tiempo de bloqueo máximo por esta causa es 2 u.t.

   El tiempo máximo de bloqueo del proceso es 22 u.t.

3. PE_2:

   - *Monitores*: Los dos monitores pueden producir inversión de prioridades. El tiempo de bloqueo máximo por esta causa es 20 u.t.
   - *Manejadores de interrupciones*: El manejador de interrupciones MI_2 está relacionado con el proceso en cuestión y, por tanto, no produce inversión de prioridad. El tiempo de cómputo del manejador está contabilizado el tiempo máximo de cómputo del proceso.

   El tiempo máximo de bloqueo del proceso es 20 u.t.

4. PP_2:

   - *Monitores*: Este proceso es el menos prioritario que accede al monitor M_2, por lo tanto éste no producirá inversión de prioridades. El tiempo de bloqueo máximo por monitores, está motivado por M_1 y su valor es 10 u.t.
   - *Manejadores de interrupciones*: Como todos los manejadores están asociados a procesos más prioritarios no producen inversión de prioridades.

   El tiempo máximo de bloqueo del proceso es 10 u.t.

5. PP_3: Este proceso es el menos prioritario entre los procesos de tiempo real y no hay manejadores de interrupciones asociados a procesos de segundo plano. Por consiguiente, no sufre inversión de prioridades.

Aplicación del método.

Para comprobar si el conjunto de procesos es planificable, se comienza con la condición suficiente. Si esta falla, se aplica la condición necesaria y suficiente.

1. PE_1:

$$I_1 = 0$$

$$\frac{20}{300} + \frac{0}{300} + \frac{12}{300} < 1$$

Por tanto PE_1 es planificable.

2. PP_1:

$$I_2 = \left\lceil \frac{400}{500} \right\rceil 20 = 20$$

$$\frac{100}{400} + \frac{20}{400} + \frac{22}{400} = \frac{142}{400} < 1$$

Por tanto PP_1 es planificable.

3. PE_2:

$$I_2 = \left\lceil \frac{500}{500} \right\rceil 20 + \left\lceil \frac{500}{400} \right\rceil 100 = 20 + 2 \times 100 = 220$$

$$\frac{40}{500} + \frac{220}{500} + \frac{20}{500} = \frac{260}{500} < 1$$

Por tanto PE_2 es planificable.

4. PP_2:

$$I_2 = \left\lceil \frac{600}{500} \right\rceil 20 + \left\lceil \frac{600}{400} \right\rceil 100 + \left\lceil \frac{600}{800} \right\rceil 40 = 2 \times 20 + 2 \times 100 + 40 = 280$$

$$\frac{150}{600} + \frac{280}{600} + \frac{10}{600} = \frac{440}{600} < 1$$

Por tanto PP_2 es planificable.

5. PP_3:

$$I_2 = \left\lceil \frac{900}{500} \right\rceil 20 + \left\lceil \frac{900}{400} \right\rceil 100 + \left\lceil \frac{900}{800} \right\rceil 40 + \left\lceil \frac{900}{600} \right\rceil 150 = 2\times20+3\times100+2\times40+2\times150 =$$

$$\frac{150}{900} + \frac{720}{900} + \frac{0}{900} < \frac{870}{900} < 1$$

Por tanto PP_3 es planificable.

Por consiguiente, se puede garantizar que en el modo `Normal_Mode` todos los procesos cumplen sus plazos de respuesta.

Es inmediato comprobar que la ocupación de procesador de los procesos de recuperación es bastante menor que la de los procesos a los que reemplaza. Por consiguiente cuando ejecuten se puede garantizar que los procesos cumplen sus plazos de respuesta.

8.4.5 Creación de la aplicación

Finalmente es interesante mostrar cómo realizar aplicaciones con el ejecutivo multitarea. Por la extensión y el empleo de código fuente, se ha decidido incluir este ejercicio en el apéndice B.

8.5 Consideraciones finales.

El ejecutivo multitarea ha sido sometido a un conjunto de pruebas exhaustivo para certificar su correción. Las características funcionales se comprobaron mediante un conjunto de casos de prueba exprésamente diseñados [Puente&91b]. En este documento se han definido pruebas de la funcionalidad de los componentes individuales, pruebas de integración parcial y pruebas de integración global.

Con objeto de evaluar las características de ejecución del ejecutivo multitarea se le han aplicado las pruebas *hartstone* [Donhoe&90]. El objetivo de estas pruebas es comprobar la capacidad de un sistema para aplicaciones de tiempo real. Se analizan aspectos como la capacidad para ejecutar tareas con periódos pequeños, tareas con mucha carga de cómputo y un conjunto elevado de tareas. Las pruebas que se proporcionan libremente, no eran adecuada para el ejecutivo multitarea, por lo que hubo que adaptarlas. Los resultados [Crespo93] fueron satisfactorios, ya que no se detecto ningún fallo de ejecución y la carga de ejecución del ejecutivo es suficiente. La tabla siguiente muestra los resultados obtenidos con los experimentos con procesos periódicos armónicos. El resto de los resultados están descritos en [Crespo93].

Experimento	Carga (KWIPS	N. Tareas	Activaciones/seg
1	1539	5	366
2	1689	5	217
3	1812	5	93
4	1728	12	177

Estos resultados se han obtenido en un computador MVME-135, equipado con un procesador MOTOROLA 68020. En los experimentos, la unidad de carga es el *kilowhestone*, que en el entorno de ejecución vale 512 microsegundos. Las unidades de carga vienen dadas en KWIPS, que significa kilowhestone por segundo. La capacidad máxima de la placa es 1892 KWIPS.

Se han aprovechando los experimentos programados para calcular experimentalmente el retardo máximo que introduce la implementación de los procesos periódicos y que es aproximadamente de 500 microsegundos. Esta sobrecarga puede ser elevada en algunas aplicaciones. Sin embargo la funcionalidad añadida que se proporciona y la facilidad de desarrollo de aplicaciones de los componentes hacen interesante su empleo, si los requisitos temporales no son excesivamente pequeños en comporación a la sobrecarga.

Finálmente, para comprobar la viabilidad del ejecutivo para desarrollar aplicaciones, se le probó con las especificaciones de carga de una aplicación industrial de aviónica, con restricciones de tiempo real crítico. Esta aplicación estaba formada por 43 procesos periódicos con las siguientes características:

Número de procesos	Período (ms)	Tiempo de cómputo (ms)
1	20	1,04
4	40	1,57
5	40	2,09
1	40	2,63
2	40	3,68
1	40	4,2
2	80	1,04
1	80	1,57
1	80	2,63
1	160	1,57
9	320	1,04
3	320	1,57
1	320	2,09
1	320	2,63
6	640	1,04
1	640	1,57
2	640	2,09
1	640	3,15

El resultado de la aplicación del análisis de planificabilidad dió como resultado que el conjunto era planificable y con una ocupación del procesador del 99,866%. La ejecución del conjunto demostró la validez del análisis porque, pese a lo ajustado del cálculo, las tareas de tiempo real cumplieron sus plazos de respuesta.

El ejecutivo multitarea está formado por 14 componentes reutilizables. El número de líneas de código de los componentes del ejecutivo son aproximadamente 8000, a las que hay que añadir 50000 de las pruebas. En cuanto a la documentación desarrollada en relación al ejecutivo multitarea, destacan los siguientes documentos:

- Especificación de los componentes [Puente&91a].
- Manual de usuario del ejecutivo multitarea [Alonso&92c].
- Componentes reutilizables del ejecutivo multitarea [Alonso&91]. En este documento se incluye el código fuente de los componentes.
- Plan de pruebas del ejecutivo multitarea. [Puente&91b]

Capítulo 9

Conclusiones y líneas de trabajo futuro.

9.1 Conclusiones.

El objetivo de esta tesis es estudiar en profundidad los métodos de planificación de sistemas de tiempo real críticos basados en prioridades, detectar las cuestiones abiertas y desarrollar protocolos, esquemas de realización y directrices que faciliten su utilización en entornos industriales. Las aportaciones de este trabajo se han encuadrado en dos apartados que son el estudio de los mecanismos necesarios para desarrollar sistemas de tiempo real basados en prioridades y el reemplazamiento dinámico en sistemas de tiempo real crítico. El proyecto industrial *Biblioteca de Componentes Ada* ha permitido probar algunas de las ideas básicas sobre la realización de sistemas de tiempo real crítico. A continuación se profundiza en estas cuestiones.

Desarrollo de sistemas de tiempo real crítico.

La utilidad industrial de los métodos de planifación basados en prioridades depende de varios factores. Uno de ellos es su capacidad de satisfacer los requisitos funcionales de estas aplicaciones. Si bien el enunciado original del método no cumple esta premisa, los trabajos de investigación realizados últimamente han permitido, en gran medida, resolver este problema. Sin embargo, no se han publicado métodos de realización práctica de algunos de los protocolos desarrollados con núcleos normalizados.

En esta tesis se han desarrollado un conjunto de esquemas y directrices para la realización de tareas de tiempo real con el propósito de satisfacer los requisitos funcionales básicos de las aplicaciones industriales. Este trabajo permite desarrollar sistemas de tiempo real críticos con las siguientes características:

- Planificación basada en prioridades.
- Coexistencia de tareas periódicas, esporádicas y de segundo plano.
- Comunicación entre tareas.

- Comprobación analítica de la planificabilidad del sistema.
- Cambio del modo de ejecución del sistema.
- Detección de fallos de temporización.
- Tratamiento de fallos de ejecución.

Las tareas de tiempo real se han diseñado de forma que, además de seguir las pautas de activación relativas a su naturaleza, son capaces de detectar los incumplimiento de sus requisitos temporales. Para tal fin, se comprueba que cumplen su plazo de respuesta y que no emplean más tiempo de cómputo que el asignado.

El cambio de modo de ejecución permite que las tareas adapten su perfil a cambios previstos del sistema o de su entorno. Se ha diseñado un método de realización práctica de cambio de modo síncrono, en el que las tareas ejecutan el protocolo cuando están en estado consistente. Este método mantiene la coherencia de los datos de las tareas, asegura su ejecución de acuerdo al modo del sistema, garantiza el cumplimiento de los plazos durante y después de un cambio y no interfiere con la ejecución de las tareas cuyo perfil no se modifica al cambiar el modo del sistema.

En el enfoque adoptado, las tareas detectan por sí solas la ocurrencia del cambio de modo. Entonces cambian su perfil de ejecución y quedan a la espera de su siguiente activación. La mayor ventaja de esta alternativa es que no requiere ninguna funcionalidad especial del núcleo de ejecución. Sin embargo, en su realización se retarda el instante efectivo del cambio de modo, lo que en alguna circunstacia podría contravenir algún requisito de diseño. La realización fidedigna del modelo de ejecución del protocolo requiere dotar al núcleo de ejecución con funciones orientadas a la realización del protocolo o emplear tareas de alta prioridad para despertar al resto, lo que reduciría la capacidad de procesamiento disponible.

El tratamiento de fallos se ha basado en el concepto de grupo de recuperación. Está compuesto por:

- Un conjunto de tareas relacionadas funcionalmente, de forma que si una falla, el resto no puede ejecutarse correctamente. Estas tareas comparten las acciones de tratamiento de fallos de ejecución. Además, los fallos de ejecución de cada tarea se encapsulan para que no se transmitan incontroladamente a otras.
- Una tarea de recuperación, que incluye las acciones de tratamiento de fallos de las tareas del grupo y proporciona la funcionalidad básica del conjunto. Esta tarea permanecerá inicialmente inactiva. Cuando alguna tarea del grupo falla, se detiene la ejecución de todas ellas y se arranca la tarea de recuperación. Su estructura básica es la misma que las tareas de tiempo real.

El funcionamiento de los grupos de recuperación se inicia cuando se produce un fallo en una tarea. Entonces, el resto detecta este evento y termina de forma controlada, en un plazo acotado y sin interferir con la ejecución de tareas no implicadas en esta operación. A continuación la tarea de recuperación se activa.

Una de las características más destacables de los esquemas que se han desarrollado es la posibilidad de comprobar analiticamente la planificabilidad de un sistema desarrollado a partir de los mismos. Esto se debe a que en su realización se ha procurado no contravenir ninguna de las hipótesis básicas del método y, cuando ha habido que hacerlo, se han proporcionado alternativas y directrices para contemplar este caso en el análisis de planificabilidad.

Un resultado de este estudio ha sido la identificación de las características mínimas que debe proporcionar un sistema operativo para realizar aplicaciones de tiempo real críticas basadas en prioridades y con las características anteriormente enumeradas. Los requisitos funcionales identificados son:

- planificación basada en prioridades.
- el concepto de prioridad debe ser básico en la operación del sistema.
- comportamiento del sistema operativo determinista y tiempos de ejecución de servicios conocidos, acotados y pequeños.
- temporizadores absolutos con precisión suficiente.
- tratamiento de interrupciones y eventos asíncronos.
- mecanismos de comunicación pasivos, basados en el protocolo del techo de prioridad, o activos, mediante paso de mensajes.
- mecanismos básicos de tratamiento de fallos.
- temporizadores relativos al tiempo de cómputo de las tareas.
- transferencia asíncrona de control asociada a los temporizadores.
- prioridades dinámicas.

Reemplazamiento dinámico en sistemas de tiempo real crítico.

Una característica deseable de los sistemas de tiempo real es la capacidad de adaptarse dinámicanente a cambios del sistema. La posibilidad de cambiar el modo no es suficiente, pues no permite adaptar la ejecución del sistema a cambios imprevistos en el momento del diseño o a la existencia de módulos erróneos. Como consecuencia, en la vida útil de una aplicación suele ser necesario cambiar tareas o incluir algunas nuevas. Esta operación de mantenimiento se suele realizar deteniendo el sistema, cargando la nueva aplicación y reiniciando su ejecución. Sin embargo hay sistemas que no se pueden detener sin producir daños materiales o económicos, como ocurre con ciertas aplicaciones espaciales y de telecomunicaciones.

Una alternativa consiste en reemplazar dinámicamente partes del software sin necesidad de detener la aplicación. Si bien este problema se ha estudiado y se han propuesto soluciones en sistemas de tipo general, no hay alternativas para los sistemas de tiempo

real críticos. En este trabajo se ha diseñado un protocolo de reemplazamiento dinámico de software en sistemas de tiempo real críticos.

Un sistema con estas características se diseña como un conjunto de unidades reemplazables de software (URS), que en el método propuesto son tareas. La comunicación entre tareas se debe realizar por mecanismos indirectos, para que no haya compartición de memoria y para que las tareas estén completamente desacopladas. Las operaciones de reemplazamiento se realizan cuando las tareas están en estado consistente, que es al final o al comienzo de una activación.

En una operación de reemplazamiento están implicadas tres unidades: la unidad supervisora, que gestiona el reemplazamiento, la reemplazadora, que es la nueva URS, y la que va a ser reemplazada. Las fases del protocolo son:

- *Fase de Carga*: La unidad supervisora, que estaba bloqueada, recibe una orden externa de reemplazamiento. Entonces solicita espacio al núcleo y comienza a cargar la unidad reemplazadora. La carga se realiza en intervalos de tiempo de duración menor o igual al tiempo de cómputo asignado a la supervisora.

- *Fase de Creación*: La unidad supervisora crea la unidad reemplazadora. A continuación, se ejecuta su código inicial, con la prioridad de la supervisora del sistema. Posteriormente, la unidad reemplazadora queda en espera de recibir la información de estado. Antes de bloquearse, cambia su prioridad al valor de la unidad a reemplazar. Entonces la supervisora indica a la unidad a reemplazar que debe ejecutar el protocolo de reemplazamiento.

- *Fase de Intercambio*: La unidad a reemplazar manda información de estado a la unidad reemplazadora y comunica a la supervisora de reemplazamiento información sobre el resultado del intercambio. Si el estado se recibe satisfactoriamente, la unidad reemplazadora comienza a ejecutar a partir de la siguiente activación de la unidad reemplazada más una fase de inicio. La unidad reemplazadora se ejecuta con la misma prioridad que la reemplazada y emplea tiempo de cómputo incluido en el tiempo de cómputo de la segunda. Consecuentemente, si el intercambio se realiza sin errores, la unidad reemplazada termina. En caso contrario, se prepara para ejecutar la siguiente activación.

- *Fase de Limpieza*: La supervisora notifica al centro externo el resultado de la operación. Si la operación no ha sido correcta, se aborta a la unidad reemplazadora. Finalmente, la supervisora libera la memoria ocupada por la unidad y queda en disposición de recibir otra orden de reemplazamiento.

Se ha comprobado que este protocolo cumple la siguiente propiedad:

Sean $\mathcal{P} = \{\tau_1, \tau_2, \ldots, \tau_i, \ldots, \tau_n, Su\}$ y $\mathcal{P}' = \{\tau_1, \tau_2, \ldots, \tau'_i, \ldots, \tau_n, Su\}$ dos conjuntos de tareas. $\mathcal{P}'$ se obtiene reemplazando en $\mathcal{P}$, la tarea τ_i por τ'_i. Los conjuntos $\mathcal{P}$ y $\mathcal{P}'$ son planificables.

Si el reemplazamiento dinámico de τ_i por τ'_i se realiza según este protocolo, entonces el conjunto de tareas cumplirá sus plazos de respuesta durante y después de una operación de reemplazamiento.

Finalmente, se han desarrollado esquemas de realización de la URS y de la unidad supervisora. Se han intentado seguir los mismos principios de diseño que en el apartado anterior, sin embargo no ha sido totalmente posible porque los entornos de ejecución normalizados no proporcionan los servicios necesarios. Como consecuencia, estas funciones se han encapsulado en un paquete y se han completado los esquemas con el citado enfoque.

9.2 Líneas de trabajo futuro.

Los trabajos futuros en relación con esta tesis se pueden encuadrar en dos apartados. En primer lugar se presentaran los trabajos inmediatos a desarrollar con objeto de completar algunos aspectos de la tesis doctoral. Por otro lado, el desarrollo de esta tesis ha permitido tener una perspectiva del estado de la técnica suficiente para detectar las líneas de investigación a largo plazo más interesantes y estimulanes.

Relación de trabajos inmediatos.

Los trabajos de interés a corto plazo se describen a continuación:

- La primera labor consiste en la compilación y prueba de los esquemas de tareas presentados. En el momento del desarrollo de esta tesis, Ada 9X estaba en fase de normalización. Por tanto, es posible que aún se produzcan modificaciones en el lenguaje que impliquen revisar los esquemas. Además, la carencia de compiladores validados en este momento ha impedido ejecutar los esquemas de tareas. Así pues, es evidente el interés de probarlos exhaustivamente cuando se estabilice el lenguaje y se disponga de las herramientas necesarias.
- Las prestaciones de los esquemas de tareas constituyen un factor influyente en su aplicabilidad. El código básico de las tareas se ha ampliado con objeto de satisfacer los requisitos funcionales comentados, lo que supone una sobrecarga respecto a la funcionalidad específica de las tareas y puede impedir el uso de los esquemas en determinadas aplicaciones. Por todo ello, sería necesario determinar los tiempos de cómputo máximos de los esquemas para conocer con precisión la sobrecarga asociada con su utilización. Las pruebas realizadas con los componentes reutilizables en el proyecto *Biblioteca de Componentes Ada* permiten augurar que las prestaciones de los nuevos esquemas serán suficientes. Esto se debe a que Ada 9X proporciona alternativas más eficientes a algunos mecanismos empleados en los citados componentes.
- El lenguaje Ada 9X y los sistemas operativos basados en la norma POSIX serán muy empleados en el desarrollo de sistemas de tiempo real crítico en los próximos

años. Por esta razón, la adaptación de los esquemas de tareas a POSIX sería un ejercicio de gran interés práctico.

- En esta tesis se ha comprobado la compatibilidad del protocolo de reemplazamiento dinámico de software con los métodos de planificación basados en prioridades. A pesar de ello sería muy interesante probar su viabilidad práctica. En el desarrollo teórico de los protocolos se obvian determinados aspectos de bajo nivel, que pueden requerir leves modificaciones en su enunciado.

- Los esquemas de realización práctica del protocolo de reemplazamiento se pueden extender para dotar al sistema con cambio de modo y con tratamiento de fallos mediante grupos de recuperación. Adicionalmente, se puede ampliar esta funcionalidad con la posibilidad cambiar dinámicamente los perfiles de ejecución de las tareas, si es necesario al reemplazar una unidad, y con la posibilidad de restaurar la funcionalidad original de la aplicación si se ha producido un fallo y se está ejecutando la tarea de recuperación. En este caso, se reemplazaría la unidad errónea, se dentendría a la tarea de recuperación y se reiniciaría la ejecución de las tareas del grupo.

Líneas de trabajo futuro.

Los sistemas de tiempo real críticos y distribuidos que sean deterministas, fácilmente mantenibles, tolerantes a fallos y con metodologías y herramientas de apoyo que dirijan su desarrollo constituyen el objetivo a largo plazo de las comunidades científica e industrial. Es evidente, que para que este panorama sea real, es necesario que transcurra bastante tiempo y que se realize un gran esfuerzo de investigación. Análogamente, son muy diversos los temas de trabajo que deben ser desarrollados para obtener este objetivo.

A continuación, se enumeran las líneas de trabajo a largo plazo más relacionadas con esta tesis:

- El desarrollo de sistemas distribuidos requiere planificar un número mayor y más complejo de recursos. Además de los dispositivos de procesamiento, que serán múltiples, es necesario planificar el acceso a los medios de comunicación. Estos recursos se deben gestionar de forma integrada y para optimizar su uso y cumplir los requistos temporales del sistema. El desarrollo de métodos de planificación eficientes, deterministas y que gestionen globalmente los diversos recursos del sistema, constituye una actividad fundamental para la realización de los sistemas distribuidos de tiempo real complejos.

- En los sistemas centralizados, los requisitos temporales se suelen expresar mediante plazos de respuesta de las tareas. En los sistemas distribuidos este situación no es válida en general. Estos sistemas están compuesto por un conjunto de procesadores, que pueden ser heterogéneos y estar especializados en alguna actividad concreta, bien por su propia naturaleza (procesadores de señales digitales) bien porque estén conectados a dispositivos especiales. En esta situación, es común que una actividad

del sistema se lleve a cabo mediante la colaboración de un conjunto de tareas que pueden estar en distintos procesadores. Los requisitos temporales se suelen especificar como plazos de respuesta globales (*end-to-end deadlines*) de una actividad, es decir, al conjunto de tareas que la realizan. En esta circunstancia, sería interesante el estudio y desarrollo de métodos de planificación distribuida que gestionen estos plazos de forma integrada.

- El diseño de sistemas distribuidos de tiempo real eficaces y complejos requiere disponer de mecanismos de comunicación entre tareas que sean deterministas, fiables, compatibles con los métodos de planificación y que garantizen plazos de respuesta de envío de los mensajes suficientemente pequeños. Actualmente no se dispone de protocolos de comunicación de sistemas de tiempo real completamente satisfactorios. Los métodos basados en paso de testigo no dan suficiente soporte para que los nodos puedan solicitar el medio con diversas prioridades y cuando se pierde el testigo su comportamiento no es determinista. Por otro lado los métodos basados en redes *ethernet* no son, por definición, deterministas.

- La complejidad de los sistemas de tiempo real críticos aumenta constantemente. La única forma de tratar este problema es mediante metodologías de diseño específicas para este tipo de sistemas. La integración de los métodos de planificación con las metodologías de diseño permitiría extraer todas sus ventajas desde las primeras fases del ciclo de vida. Esta labor supondría, entre otras actividades, la provisión de directrices metodológicas para realizar los diversos componentes del sistema, abstracciones del método orientadas al desarrollo sistemas basados en prioridades, generación automática de código, etc..

- La visibilidad completa del sistema durante el desarrollo, es una característica que permitiría detectar errores desde las primeras fases del ciclo de vida. Para este fin, el prototipado incremental y heterogéneo constituye una poderosa herramienta. La integración de esta técnica con los métodos de planificación basados en prioridades es básica para su viabilidad. Para ello, sería, al menos, necesario disponer de mecanismos y prototipos prediseñados para representar el núcleo de ejecución del sistema, emular los dispositivos hardware y las operaciones de entrada/salida y contemplar las características específicas de los métodos de planificación empleados.

Apéndice A

Análisis detallado del protocolo de cambio de modo.

El objetivo de este apartado es presentar el análisis detallado de la ejecución de las tareas durante un cambio de modo, descrito en [Tindell&92a].

Es importante recordar que las ecuaciones que se presentan permiten analizar el comportamiento de las tareas durante la ejecución de un cambio de modo. El análisis de las tareas al terminar este estado transitorio es el descrito en los apartados 4.1.1, 4.2.3 y 4.3.6.

Para facilitar la lectura de este apéndice, se expone un glosario de los términos usados específicamente en este análisis:

- *Modo previo*: Modo del sistema vigente antes del cambio.
- *Modo nuevo*: Modo al que transita el sistemas al cambiar de modo.
- Un subíndice del tipo j se utiliza para referirse a parámetros del modo previo de una tarea.
- Un subíndice del tipo j' se utiliza para referirse a parámetros del modo nuevo de un tarea.
- W_i: Ventana de tiempo que incluye el tiempo de respuesta de la tarea τ_i en una activación.
- x: Tiempo que transcurre desde que se activa la tarea en estudio hasta que se produce un cambio de modo.
- OMV: Acrónimo que representa una tarea en el modo previo del sistema.
- NMV: Acrónimo que representa una tarea en el modo nuevo del sistema.
- WNT: Tarea nueva, que son las activas en el modo nuevo e inactivas en el modo previo.

- R_n: Tiempo que transcurre desde que se produce el cambio de modo hasta que se activa una tarea nueva.
- $C_j^{\uparrow i}$: Función que devuelve el valor del tiempo de cómputo de la tarea τ_j en el modo previo si su prioridad es mayor que la de τ_i.
- k_j: Su valor es $\left\lceil \frac{x}{T_j} \right\rceil$
- $\lceil y \rceil_0$: Función techo que devuelve 0 si y es negativo y el menor entero mayor que y en el resto de los casos.

El objetivo de este análisis es determinar el tiempo de respuesta más desfavorable de una tarea, cuando se produce un cambio de modo durante la ejecución de una activación o los tiempos de respuesta de la primera activación en un modo de una tarea. La caracterización completa de estos parámetros implica estudiar los siguientes casos:

1. Tiempo de respuesta de una tarea, τ_i, que se activa en el modo previo y se completa en el nuevo.
2. Tiempo de respuesta de la primera activación de una tarea en el modo nuevo.
3. Tiempo de respuesta de la primera activación de una tarea en el modo nuevo, estando inactiva en el previo.

Tarea activada en el modo previo

La interferencia máxima que sufre una tarea τ_i cuando se produce un cambio de modo durante su activación, está compuesta por dos efectos:

- Interferencia de las tareas activas en los dos modos
- Interferencia de las tareas nuevas.

Interferencia de tareas activas en los dos modos. Para calcular la interferencia de tareas activas en ambos modos, se calcula, en primer lugar, la interferencia de una tarea, τ_j, y, a continuación se extiende el análisis al conjunto de tareas. El estudio de esta interferencia se divide en cuatro casos, según sea la prioridad de la tarea en el modo previo y en el nuevo.

1. $pr_i > pr_j$ y $pr_i > pr_{j'}$: Como es obvio, en este caso la interferencia es nula.
2. $pr_i > pr_j$ y $pr_i < pr_{j'}$: La interferencia está determinada por el número máximo de activaciones de $\tau_{j'}$ en el intervalo $[x, W_i)$:

$$\left\lceil \frac{W_i - x}{T_j{}'} \right\rceil_0 C_j{}' \qquad \text{(A.1)}$$

3. $pr_i < pr_j$ y $pr_i > pr_{j'}$: La interferencia está determinada por el número máximo de activaciones de τ_j en el intervalo $[k.T_i, W_i)$:

$$\left\lceil \frac{x}{T_j} \right\rceil_0 C_j \tag{A.2}$$

4. $pr_i < pr_j$ y $pr_i < pr_{j'}$: Intuitivamente, la interferencia en este caso sería la suma de los dos anteriores. Sin embargo, este cálculo es pesimista. Por lo que se determina un cálculo preciso. En cualquier caso, este enfoque es una condición suficiente para el cálculo de la planificabilidad de una tarea durante un cambio de modo. Como este cálculo es más sencillo que el que se va a presentar a continuación, parece adecuado realizarle en primer lugar, y sólo hacer el análisis más preciso en caso de que la tarea fuera no planificable.

   La interferencia mayor que sufre una tarea τ_i por una tarea τ_i ocurre, con las características anteriores, acontece en una de las dos situaciones siguientes [Tindell&92a]:

   (a) τ_i y τ_j se activan al mismo tiempo (figura A.1).

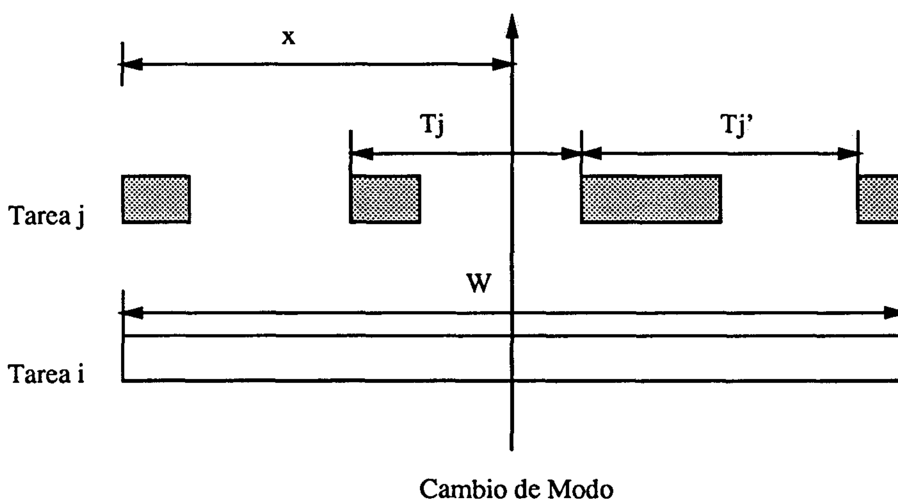

Figura A.1: τ_i y τ_j comparten un instante crítico.

La interferencia es la suma de la interferencia de la tarea en el modo previo y en el modo nuevo:

$$\left\lceil \frac{x}{T_j} \right\rceil C_j + \left\lceil \frac{W_i - \left\lceil \frac{x}{T_j} \right\rceil T_j}{T_{j'}} \right\rceil C_{j'} \tag{A.3}$$

Esta ecuación se puede reescribir como:

$$k_j C_j + \left\lceil \frac{W_i - k_j T_j}{T_{j'}} \right\rceil C_{j'} \tag{A.4}$$

Donde, $k_j = \left\lceil \frac{x}{T_j} \right\rceil$

(b) En el modo nuevo τ_j se activa inmediatamente después del cambio de modo (figura A.2). En este caso la interferencia total es:

$$\left\lfloor \frac{x}{T_j} \right\rfloor C_j + \left\lceil \frac{W_i - x}{T_{j'}} \right\rceil C_{j'} \tag{A.5}$$

Intuitivamente puede parecer que al primer sumando se le debería aplicar la función techo en lugar de la función suelo. Posteriormente se justificará esta elección.

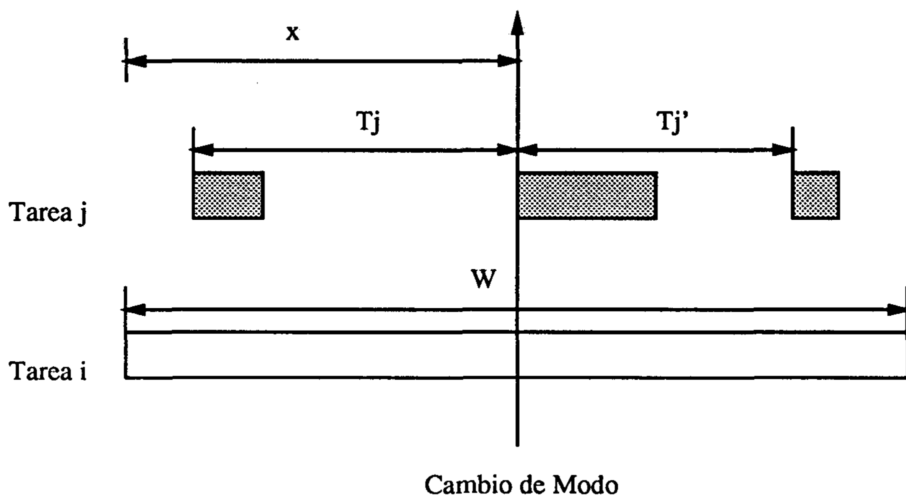

Figura A.2: τ_j se activa inmediatamente después que el cambio de modo

Consecuentemente la interferencia mayor de la tarea τ_j en la ejecución de τ_i se determina por el máximo de las ecuaciones A.4 y A.5.

Interferencia de tareas nuevas. La interferencia que una tarea τ_n sobre τ_i viene dada por:

$$\left\lceil \frac{W_i - R_n - x}{T_n} \right\rceil_0 C_n \tag{A.6}$$

Tiempo de respuesta máximo. En los apartados anteriores, se ha calculado la interferencia más desfavorable un tipo de tarea. Consecuentemente, para calcular el caso más desfavorable habrá que sumar la interferencia del conjunto completo de tareas.

$$W_i(x) = \sum_{j \in omv} \max\left(\left\lfloor \frac{x}{T_j} \right\rfloor C_j^{\uparrow i} + \left\lceil \frac{W_i(x) - x}{T_{j'}} \right\rceil C_{j'}^{\uparrow i} ,\quad k_j C_j^{\uparrow i} + \left\lceil \frac{W_i(x) - k_j T_j}{T_{j'}} \right\rceil C_{j'}^{\uparrow i} \right) \tag{A.7}$$

$$\sum_{n \in wnt} \left\lceil \frac{W_i(x) - R_n - x}{T_n} \right\rceil_0 C_n^{\uparrow i} + C_i + B_i$$

La ecuación presentada es recurrente, ya que el término W_i aparece a la derecha y a la izquierda de la expresión. Además, es sencillo comprobar que $W_i^n(x) \leq W_i^{n+1}(x)$. Por consiguiente, la comprobación de la planificabilidad se inicia con el valor $W_0 = 0$ y se itera con los sucesivos valores de la ventana, hasta que el proceso converge, $W_i^n(x) = W_i^{n+1}(x)$, o el plazo de respuesta no se cumple, $W_i^n(x) > D_i$. La ecuación anterior queda:

$$W_i^{l+1}(x) = \sum_{j \in omv} \max \left(\left\lfloor \frac{x}{T_j} \right\rfloor C_j^{\uparrow i} + \left\lceil \frac{W_i^l(x) - x}{T_{j'}} \right\rceil C_{j'}^{\uparrow i}, \quad k_j C_j^{\uparrow i} + \left\lceil \frac{W_i^l(x) - k_j T_j}{T_{j'}} \right\rceil C_j^{\uparrow i} \right) \sum_{n \in wnt} \left\lceil \frac{W_i^l(x) - R_n - x}{T_n} \right\rceil_0 C_n^{\uparrow i} + C_i + B_i \quad \text{(A.8)}$$

Es ahora el momento de justificar la selección de la función suelo en el término derecho de la ecuación A.5. Si la tarea τ_j se activara δ unidades de tiempo antes que τ_i entonces la activación de τ_j al mismo tiempo que τ_i debería incrementar el impacto computacional al menos en δ. Así pues en la siguiente iteración en el cálculo de W_i, el caso más desfavorable parcial sería la parte derecha de la ecuación A.5, lo que tiene en cuenta por completo del cómputo de τ_j incluido en el primer tiempo x de W_i.

Finalmente, como el valor de W_i es una función de x, por lo que el tiempo de respuesta más desfavorable vendrá dado por:

$$r_i = \max_{\forall x}[W_i(x)] \quad \text{(A.9)}$$

La evaluación de esta función para todos los valores de x resultaría en que muchas veces se obtendría el mismo valor. Por tanto, hay que determinar un conjunto de valores de x significativos. Estos valores significativos son aquellos que significan valores diferentes de k_j y que maximizan la carga de tareas nuevas. El conjunto de valores de x es:

$$x \in \{0, \epsilon, T_j + \epsilon, 2T_j + \epsilon, \ldots\} \quad \text{(A.10)}$$

donde ϵ es el menor incremento significativo del reloj del sistema.

Primera activación de una tarea en el modo nuevo.

La necesidad de caracterizar este caso se debe a la posibilidad de que determinadas tareas activadas antes del cambio de modo interfiera con la primera ejecución de una tarea en el mono nuevo . El tiempo de respuesta más desfavorable acontece en una de las situaciones siguientes:

1. Tiempo de respuesta de la tarea, τ_i, en el caso normal, es decir, en ausencia de cambio de modo.

2. Tiempo de respuesta de la tarea, τ_i, cuando se activa inmediatamente después de producirse el cambio de modo.

3. Tiempo de respuesta de la tarea, τ_i, cuando se activa después de producirse el cambio de modo.

Ejecución normal de la tarea. El tiempo de respuesta más desfavorable de la tarea, τ_i, en ausencia de cambio de modo, se calcula mediante la ecuación 4.2.3.

La tarea se activa inmediatamente después del cambio de modo. El tiempo de respuesta en este caso estará contenido en una ventaba que se denomina WMC_i. La interferencia mayor de una tarea, τ_j, activa en el modo previo, sobre la primera activación de una tarea τ_i en el modo nuevo se produce en uno de los siguientes casos:

1. τ_j en el modo previo se activa inmediatamente antes del cambio de modo. En este caso, la interferencia es:

$$C_j^{\uparrow i} + \left\lceil \frac{WMC_i - T_j}{T_{j'}} \right\rceil C_{j'}^{\uparrow i} \tag{A.11}$$

2. τ_j en el modo nuevo se activa inmediatamente antes del cambio de modo. La interferencia es:

$$\left\lceil \frac{WMC_i}{T_{j'}} \right\rceil C_{j'}^{\uparrow i} \tag{A.12}$$

Habrá que añadir el efecto producido por las tareas nuevas, que no estaban activas en el modo anterior.

Consecuentemente, el mayor tiempo de respuesta en este caso viene dado por:

$$\begin{aligned} r_i^{MC} = WMC_i &= C_i' + \sum_{j \in omv} \max \left(C_j^{\uparrow i} + \left\lceil \frac{WMC_i - T_j}{T_{j'}} \right\rceil C_{j'}^{\uparrow i}, \left\lceil \frac{WMC_i}{T_{j'}} \right\rceil C_{j'}^{\uparrow i} \right) \\ &\sum_{n \in wnt} \left\lceil \frac{WMC_i - R_n}{T_n} \right\rceil C_n^{\uparrow i} \end{aligned} \tag{A.13}$$

La tarea se activa después del cambio de modo. Para calcular este caso, se considera una ventana, $WOMV$ que comienza en la última activación τ_i en el modo previo. En esta ventana se incluye los tiempos de cómputo de las dos versiones de la tarea y la interferencia máxima del resto de las tareas. Este valor viene dado por:

$$\begin{aligned} WOMV_i^{l+1}(x) &= C_{i'} + \sum_{n \in nmt} \left\lfloor \frac{WOMV_i^l - R_n - x}{T_n} \right\rfloor C_n^{\uparrow i} + \\ &+ \sum_{n \in omv} \max \left(\left\lfloor \frac{x}{T_j} \right\rfloor C_j^{\uparrow i} + \left\lceil \frac{WOMV_i^l(x) - x}{T_{j'}} \right\rceil_0 C_{j'}^{\uparrow i}, \; k_j C_j^{\uparrow i} + \left\lceil \frac{WOMV_i^l(x) - k_j T}{T_{j'}} \right. \end{aligned}$$

Aunque la tarea τ_i se incluye en el conjunto OMV, el valor $C_{i'}$ no se considera por duplicado. Esta ecuación se debe calcular para los valores de x del conjunto definido en la ecuación A.10. Además, el tiempo de respuesta más desfavorable de la primera activación de la tarea en el modo nuevo, se debe calcular precisamente desde el momento de la activación. Por consiguiente, habrá que descontar de la ventana el valor del período de la tarea del modo previo.

$$r_i^{OMV} = \max_{\forall x}[WOMV_i(x)] - T_i \tag{A.15}$$

Tiempo de respuesta máximo. El tiempo de respuesta más desfavorable de la primera activación de de una tarea en el modo nuevo , τ_i, es el máximo de los anteriores, es decir:

$$r_{i'} = \max(r^n, r_i^{MC}, r_i^{OMV}) \tag{A.16}$$

Primera activación de una tarea nueva.

Al igual que el caso anterior, es necesario estudiar este caso por la posibilidad de que determinadas tareas activadas en el modo previo, interfieran con la primera ejecución de una tarea nueva. El tiempo de respuesta es el mayor entre el valor en ausencia de cambio de modo, r_n, y el incluido en una ventana que comienza cuando se produce el cambio de modo, que coincide con el valor calculado en la ecuación A.13. Sin embargo, una tarea nueva puede activarse R_i instantes después del cambio de modo y, por tanto, este valor hay que descontarle de la ventana calculada. Consecuentemente, el tiempo de respuesta mayor en la primera activación de una tarea nueva es:

$$r_n = \max(r^n, r_i^{MC} - R_i) \tag{A.17}$$

Apéndice B

Desarrollo de sistemas con el ejecutivo multitarea

En este capítulo se va a ilustrar el uso de los componentes del ejecutivo multitarea para desarrollar la aplicación ejemplo de la sección 8.4.2. El uso de los componentes se muestra creando un objeto de cada tipo, en un orden consistente.

Ha sido imposible incluir en este documento toda la información referente a los parámetros y componentes que se emplean en este capítulo. Se espera que la información que se proporcionar sea suficiente para dar al lector una idea clara de las características principales de los componentes y la forma de emplearlos. En el documento [Alonso&92c] se encuentra toda la información necesaria para el empleo de estos componentes.

B.1 Modos

La definición del tipo con los modos de ejecución del sistema se debe hacer en una sóla ocasión y se debe usar en todos los componentes. A continuación se muestra un ejemplo de definición con los tres modos de la aplicación ejemplo:

```
package Execution_Modes_Definition is

   Execution_Modes is (Normal_Mode, Critical_Mode, Alarm_Mode);

end Execution_Modes_Definition;
```

B.2 Perfiles de ejecución

Es necesario definir el conjunto de tipos de datos relacionados con los perfiles de ejecución. Para tal fin se utiliza el componente genérico `Execution_Profiles`. Este componente se debe instanciar una sola vez en una aplicación y ser utilizado por el resto de los componentes que lo necesiten. A continuación, se muestra como se instanciaría este paquete a partir de la declaración de modos del sistema:

El valor de `Memory_Size` debe ser suficiente para almacenar los datos del componente.

Finalmente hay que proporcionar el parámetro que indique que se debe hacer en caso de fallo del proceso de recuperación. Las opciones posibles son parar el sistema o parar el proceso. En este caso se ha elegido la primera opción.

```
--/*****************************************************************
--/ package specification Recovery_Process_Package_1
--/*****************************************************************

with Application_Execution_Profiles;
package Recovery_Process_Package_1 is

   function Error_Happened return Boolean;

   procedure Signal_Error;

   procedure Define_Mode_Delays_Table
            (Mode_Delay : in Application_Execution_Profiles.Mode_Delays);

end Recovery_Process_Package_1;

--/*****************************************************************
--/ package body Recovery_Process_1
--/*****************************************************************
with Recovery_Process;
with Hardware_Interface;
with Execution_Modes_Definition;
with Application_Supervisor;
with Run_Time_System_Interface;

package body Recovery_Process_Package_1 is

   Maximum_Size : Hardware_Interface.Hw_Long_Integer := 2048;
   Fail_Action  : Run_Time_System_Interface.Failure_Action_Type
                :=Run_Time_System_Interface.Stop_System;

   function Current_Local_Mode
      return Execution_Modes_Definition.Mode_Type;

   --/*********************************************************
   --/ procedure Process_Activity
   --/*********************************************************

     %. . .%

   procedure Process_Activity is
     %. . .%
   begin
     %. . .%
   end Process_Activity;
```

```
--/*********************************************************
--/ package specification Profiles_Declaration
--/*********************************************************
package Profiles_Declaration is

   Process_Mode_Profiles :
    Application_Execution_Profiles.Periodic_Mode_Profile;

end Profiles_Declaration;

--/*********************************************************
--/ package body Profiles_Declaration
--/*********************************************************

package body Profiles_Declaration is
   use  Application_Execution_Profiles;

begin

   Process_Mode_Profiles := Set_Periodic_Mode_Profile
    (Mode_Profile => Process_Mode_Profiles,
     Mode         => Execution_Modes_Definition.Normal_Mode,
     Profile      => Make_Periodic_Profile
                          (Period   => 0.300
                           Phase    => 0,
                           Priority => 94));

   Process_Mode_Profiles := Set_Periodic_Mode_Profile
    (Mode_Profile => Process_Mode_Profiles,
     Mode         => Execution_Modes_Definition.Critical_Mode,
     Profile      => Make_Periodic_Profile
                          (Period   => 0.300,
                           Phase    => 0,
                           Priority => 90));

   Process_Mode_Profiles := Set_Periodic_Mode_Profile
    (Mode_Profile => Process_Mode_Profiles,
     Mode         => Execution_Modes_Definition.Alarm_Mode,
     Profile      => Make_Periodic_Profile
                          (Period   => 0.300,
                           Phase    => 0,
                           Priority => 92));


end Profiles_Declaration;

--/*********************************************************
--/ package Recovery_Process_1
--/*********************************************************
package Recovery_Process_1 is new Recovery_Process
(
-- Application parameters
Periodic_Profile_Type               => Application_Execution_Profiles.Periodic_Profile,
```

```
   Is_Inactive_Periodic_Profile      => Application_Execution_Profiles.Is_Inactive_Periodic_Prof
   Get_Period                        => Application_Execution_Profiles.Get_Period,
   Get_Phase                         => Application_Execution_Profiles.Get_Phase,
   Get_Priority                      => Application_Execution_Profiles.Get_Priority,
   Mode_Type                         => Execution_Modes_Definition.Execution_Modes,
   Periodic_Mode_Profile_Type        => Application_Execution_Profiles.Periodic_Mode_Profile,
   Get_Periodic_Profile              => Application_Execution_Profiles.Get_Periodic_Profile,
   Mode_Delays_Type                  => Application_Execution_Profiles.Mode_Delays,
   Set_Mode_Delay                    => Application_Execution_Profiles.Set_Mode_Delay,
   Get_Mode_Delay                    => Application_Execution_Profiles.Get_Mode_Delay,
   Current_System_Mode               => Application_Supervisor.Current_System_Mode,
   Report_to_System_Supervisor_Mode_Delays => Application_Supervisor.Define_Mode_Delays_Table,
   Sleep                             => Application_Supervisor.Sleep,
   Get_Mode_Change_Status            => Application_Supervisor.Get_Mode_Change_Status,
   Stop_System                       => Application_Supervisor.Stop_System,

   -- Process parameters
   Activity                          => Process_Activity,
   Maximum_Size                      => Maximum_Size,
   Periodic_Mode_Profiles            => Profiles_Declaration.Process_Mode_Profiles,
   Fail_Action                       => Fail_Action
);


   function Error_Happened return Boolean is
   begin
      return Recovery_Process_1.Error_Happened;
   end Error_Happened;


   procedure Signal_Error is
   begin
      Recovery_Process_1.Signal_Error;
   end Signal_Error;


   procedure Define_Mode_Delays_Table
        (Mode_Delay : in Application_Execution_Profiles.Mode_Delays) is
   begin
      Recovery_Process_1.Accept_Mode_Delays(Mode_Delay);
   end Define_Mode_Delays_Table;


   function Current_Local_Mode return Execution_Modes_Definition.Execution_Modes is
   begin
      return Recovery_Process_1.Current_Local_Mode;
   end Current_Local_Mode;

end Recovery_Process_Package_1;
```

B.5 Procesos Periódicos

Ahora se definen los procesos periódicos. Para ello se usa componente Periodic_Process. Para facilitar la labor de creación del proceso, se ha desarrollado una plantilla, Periodic_Process_Shell que se emplea en este apartado.

En este ejemplo se muestra la creación del proceso periódico denominado anteriormente PP_1 y que el nombre que se le asigna en la aplicación es Periodic_Process_1. La de los otros procesos es análoga.

Como el proceso periódico pertenece al grupo de recuperación del proceso de recuperación PR_1, las parámetros actuales del grupo de interacción con éste se usan los procedimientos exportados por el paquete Recovery_Process_Package_1. Los procesos periódicos del grupo de PR_2 harán lo propio con Recovery_Process_Package_2.

El procedimiento con la actividad del proceso se debe desarrollar siguiendo las normas de ejecución del método de planificación.

El valor de Memory_Size debe ser suficiente para almacenar los datos del componente.

```
--/*****************************************************************
--/ package specification Periodic_Process_Package_1
--/*****************************************************************

package Periodic_Process_Package_1 is

end Periodic_Process_Package_1;

--/*****************************************************************
--/ package body Periodic_Process_Package_1
--/*****************************************************************
with Periodic_Process;
with Hardware_Interface;
with Execution_Modes_Definition;
with Application_Execution_Profiles;
with Application_Supervisor;
with Recovery_Process_Package_1;

package body Periodic_Process_Package_1 is

   Maximum_Size : Hardware_Interface.Hw_Long_Integer := 1024;

   function Current_Local_Mode return Execution_Modes_Definition.Mode_Type;

   --/***********************************************************
   --/ procedure Process_Activity
   --/***********************************************************
      %. . .%
   procedure Process_Activity is
      %. . .%
   begin
      %. . .%
   end Process_Activity;
```

```
--/*************************************************************
--/ package specification Profiles_Declaration
--/*************************************************************

package Profiles_Declaration is

   Process_Mode_Profiles : Application_Execution_Profiles.Periodic_Mode_Profile;

end Profiles_Declaration;

--/*************************************************************
--/ package body Profiles_Declaration
--/*************************************************************

package body Profiles_Declaration is
   use Application_Execution_Profiles;

begin

   Process_Mode_Profiles := Set_Periodic_Mode_Profile
    (Mode_Profile => Process_Mode_Profiles,
     Mode         => Execution_Modes_Definition.Normal_Mode,
     Profile      => Make_Periodic_Profile
                          (Period   => 0.400
                           Phase    => 0,
                           Priority => 93));

   Process_Mode_Profiles := Set_Periodic_Mode_Profile
    (Mode_Profile => Process_Mode_Profiles,
     Mode         => Execution_Modes_Definition.Critical_Mode,
     Profile      => Make_Periodic_Profile
                          (Period   => 0.200,
                           Phase    => 0,
                           Priority => 96));

   Process_Mode_Profiles := Set_Periodic_Mode_Profile
    (Mode_Profile => Process_Mode_Profiles,
     Mode         => Execution_Modes_Definition.Alarm_Mode,
     Profile      => Make_Periodic_Profile
                          (Period   => 0.400,
                           Phase    => 0,
                           Priority => 91));


end Profiles_Declaration;

--/*************************************************************
--/ package Periodic_Process_1
--/*************************************************************
package Periodic_Process_1 is new Periodic_Process
(
```

```
   -- Application parameters
   Periodic_Profile_Type              => Application_Execution_Profiles.Periodic_Profile,
   Is_Inactive_Periodic_Profile       => Application_Execution_Profiles.Is_Inactive_Periodic_Pr
   Get_Period                         => Application_Execution_Profiles.Get_Period,
   Get_Phase                          => Application_Execution_Profiles.Get_Phase,
   Get_Priority                       => Application_Execution_Profiles.Get_Priority,
   Mode_Type                          => Execution_Modes_Definition.Execution_Modes,
   Periodic_Mode_Profile_Type         => Application_Execution_Profiles.Periodic_Mode_Profile,
   Get_Periodic_Profile               => Application_Execution_Profiles.Get_Periodic_Profile,
   Mode_Delays_Type                   => Application_Execution_Profiles.Mode_Delays,
   Set_Mode_Delay                     => Application_Execution_Profiles.Set_Mode_Delay,
   Initial_Activation_Time            => Application_Supervisor.Initial_Activation_Time,
   Current_System_Mode                => Application_Supervisor.Current_System_Mode,
   Report_to_System_Supervisor_Mode_Delays => Application_Supervisor.Define_Mode_Delays_Table,
   Report_to_Recovery_Process_Mode_Delays  => Recovery_Process_Package_1.Define_Mode_Delays_Ta
   Sleep                              => Application_Supervisor.Sleep,
   Get_Mode_Change_Status             => Application_Supervisor.Get_Mode_Change_Status,

   -- Process parameters
   Activity                           => Process_Activity,
   Maximum_Size                       => Maximum_Size,
   Periodic_Mode_Profiles             => Profiles_Declaration.Process_Mode_Profiles,
   Check_Recovery_Error               => Recovery_Process_Package_1.Error_Happened,
   Report_Recovery_Error              => Recovery_Process_Package_1.Signal_Error);

   --/*********************************************************
   --/ function Current_Local_Mode
   --/*********************************************************
   function Current_Local_Mode
      return Execution_Modes_Definition.Mode_Type is

   begin
      return Periodic_Process_1.Current_Local_Mode;

   end Current_Local_Mode;

end Periodic_Process_Package_1;
```

B.6 Procesos Esporádicos

Para definir los procesos esporádicos se usa componente `Sporadic_Server_Process`. Este proceso necesita una serie de definiciones. Para facilitar esta labor, se ha desarrollado una plantilla, `Sporadic_Server_Process_Shell` que se emplea en este apartado.

En este ejemplo se muestra la creación del proceso periódico denominado anteriormente PE_1 y que el nombre que se le asigna en la aplicación es `Sporadic_Server_Process_1`. La del otro proceso es análoga. Respecto a los parámetros formales relacionados con la interacción con el proceso de recuperación, se aplican las mismas consideraciones que en el apartado anterior. Como pertenece al grupo de recuperación de PR_1, se usan

los procedimientos exportados por el paquete Recovery_Process_Package_1.

El procedimiento con la actividad del proceso se debe desarrollar siguiendo las normas de ejecución del método de planificación.

El valor de Memory_Size debe ser suficiente para almacenar los datos del componente.

El tipo Aperiodic_Event_Type permite al usuario determinar un tipo de datos para representar información asociado al evento aperiódico. En este caso se va a suponer que la información de interés que se obtiene después de la interrupción al procesador es una palabra de 8 bits, que se representa con el tipo Hardware_Interface.Hw_Byte.

```
--/*******************************************************************
--/ package specification Sporadic_Server_Process_Package_1
--/*******************************************************************
with Calendar;
with Hardware_Interface;

package Sporadic_Server_Process_Package_1 is

   type Aperiodic_Event_Type is Hardware_Interface.Hw_Byte

   procedure Signal_Aperiodic_Event
             (Event_Information : in Aperiodic_Event_Type;
              Event_Time        : in Calendar.Time);

end Sporadic_Server_Process_Package_1;

--/*******************************************************************
--/ package body Sporadic_Server_Process_Package_1
--/*******************************************************************
with Sporadic_Server_Process;
with Execution_Modes_Definition;
with Execution_Modes_Definition;
with Application_Supervisor;
with Recovery_Process_Package_1;

package body Sporadic_Server_Process_Package_1 is

   Maximum_Size : Hardware_Interface.Hw_Long_Integer := $Memory_Size$;

   function Current_Local_Mode
      return Execution_Modes_Definition.Execution_Modes;

   --/*************************************************************
   --/ procedure Process_Activity
   --/*************************************************************
      %. . .%
   procedure Process_Activity
           (The_Event_Information : in Aperiodic_Event_Type) is
      %. . .%
   begin
      %. . .%
   end Process_Activity;
```

```
--/*********************************************************
--/ package specification Profiles_Declaration
--/*********************************************************
package Profiles_Declaration is

   Process_Mode_Profiles : Execution_Modes_Definition.Sporadic_Mode_Profile;

end Profiles_Declaration;

--/*********************************************************
--/ package body Profiles_Declaration
--/*********************************************************

use  Execution_Modes_Definition;

package body Profiles_Declaration is

begin

   Process_Mode_Profiles := Set_Sporadic_Mode_Profile
    (Mode_Profile => Process_Mode_Profiles,
     Mode         => Execution_Modes_Definition.Normal_Mode,
     Profile      => Make_Sporadic_Profile
          (Inter_Arrival_Time => 0.500,
           Deadline           => 0.300,
           Priority           => 96));

   Process_Mode_Profiles := Set_Sporadic_Mode_Profile
    (Mode_Profile => Process_Mode_Profiles,
     Mode         => Execution_Modes_Definition.Critical_Mode,
     Profile      => Make_Sporadic_Profile
          (Inter_Arrival_Time => 0.500,
           Deadline           => 0.300,
           Priority           => 91);

   Process_Mode_Profiles := Set_Sporadic_Mode_Profile
    (Mode_Profile => Process_Mode_Profiles,
     Mode         => Execution_Modes_Definition.Alarm_Mode,
     Profile      => Make_Sporadic_Profile
          (Inter_Arrival_Time => 0.500,
           Deadline           => 0.300,
           Priority           => 93));

end Profiles_Declaration;


--/*********************************************************
--/ package Sporadic_Server_Process_1
--/*********************************************************
package Sporadic_Server_Process_1 is new Sporadic_Server_Process
(
```

```
   -- Application parameters
   Sporadic_Profile_Type             => Execution_Modes_Definition.Sporadic_Profile,
   Is_Inactive_Sporadic_Profile      => Execution_Modes_Definition.Is_Inactive_Sporadic_Profile,
   Get_Inter_Arrival_Time            => Execution_Modes_Definition.Get_Inter_Arrival_Time,
   Get_Deadline                      => Execution_Modes_Definition.Get_Deadline,
   Get_Priority                      => Execution_Modes_Definition.Get_Priority,
   Mode_Type                         => Execution_Modes_Definition.Execution_Modes,
   Sporadic_Mode_Profile_Type        => Execution_Modes_Definition.Sporadic_Mode_Profile,
   Get_Sporadic_Profile              => Execution_Modes_Definition.Get_Sporadic_Profile,
   Mode_Delays_Type                  => Execution_Modes_Definition.Mode_Delays,
   Set_Mode_Delay                    => Execution_Modes_Definition.Set_Mode_Delay,
   Initial_Activation_Time           => Application_Supervisor.Initial_Activation_Time,
   Current_System_Mode               => Application_Supervisor.Current_System_Mode,
  Report_to_System_Supervisor_Mode_Delays=> Application_Supervisor.Define_Mode_Delays_Table,
   Report_to_Recovery_Process_Mode_Delays  => Recovery_Process_Package_1.Define_Mode_Delays_Tabl
   Sleep                             => Application_Supervisor.Sleep,
   Get_Mode_Change_Status            => Application_Supervisor.Get_Mode_Change_Status,
   Aperiodic_Event_Type              => Aperiodic_Event_Type,

   -- Process parameters
   Activity                          => Process_Activity,
   Maximum_Size                      => Maximum_Size,
   Sporadic_Mode_Profiles            => Profiles_Declaration.Process_Mode_Profiles,
   Check_Recovery_Error              => Recovery_Process_Package_1.Error_Happened,
   Report_Recovery_Error             => Recovery_Process_Package_1.Signal_Error
);

   --/*********************************************************
   --/ procedure Signal_Aperiodic_Event
   --/*********************************************************
   procedure Signal_Aperiodic_Event
       (Event_Information : in Aperiodic_Event_Type;
        Event_Time        : in Calendar.Time) is

   begin
      Sporadic_Server_Process_1.Signal_Aperiodic_Event
         (Event_Information,
          Event_Time);
   end Signal_Aperiodic_Event;


   --/*********************************************************
   --/ procedure Current_Local_Mode
   --/*********************************************************
   function Current_Local_Mode return Execution_Modes_Definition.Execution_Modes is

   begin
      return Sporadic_Server_Process_1.Current_Local_Mode;
   end Current_Local_Mode;

end Sporadic_Server_Process_Package_1;
```

B.7 Monitores

Los monitores se crean con la plantilla `Monitor_Shell`. Un monitor exporta una serie de subprogramas mediante los cuales se comunican los procesos de tiempo real. Estos procedimientos son, en realidad, llamadas a puntos de entrada de una tarea, para garantizar exclusión mútua.

Desde el punto de vista del ejecutivo, es importante la prioridad que se le asigna al monitor. En la plantilla, se asigna el valor de la prioridad a la variable `Monitor_Priority`. El ejemplo que se adjunta corresponde a Mo_1, por lo que su prioridad es 97. Este monitor exporta dos subprogramas, uno lee un dato, `Read_Value` y el otro lo escribe, `Write_Value`.

En el volumen de tipos abstractos de datos se proporciona un monitor que controla el acceso de un conjunto de procesos a datos compartidos. Aunque este componente no está diseñado de acuerdo a las necesidades específicas del ejecutivo multitarea, sirve como ejemplo orientativo de la funcionalidad de un monitor.

```
--/*********************************************************************
--/ package specification Monitor_1
--/*********************************************************************

package Monitor_1 is

  procedure Write_Data (Value : Hardware_Interface.Hw_Integer);

  function Read_Data return Hardware_Interface.Hw_Integer;

end Monitor_1;

--/*********************************************************************
--/ package body Monitor_1
--/*********************************************************************

with Hardware_Interface;
with Run_Time_System_Interface;

package body Monitor_1 is

   task Monitor_Task is

      entry Write_Data (A_Value : in  Hardware_Interface.Hw_Integer);

      entry Read_Data  (A_Value : out Hardware_Interface.Hw_Integer);

   end Monitor_Task;


  procedure Write_Data (Value : in Hardware_Interface.Hw_Integer) is
   begin
      Monitor_Task.Write_Data (Value);
   end Write_Data;
```

```
  function Read_Data return Hardware_Interface.Hw_Integer is
       Value : Hardware_Interface.Hw_Integer;
   begin
      Monitor_Task.Read_Data (Value);
      return Value;
   end Read_Data;

   --/*****************************************************************
   --/ task Monitor_Task
   --/*****************************************************************

   task body Monitor_Task is

      Monitor_Priority : Run_Time_System_Interface.Priority_Type := 97;

   begin

     Run_Time_System_Interface.Set_Priority(Monitor_Priority);

     loop

      select

            accept Write_Data (A_Value : in  Hardware_Interface.Hw_Integer)is
               %. . .%
            end Write_Data;

         or

            accept Read_Data (A_Value : out Hardware_Interface.Hw_Integer) is
               %. . .%
            end Read_Data;

         or

            terminate;

      end select;

     end loop;

   exception
     when others =>
       %. . .%
   end Monitor_Task;
end Monitor_1;
```

B.8 Procesos de segundo plano

Los procesos de segundo plano son tareas con prioridad menor que cualquier proceso de tiempo real y cuya ejecución es independiente del modo de ejecución del sistema. Sólo se pueden comunicar con otros procesos de segundo plano, ya que en otro caso podrían incurrir en inversiones de prioridad inesperadas.

A continuación se muestra esquemáticamente como quedaría PSP_1.

```
--/*****************************************************************
--/ package specification Background_Process_Package_1
--/*****************************************************************

package Background_Process_Package_1 is

end Background_Process_Package_1;

--/*****************************************************************
--/ package body %Background_Process_Package_Name%
--/*****************************************************************

with Run_Time_System_Interface;
package body Background_Process_Package_1 is


   --/**************************************************************
   --/ task Background_Task_1
   --/**************************************************************

   task Background_Task_1;

   task body Background_Task_1 is

      Process_Priority : Run_Time_System_Interface.Priority_Type := 88,

   begin

     Run_Time_System_Interface.Set_Priority (Process_Priority);

      %. . .%

     exception
        when others =>
             %...%
   end Background_Task_1;

end  Background_Process_Package_1;
```

B.9 Manejadores de interrupciones

Los manejadores de interrupciones se desarrollan a partir del componente `Interrupt_Handler_Shell`. Es importante identificar correctamente la interrupción a la que se asocia.

La ejecución de los manejadores de interrupciones produce inversión de prioridad, ya que según el modelo Ada, ejecutarán con la prioridad máxima. Por consiguiente el tratamiento de la interrupción debe ser tan breve como sea posible. La aproximación adecuada según la teoría de planificación es detectar la interrupción con el manejador correspondiente, realizar el tratamiento mínimo necesario y comunicar el evento al proceso correspondiente, que tratará el evento con la prioridad adecuada.

En este ejemplo se muestra la creación del manejador de interrupciones denominado anteriormente MI_1 y el nombre que se le asigna en la aplicación es `Interrupt_Handler_1`. La creación del otro manejador es análoga. El servidor esporádico SE_1 trata el evento asociado con la interrupción que trata MI_1.

```
--/*****************************************************************
--/ package specification Interrupt_Handler_Package_1
--/*****************************************************************

with System;
with Sporadic_Server_Process_Package_1;

package Interrupt_Handler_Package_1 is
   --/*************************************************************
   --/ task Interrupt_Handler_Task_1
   --/*************************************************************

   task Interrupt_Handler_Task_1 is

       entry Interrupt_Entry_1;
       for Interrupt_Entry_1 use at System.To_Address (16#140#);

   end Interrupt_Handler_Task_1;

end Interrupt_Handler_Package_1;

--/*****************************************************************
--/ package body Interrupt_Handler_Package_1
--/*****************************************************************

package body Interrupt_Handler_Package_1 is

   funtion Get_Byte return Sporadic_Server_Process_Package_1.Aperiodic_Event_Type is
       . . . .
    begin
       . . . .
    end Get_Byte;
```

```
   --/*************************************************************
   --/ task Interrupt_Handler_Task_1
   --/*************************************************************

   task body Interrupt_Handler_Task_1 is

      Interrupt_Time : Calendar.Time;

   begin

      %. . .%

    loop

      %. . .%

    accept Interrupt_Entry_1 do

      Interrupt_Time := Calendar.Clock;
      %. . .%
      Sporadic_Server_Process_Package_1.Signal_Aperiodic_Event
           (Event_Information => Get_Byte,
           (Event_Time        => Interrupt_Time);
      %. . .%
    end;

    end loop;

     exception
       when others =>
       %. . .%
   end Interrupt_Handler_Task_1;

end  Interrupt_Handler_Package_1;
```

B.10 Procedimiento Principal

El procedimiento principal es un procedimiento sin parámetros que usa los paquetes que incluyen la definición de procesos.

```
with Application_Supervisor;
with Recovery_Process_Package_1;
with Recovery_Process_Package_2;
with Periodic_Process_Package_1;
with Periodic_Process_Package_2;
with Periodic_Process_Package_3;
with Periodic_Process_Package_4;
with Sporadic_Server_Process_Package_1;
with Sporadic_Server_Process_Package_2;
```

```
with Monitor_Package_1;
with Monitor_Package_2;
with Background_Process_Package_1;
with Background_Process_Package_2;
with Interrupt_Handler_Package_1;
with Interrupt_Handler_Package_2;

procedure Main_Aplication_Procedure is
  begin
    null;
  end Main_Aplication_Procedure;
```

Bibliografía.

[Ada9XAnS92] *Ada 9X Snapshot of Annexes Prior to Revision.* Intermetrics, Inc., Octubre 1992.

[Ada9XIntro93] *Introducing Ada 9X.* Intermetrics, Inc., Febrero 1993.

[Ada9xMap92] *Ada 9X Mapping. Version 4.0.* Intermetrics, Inc., Marzo 1992.

[Ada9XMaS92] *Ada 9X Snapshot of Mapping Specification Prior to Revision. Version 4.6.* Intermetrics, Inc., Junio 1992.

[Ada9XRM93] *Ada 9X Reference Manual. Draft. Version 4.0.* Intermetrics, Inc., Junio 1993.

[AdaLRM83] *Military Standard Ada Programming Language.* American National Standard Institute, Inc., Enero 1983. ANSI/MIL-STD-1815A.

[AdaPI90] ACM, editor. *Ada Performance Issues.* Volume X, 3, Ada Letters Special Issue, 1990.

[Agrawal&92] G. Agrawal, B. Chen, W. Zhao, S. Davari. Guaranteeing Synchronous Message Deadlines in High Speed Networks with Timed Token Protocol. In *Proceedings of IEEE Internationsl Conference on Distributed Computing Systems*, 1992.

[Alonso&90] A. Alonso, L. Gómez, J. Zamorano, J.A. de la Puente. Sistemas de tiempo real y Ada. *Novática*, 16(89), 1990.

[Alonso&91] Alejandro Alonso, Juan Luis Redondo, José Ignacio Tortosa, Gonzalo Génova, Juan Antonio de la Puente. *Ejecutivo multitarea. Componentes reutilizables.* Julio 1991.

[Alonso&92a] Alejandro Alonso, Juan A. de la Puente. Implementing Mode Changes and Fault Recovery for Hard Real-Time Systems in Ada. In *International Workshop on Real-Time Programming*, 1992.

[Alonso&92b] Alejandro Alonso, Juan A. de la Puente. Reusable Real-Time Executive in Ada. In *6th International Workshop on Real-Time Ada Issues*, 1992.

[Alonso&92c] Alejandro Alonso, Juan Luis Redondo, Carlos Blanco, José Ignacio Tortosa, Juan Antonio de la Puente. *Ejecutivo multitarea. Manual de Usuario.* Enero 1992.

[Alonso&93] Alejandro Alonso, Juan A. de la Puente. Dynamic Replacement of software in hard real-time systems. In *Proceedings of Euromicro'93 Workshop on Real-Time Systems*, 1993.

[Amador&88] J. Amador, J. Campos. *SWRU Concept for ARTT.* COL-GMV-ARTT2-TNSWRU, GMV, Septiembre 1988.

[Amador&91] Jorge Amador, Belén de Vicente, Alejandro Alonso. Dynamically Replaceable Software: A Design Method. In *European Software Engineering Conference*, 1991.

[Astrom&90] Karl Astrom, Bjorn Wittenmark. *Computer-Controlled Systems.* Prentice-Hall, 1990.

[Atkinson&88] C. Atkinson, T. Moreton, A. Natali. *Ada for Distributed Systems.* Cambridge University Press, 1988.

[Audsley&91] N.C. Audsley, A. Burns, M.F. Richardson, A.J. Wellings. Hard Real-Time Scheduling: The Deadline-Monotonic Approach. In *Eigth IEEE Workshop on Real-Time Operating Systems and Software, in conjunction with IFAC/IFIP Workshop on Real-Time Programming*, 1991.

[Audsley91a] N.C. Audsley. *The Deadline-Monotonic Approach.* YCS 146, University of York. Department of Computer Science, 1991.

[Audsley91b] N.C. Audsley. *Resource Control for Hard Real-Time Systems: A Review.* YCS 159, University of York. Department of Computer Science, 1991.

[Audsley&92] N.C. Audsley, A. Burns, M.F. Richardson, A. Wellings. Deadline Monotonic Scheduling Theory. In *International Workshop on Real-Time Programming*, 1992.

[Bae91] British Aerospace Systems Limited. *Hard Real-Time Operating Systems Kernel.* Task 6 Deliverable on ESTEC Contract 9198/90/NL/SF, B.A., 1991.

[Baker&89] T.P. Baker, A. Shaw. The cyclic executive model and Ada. *Real-Time Systems*, 1(1), 1989.

[Bal90] H.E. Bal. *Programming distributed systems.* Silicon Press, 1990.

[Barnes89] John Barnes. *Programming in Ada.* Addison Wesley, third edition, 1989.

[Bellcore93] Bellcore Information Networking Laboratory. The Touring Machine System. *Communications of the ACM*, 36(1), Enero 1993.

[Booch87] Grady Booch. *Software components with Ada.* Benjamin Cummings, 1987.

[Borger&89] M. Borger, M. Klein, R. Veltre. *Real-Time Software Engineering in Ada: Observations and Guidelines.* Technical Report CMU/SEI-89-TR-22;ESD-TR-89-30, Carnegie-Mellon University, Software Engineering Institute, 1989.

[Burns&87b] Alan Burns, Andy J. Wellings. Real-Time Ada Issues. *Ada Letters*, 7(6), 1987.

[Burns&89] A. Burns, A.J. Wellings. *Real-time systems and their programming languages.* Addison-Wesley, 1989.

[Burns&91] A. Burns, A.J. Wellings. *Hard Real-Time Operating Systems Kernel.* Task 1 Deliverable on ESTEC Contract 9198/90/NL/SF, Department of Computer Science, University of York, UK, Septiembre 1991.

[Burns&91a] A. Burns, A.J. Wellings. *Extending the Application of the Hard Real-Time Theory.* Task 2 Deliverable on ESTEC Contract 9198/90/NL/SF, Department of Computer Science, University of York, UK, Septiembre 1991.

[CASA&91a] Juan Luis Freniche, Luis Redondo, Enrique Martín. *Especificación de requerimientos software para los ejecutivos reusables Ada de aviónica.* 1991. Proyecto Biblioteca de Componentes Ada.

[Chandrasekaran&91] B. Chandrasekaran, R.Bhatnagar, D.Sharma. Real-Time Disturbance Control. *Communications of the ACM*, 34(8), 1991.

[Chen&89] M.I. Chen, K.J. Lin. *Dynamic Priority Ceilings: A concurrency control protocol for Real-Time Systems.* Technical Report, Dept. of Computer Science, University of Illinois at Urbana-Champaign, Abril 1989.

[Chetto&90] H. Chetto, M. Silly, T. Bouchentouf. Dynamic Scheduling of Real-Time Tasks under Precedence Constraints. *Real-Time Systems*, 2(3), Septiembre 1990.

[Cornhill&87a] D. Cornhill, L. Sha, J.P.Lehoczky, R. Rajkumar, H. Tokuda. Limitations of Ada for real-time scheduling. In *1st International Workshop on Real-Time Ada Issues*, 1987.

[Crespo93] Juan Carlos Crespo. *Metodología de diseño de sistemas con unidades reemplazables de software.* ETSI de Telecomunicación. Universidad Politécnica de Madrid, 1993.

[Damm&88] A. Damm, J. Reisinger, W. Schwabl, H. Kopetz. *The Real-Time Operating System of MARS.* Technical Report 17/88, Technische Universität Wien, 1988.

[Donhoe&90] P.Donhoe, R.Shapiro. *Hartstone Benchmark Users Guide.* Technical Report, Carnegie-Mellon University, Software Engineering Institute, 1990.

[Dorf89] R. Dorf. *Sistemas Modernos de Control. Teoría y Práctica.* Addison-Wesley Iberoamericana, 1990.

[Etzkorn92] Georg Etzkorn. Change Programming in Distributed Systems. In Jeff Kramer, editor, *Proceedings of the International Workshop Configurable Distributed Systems*, Institution of Electrical Engineers, UK, 1992.

[Fernández&90a] José Luis Fernández, Juan Antonio de la Puente, Alejandro Alonso, Gonzalo Alonso. *Manual de diseño de componentes.* 1990. Proyecto Biblioteca de Componentes Ada.

[Fernández&91] Jose L. Fernández, Juan A. de la Puente. Constructing a Pilot Library of Components for Avionic Systems. In Dimitris Christodoulakis, editor, *Ada: The Choice for '92*, Ada Europe, Springer-Verlag, 1991.

[Fernández&93] José Luis Fernández, Juan de la Puente, Alejandro Alonso, Nicolás Súarez, Juan Zamorano. Reusable Software Components and Executives for Real-Time Systems. In *Second International Workshop on Software Reuse*, 1993.

[Futurebus91] *Futerebus P896. 1,2,3 Specifications.* IEEE, 1991.

[Gafford90] J.D. Gafford. Rate Monotonic Scheduling. *IEEE Micro*, 1990.

[Gayakwad&88] R.Gayakwad, L.Sokoloff. *Analog and Digital Control Systems.* Prentice-Hall, 1988.

[Goldberg&89] M. Goldberg, M. Georganas, J. Roberston, J. Mastronardi, S. Reed. A prototype multimedia radiology communication system. In *Proceedings of the Second IEEE International Workshop on Multimedia Communications*, Abril 1989.

[González&91] Michael González Harbour, Lui Sha. *An Application-Level Implementation of the Sporadic Server.* Technical Report CMU/SEI-91-TR-26;ESD-91-TR-26, Carnegie-Mellon University, Software Engineering Institute, 1991.

[González&91a] M. González, M.Klein, J.Lehoczky. Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority. In *IEEE Real-Time Systems Symposium*, 1991. San Antonio, USA.

[Goodenough&88] J.B. Goodenough, L. Sha. The priority ceiling protocol: a method for minimizing the blocking of high priority Ada tasks. In *Second International Workshop on Real-Time Ada Issues*, ACM SIGAda, 1988. Ada Letters, 8(7).

[Gopinath&89] P. Gopinath, K. Schwan. CHAOS: Why One Cannot Have Only an Operating System for Real-Time Applications. *Operating System Review. ACM Press*, 23(3), Julio 1989.

[Goscinski91] A. Goscinski. *Distributed Operting Systems. The Logical Design.* Addison-Wesley Publishing Company, 1991.

[Greenberg90] S. Greenberg. Sharing views and interactions with single-user applications. In *Proceedings of the Conference on Office Information Systems (COIS'90)*, Abril 1990.

[Hood89a] *HOOD User Manual Issue 3.0.* European Space Agency, Diciembre 1989.

[Hood89b] *HOOD Reference Manual Issue 3.0.* European Space Agency, Septiembre 1989.

[IEEE 802.5] *802.5: Token Ring Access Method.* IEEE, 1985.

[Kandlur&89] D. Kandlur, D. Kiskis, K. Shin. HARTOS: A Distributed Real-Time Operating Systems. *ACM Operating Systems Review*, 23(3), 1989.

[Kernighan&84] B. W. Kernighan, R. Pike. *The UNIX Programming Environment.* Prentice Hall, 1984.

[Klein&90] M.H. Klein, Thomas Ralya. *An analysis of Input/Output Paradigms for Real-Time Systems.* Technical Report CMU/SEI-90-TR-19;ESD-90-TR-220, Carnegie-Mellon University, Software Engineering Institute, 1990.

[Kopetz&89] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schawbl, C. Senft, R. Zainlinger. Distributed Fault-Tolerant Real-Time Systems: The MARS approach. *IEEE Micro*, 9(1), Febrero 1989.

[Kopetz&89a] H. Kopetz, G. Grünsteidl, J. Reisinger. *Fault-Tolerant Membership Service in a Synchronous Distributed Real-Time System.* Technical Report 4/89, Technische Universität Wien, 1989.

[Kopetz&89b] H. Kopetz, W. Schwabl. *Global Time in Distributed Real-Time System.* Technical Report 15/89, Technische Universität Wien, 1989.

[Kopetz&89c] H. Kopetz, H. Kantz, J. Reisinger G. Grünsteidl, P.Pushchner. *Tolerating Transient faults in MARS.* Technical Report 18/89, Technische Universität Wien, 1989.

[Kopetz&91] H. Kopetz, R. Zainlinger, G. Fohler, H. Kantz, P. Puschner, W. Schütz. An Engineering Approach to Hard Real-TIme Systems Design. In *European Software Engineering Conference*, 1991.

[Kopetz&92] H. Kopetz, G. Fohler, G. Grünsteidl, H. Kantz, G. Pospischil, P. Puschner, J. Reisinger, R. Schlatterbeck, W. Schütz, A. Vrchoticky, R. Zainlinger. The Programmer's View of MARS. In *IEEE Real-Time Systems Symposium*, 1992.

[Kramer92] Jeff Kramer, editor. *Proceedings of the International Workshop Configurable Distributed Systems*, Institution of Electrical Engineers, UK, 1992.

[Laarhoven&87] P. Laarhoven, E. Aaarts. *Simulated Annealing: Theory and Applications.* D. Reidel Publishing, 1987.

[Lehoczky&87] J.P.Lehoczky, L. Sha, J. Strosnider. Enhancing aperiodic responsiveness in a hard real-time environment. In *IEEE Real-Time Systems Symposium*, 1987.

[Lehoczky&89] J.P.Lehoczky, L. Sha, Y. Ding. The Rate Monotonic Scheduling Algorithm : Exact Characterization and Average Case Behavior. In *IEEE Real-Time Systems Symposium*, 1989.

[Lehoczky90] J.P.Lehoczky. Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. In *IEEE Real-Time Systems Symposium*, 1990.

[Lehoczky&91] J. Lehoczky, L. Sha, J.K. Strosnider, Hide Tokuda. Fixed Priority Scheduling Theory for Hard Real-Time Systems. In A.M. van Tilborg, G.M. Koob, editors, *Foundations of Real-Time Computing. Scheduling and Resource Management*, Kluwer Academic Publishers, 1991.

[Lenstra&77] J.K. Lenstra, A.H.G. Rinnooy Kan, P. Bruchker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, (1), 1977.

[Leung&82] J.Y.T. Leung, J. Whitehead. On the complexity of fixed-priority of Periodic Real-Time Tasks. *Performance Evaluation*, 2(4), 1982.

[Levi&89] S. Levi, S. Tripathi, S. Carson, A. Agrawala. The MARUTI Hard Real-Time Operating System. *Operating System Review. ACM Press*, 23(3), Julio 1989.

[Liu&73] C.L. Liu, J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1), 1973.

[Locke92] C.D. Locke. Software Architecture for Hard Real-Time Applications: Cyclic Executives vs. Fixed Priority Executives. *Real-Time Systems*, 4(1), Marzo 1992.

[Magee&89] J. Magee, J. Kramer, M. Sloman. Constructing Distributed Systems in CONIC. *IEEE Transactions on Software Engineering*, 15(6), Junio 1989.

[Mercer&90] C. Mercer, H. Tokuda. The ARTS Real-Time Objecto Model. In *IEEE Real-Time Systems Symposium*, 1990.

[Middleton89] D.H. Middleton, editor. *Avionic Systems*. Longman Scientific & Technical, 1989.

[Mok83] Aloysius. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*. PhD thesis, MIT, 1983.

[Mok84a] A.K. Mok. The design of real-time programming systems based on process models. In *IEEE Real-Time Systems Symposium*, 1984.

[Mullender89] S. Mullender, editor. *Distributed Systems.* ACM Press, 1989.

[Nicolau90] C. Nicolau. An Architecture for Real-Time Multimedia Communication Systems. *IEEE Journal on Selected Areas in Communications*, 8(3), Abril 1990.

[Northcutt87] J. Duane Northcutt. *Mechanisms for Reliable Distributed Real-Time Operating Systems – The Alpha Kernel.* Academic Press, 1987.

[Ogata87] Katsuhiko Ogata. *Discrete-Time Control Systems.* Prentice-Hall, 1987.

[Ogata90] Katsuhiko Ogata. *Modern Control Engineering.* Prentice-Hall, 1990.

[Olsson&92] G. Olsson, G. Piani. *Computer Systems for Automation and Control.* Prentice Hall International, 1992.

[Payton&91] D. Payton, T. Bihari. Intelligent Real-Time Control of Robotic Vehicles. *Communications of the ACM*, 34(8), 1991.

[Peterson&89] James Peterson, Abraham Silberschatz. *Operating Systems Concepts.* Addison-Wesley, third edition, 1989.

[Pleinevaux92] P. Pleinevaux. An Improved Hard Real-Time Scheduling for the IEEE 802.5. *Real-Time Systems*, 4(2), Junio 1992.

[POSIX.1 90] ISO. *ISO/IEC 9945-1, Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].* ISO, 1990.

[POSIX.13 92] IEEE. *POSIX.13 Standarized Application Environment Profile - POSIX Realtime Application Support.* IEEE, 1992.

[POSIX.1b 93] *Real-Time extensions to POSIX. IEEE Standard P1003.4.* IEEE, Marzo 1993.

[POSIX.1c 93] *Threads Extension for Portable Operating Systems to POSIX. IEEE Standard P1003.4a.* IEEE, Abril 1993.

[Puente&91a] Juan Antonio de la Puente, Alejandro Alonso, Juan Luis Redondo, Juan Zamorano, José Ignacio Tortosa, Gonzalo Génova, Carlos Blanco. *Ejecutivos Reutilizables para Aviónica. Especificación de Componentes.* Marzo 1991.

[Puente&91b] Juan Antonio de la Puente, Alejandro Alonso, Juan Luis Redondo, Juan Zamorano, José Ignacio Tortosa, Gonzalo Génova, Carlos Blanco. *Ejecutivos Reutilizables para Aviónica. Casos de Prueba.* Abril 1991.

[Puente&92] Juan de la Puente, Juan Zamorano, Alejandro Alonso, José Luis Fernández. Reusable Executives for Hard Real-Time Systems in Ada. In Jan van Katwijk, editor, *Ada: moving towards 2000*, Ada Europe, Springer-Verlag, 1992.

[Puente&92a] Juan de la Puente, Juan Zamorano, Alejandro Alonso, José Luis Fernández. Ejecutivos reutilizables para desarrollo de sistemas de tiempo real en Ada. In *II Congreso INTA*, 1992.

[Rajkumar&88] R. Rajkumar, L. Sha, J.P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *IEEE Real-Time Systems Symposium*, 1988.

[Rajkumar&89] R. Rajkumar, L. Sha. Accomplishing mode changes in Ada. In *3rd International Workshop on Real-Time Ada Issues*, 1989.

[Rajkumar91] R. Rajkumar. *Synchronization in Real-Time Systems. A priority inheritance approach.* Kluwer Academic Publishers, 1991.

[Ramamritham&84] K. Ramamritham, J.A. Stankovic. Dynamic task scheduling in hard real-time distributed systems. *IEEE Software*, Julio 1984.

[Ramamritham&89] K. Ramamritham, J.A. Stankovic, W. Zhao. Distributed Scheduling of Tasks with Deadlines and Resource Requirements. *IEEE Transctions on Computers*, 38(8), 1989.

[Ready86] J.F. Ready. VRTX: A real-time operating system for embedded microprocessor applications. *IEEE Micro*, Agosto 1986.

[Reisinger89] J. Reisinger. *Inside the MARS Operating System.* Technical Report 7/89, Technische Universität Wien, 1989.

[Rodd&89] M.G. Rodd, F.Deravi. *Communication systems for industrial automation.* Prentice Hall International, 1989.

[Sanz90] Ricardo Sanz Bravo. *Arquitectura de control inteligente de procesos.* PhD thesis, ETSI de Industriales. Universidad Politécnica de Madrid, 1990.

[Schooler91] E. Schooler. *A distributed Architecturefor multimedia conference control.* Technical Report ISI/RR-91-289, ISI, Noviembre 1991.

[Sha&86] L. Sha, J.P. Lehoczky, R. Rajkumar. Solutions for some practical problems in prioritized preemptive scheduling. In *IEEE Real-Time Systems Symposium*, 1986.

[Sha&86a] L. Sha, J.P. Lehoczky, R. Rajkumar. *Integrated Approaches to processor and I/O Scheduling in the presence of transient overloads and stochastic execution times.* Technical Report , Carnegie-Mellon University, 1986.

[Sha&87] Lui Sha, Ragunathan Rajkumar, John P. Lehoczky. *Priority Inheritance Protocols, An Approach to Real-Time Synchronization.* Technical Report CMU-CS-87-181, Department of CS, ECE and Statistics, Carnegie Mellon University, 1987.

[Sha&89a] Lui Sha, John B. Goodenough. *Real-Time Scheduling Theory and Ada.* Technical Report CMU/SEI-89-TR-14;ESD-TR-89-22, Carnegie-Mellon University, Software Engineering Institute, 1989.

[Sha&89c] L. Sha, R. Rajkumar, J.P. Lehoczky, K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1(3), 1989.

[Sha&90] Lui Sha, John B. Goodenough. Real-Time Scheduling Theory and Ada. *IEEE Computer*, 23(4), 1990.

[Sha&91a] L. Sha, M.H. Klein, J.B. Goodenough. Rate Monotonic Analysis for Real-Time Systems. In A.M. van Tilborg, G.M. Koob, editors, *Foundations of Real-Time Computing. Scheduling and Resource Management*, Kluwer Academic Publishers, 1991.

[Sha&91b] Lui Sha, R. Rajkumar, J.P. Lehoczky. Real-Time Applications Using IEEE Futurebus+ Synchronization. *IEEE Micro*, 10(3), Junio 1991.

[Sha&93] L. Sha, S. Sathaye. *Distributed Real-Time System Design: Theoretical Concepts and Applications*. Technical Report CMU/SEI-93-TR-2, Carnegie-Mellon University, Marzo 1993.

[Silva85] M. Silva. *Las redes de Petri en la Automática y en la Informática*. Editorial AC, 1985. In Spanish.

[Smith88] R. Smith. A prototype futuristic techology for distance education. In *NATO Research Workshop on New Directions in Education Technology*, 1988.

[Sommerville89] I. Sommerville. *Software Engineering*. Addison-Wesley, 3 edition, 1989.

[Spitzer87] C. Spitzer. *Digital Avionic Systems*. Prentice-Hall, 1987.

[Sprunt&88] B. Sprunt, J. Lehoczky, L. Sha. Exploiting unused time for aperiodic service using the extended priority exchange algorithm. In *IEEE Real-Time Systems Symposium*, 1988.

[Sprunt&89] B. Sprunt, L. Sha, J.P. Lehoczky. Aperiodic task scheduling for hard real-time systems. *Real-Time Systems*, 1(1), 1989.

[Stankovic&87] J.A. Stankovic, K. Ramamritham. The design of the spring kernel. In *IEEE Real-Time Systems Symposium*, 1987.

[Stankovic88a] J.A. Stankovic. Real-time computer systems: the next generation. In K. Ramamrithan J.A. Stankovic, editor, *Hard real-time systems*, IEEE CS Press, 1988.

[Stankovic&88b] J.A. Stankovic, K. Ramamrithan, editors. *Hard real-time systems*. IEEE CS Press, 1988.

[Stankovic88a] J.A. Stankovic. Misconceptions about real-time programming. *IEEE Computer*, Octubre 1988.

[Stankovic&89] J.A. Stankovic, K. Ramamritham. The Spring Kernel: A New Paradigm for Real-Time Operating Systems. *ACM Operating Systems Review*, 23(3), 1989.

[Stankovic90] J.A. Stankovic. The Spring Architecture. *IEEE*, 23(3), 1990.

[Stankovic&91] J.A. Stankovic, K. Ramamritham. The Spring Kernel: A New Paradigm for Real-Time Systems. *IEEE Software*, 23(3), 1991.

[Strosnider&89] J. Strosnider, T. Marchok. Responsive, Determinstic IEEE 802.5 Token Ring Scheduling. *Real-Time Systems*, 1(2), Septiembre 1989.

[Syms&91] T. Syms, C.L.Braun. Software Reuse: Customer vs. Constractor Point-Counterpoint. In Dimitris Christodoulakis, editor, *Ada: The Choice for '92*, Ada Europe, Springer-Verlag, 1991.

[Tanenbaum&85] A. Tanenbaum, R. van Renesse. distributed Operating Systems. *ACM Computing Surveys*, 17(4), Diciembre 1985.

[Tanenbaum88] A. Tanenbaum. *Computer Networks*. Prentice-Hall, second edition, 1988.

[Tanenbaum92] A. Tanenbaum. *Modern Operating Systems*. Prentice-Hall, 1992.

[Tindell90] K. Tindell. Dynamic Code Replacement and Ada. *Ada Letters*, X(7), 1990.

[Tindell&92] K. Tindell, A.Burns, A. Wellings. Mode Changes in priority pre-emptively scheduled Systems. In *IEEE Real-Time Systems Symposium*, 1992. Phoenix, USA.

[Tindell&92a] K. Tindell, A.Burns, A. Wellings. Allocating Hard Real-Time Tasks: An NP-Hard Problem Made Easy. *Real-Time Systems*, 4(2), Junio 1992.

[Tokuda&89] H. Tokuda, C. Mercer. ARTS: A distributed Real-Time Kernel. *Operating System Review. ACM Press*, 23(3), Julio 1989.

[Tokuda&90] H. Tokuda, T. Nakajima, P. Rao. Real-Time Mach: Towards a Predictable Real-Time System. In *Proceedings Usenix Mach Workshop*, Octubre 1990.

[Velasco91] J. Ramón Velasco Pérez. *Arquitectura para sistemas de control inteligentes*. PhD thesis, ETSI de Telecomunicación. Universidad Politécnica de Madrid, 1991.

[Vicente90] Belén de Vicente. *Metodología de diseño de sistemas con unidades reemplazables de software*. Facultad de Informática. Universidad Politécnica de Madrid, 1990.

[Vicente&91] Belén de Vicente, Alejandro Alonso, Jorge Amador. Dynamic Software Replacement Model and its Ada implementation. In *Tri-Ada'91*, 1991.

[Vin&91] H. Vin, P. Zellweger, D. Swinehart, P. Rangan. Multimedia Conferencing in the Etherphone Environment. *IEEE Computer*, 24(10), Octubre 1991.

[Ward&85] P.T. Ward, S.J. Mellor. *Structured development for real-time systems*. Volume 1, 2, 3, Yourdon Press, 1985.

[Ward86] Paul T. Ward. The Transformation Schema: An Extension of the Data Flow Diagram to represent Control and Timing. *IEEE Transactions on Software Engineering*, 12(2), 1986.

[Wedde&89] H. Wedde, G. Alijani, W. Brown, S. Chen, G. Kang, B. Kim. Operating System Support for Adaptative Distributed Real-Time Systems in DRAGON SLAYER. *Operating System Review. ACM Press*, 23(3), Julio 1989.

[Zamorano&92] Juan Zamorano, Juan Luis Redondo, Carlos Blanco, José Ignacio Tortosa, Juan Antonio de la Puente. *Ejecutivo cíclico. Manual de Usuario*. Enero 1992.

[Zhao&87a] W. Zhao, K. Ramamritham, J.A. Stankovic. Scheduling tasks with resource requirements in hard real-time systems. *IEEE Transactions on Software Engineering*, 13(5), 1987.

[Zhao&87b] W. Zhao, K. Ramamritham, J.A. Stankovic. Preemptive Scheduling Under Time and Resource Constraints. *IEEE Transctions on Computers*, C-36(8), 1987.

Curriculum Vitae

Alejandro Alonso Muñoz nació en Madrid, el 13 de Junio de 1961. Los cursos de EGB, BUP y COU los realizó en Madrid. Entre los años 1979 y 1985 estudia en la Facultad de Informática de la Universidad Politécnica de Madrid.

En 1985 obtiene una plaza de profesor en la Facultad de Informática de Madrid. En 1987 pasa a formar parte del Departamento de Arquitectura y Tecnología de Sistemas Informáticos en el mismo centro. En 1991 obtiene una plaza de profesor en el Grupo de Ingeniería de Control perteneciente al Departamento de Tecnologías Especiales Aplicadas a la Telecomunicación en la ETSI de Telecomunicación de Madrid. Durante su actividad docente ha impartido cursos sobre sistemas digitales, sistemas operativos, sistemas de tiempo real, control de procesos y, en la actualidad, es responsable del laboratorio de Servomencanismos II, relativo a sistemas de control digitales.

Ha participado en diversos proyectos de investigación, entre los que destacan *Planificación de procesos en sistemas informáticos con restricciones de tiempo real críticas* y *Técnicas avanzadas de desarrollo de software para el control de procesos con requisitos de seguridad críticos*, en el proyecto industrial *Biblioteca de Componentes Ada* y en los proyectos ESPRIT: IPTES (*Incremental Prototyping Technology for Embedded Real-Time Systems*) e IDERS (*Integrated Development Environment for Embedded Real-Time Systems with Complete Process and Product Visibility*).

Es autor, junto con otros investigadores, de varios artículos publicados en revistas y congresos nacionales e internacionales.